Advanced Topics in Machine Learning 2017-2018

Yevgeny Seldin Christian Igel Tobias Sommer Thune Julian Zimmert

Home Assignment 6

Deadline: Tuesday, 24 October 2017, 23:59

The assignments must be answered individually - each student must write and submit his/her own solution. We encourage you to work on the assignments on your own, but we do not prevent you from discussing the questions in small groups. If you do so, you are requested to list the group partners in your individual submission.

Submission format: *Please, upload your answers in a single* `.pdf` *file and additional* `.zip` *file with all the code that you used to solve the assignment. (The* `.pdf` *should* ***not*** *be part of the* `.zip` *file.)*

IMPORTANT: *We are interested in how you solve the problems, not in the final answers. Please, write down the calculations and comment your solutions.*

1 Feature space dimensionality (35 points)

The representer theorem tells us that the canonical SVM optimization problem for n data points has a solution that can be written in the form of a kernel expansion plus bias parameter with $n+1$ real valued coefficients. But do we really always need so many parameters?

1. Assume that the input space is the $\mathbb{R}^d$. If a linear kernel is used, what is the minimum number of parameters guaranteed to be sufficient to define the SVM solution for arbitrary training sets with n data points?

2. In general, how many parameters are sufficient if the dimensionality of the kernel-induced feature space is d'? Again, give a bound that holds for any training data set with n points.

3. Let $d' < n$. Show that the Gram matrix is not *strictly* positive definite (i.e., is only positive semi-definite). You can use the following hint. Let the data set be $\{(\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_n, y_n)\}$. Let k be the kernel, Φ the kernel induced feature map, and $\boldsymbol{K}$ be the $n \times n$ Gram matrix. Then for an arbitrary vector $\boldsymbol{z} = (z_1, \ldots, z_n)^{\mathrm{T}} \in \mathbb{R}^n$ we have

$$\boldsymbol{z}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{z} = \sum_{i,j=1}^{n} k(\mathbf{x}_j, \mathbf{x}_i) z_i z_j = \sum_{i,j=1}^{n} \langle \Phi(\mathbf{x}_j), \Phi(\mathbf{x}_i) \rangle z_i z_j = \sum_{i,j=1}^{n} \langle z_j \Phi(\mathbf{x}_j), z_i \Phi(\mathbf{x}_i) \rangle$$
$$= \left\langle \sum_{j=1}^{n} z_j \Phi(\mathbf{x}_j), \sum_{i=1}^{n} z_i \Phi(\mathbf{x}_i) \right\rangle = \ldots$$

4. If a Gaussian kernel is used, the Gram matrix always has full rank assuming $\mathbf{x}_i \neq \mathbf{x}_j$ for $i \neq j$. What is the minimum number of parameters guaranteed to be sufficient to define the SVM solution for arbitrary training sets with n data points if a Gaussian kernel is used?

2 The doubling trick (30 points)

Consider the following forecasting strategy (the "doubling trick"): time is divided into periods $(2^m, \ldots, 2^{m+1} - 1)$, where $m = 0, 1, 2, \ldots$. (In other words, the periods are $(1), (2, 3), (4, \ldots, 7), (8, \ldots, 15), \ldots$.) In period

$(2^m, \dots, 2^{m+1}-1)$ the strategy uses the optimized Hedge forecaster (from Home Assignment 5) initialized at time 2^m with parameter $\eta_m = \sqrt{\frac{8 \ln N}{2^m}}$. Thus, the Hedge forecaster is reset at each time instance that is an integer power of 2 and is restarted with a new value of η. By the analysis of optimized Hedge we know that with $\eta_m = \sqrt{\frac{8 \ln N}{2^m}}$ its expected regret within the period $(2^m, \dots, 2^{m+1}-1)$ is bounded by $\sqrt{\frac{1}{2} 2^m \ln N}$.

1. Prove that for any $T = 2^m - 1$ the overall expected regret (considering the time period $(1, \dots, T)$) of this forecasting strategy satisfies

$$\mathbb{E}[R_T] \leq \frac{1}{\sqrt{2}-1} \sqrt{\frac{1}{2} T \ln N}.$$

   (Hint: at some point in the proof you will have to sum up a geometric series.)

2. Prove that for any arbitrary time T the expected regret of this forecasting strategy satisfies

$$\mathbb{E}[R_T] \leq \frac{\sqrt{2}}{\sqrt{2}-1} \sqrt{\frac{1}{2} T \ln N}.$$

Remark: The expected regret of "anytime" Hedge with $\eta_t = 2\sqrt{\frac{\ln N}{t}}$ satisfies $\mathbb{E}[R_T] \leq \sqrt{T \ln N}$ for any T. For comparison, $\frac{1}{\sqrt{2}(\sqrt{2}-1)} \approx 1.7$ and $\frac{1}{\sqrt{2}-1} \approx 2.4$. Thus, anytime Hedge is both more elegant and more efficient than Hedge with the doubling trick.

3 Empirical evaluation of UCB1 and EXP3 algorithms (35 points)

Implement and compare the performance of UCB1 and EXP3 in the i.i.d. multiarmed bandit setting. For EXP3 take time-varying $\eta_t = \sqrt{\frac{\ln K}{tK}}$. You can use the following settings:

- Time horizon $T = 10000$.
- Take a single best arm with bias $\mu^* = 0.5$.
- Take $K - 1$ suboptimal arms for $K = 2, 4, 8, 16$.
- For suboptimal arms take $\mu = \mu^* - \frac{1}{4}$, $\mu = \mu^* - \frac{1}{8}$, $\mu = \mu^* - \frac{1}{16}$ (3 different experiments with all suboptimal arms being equally bad).

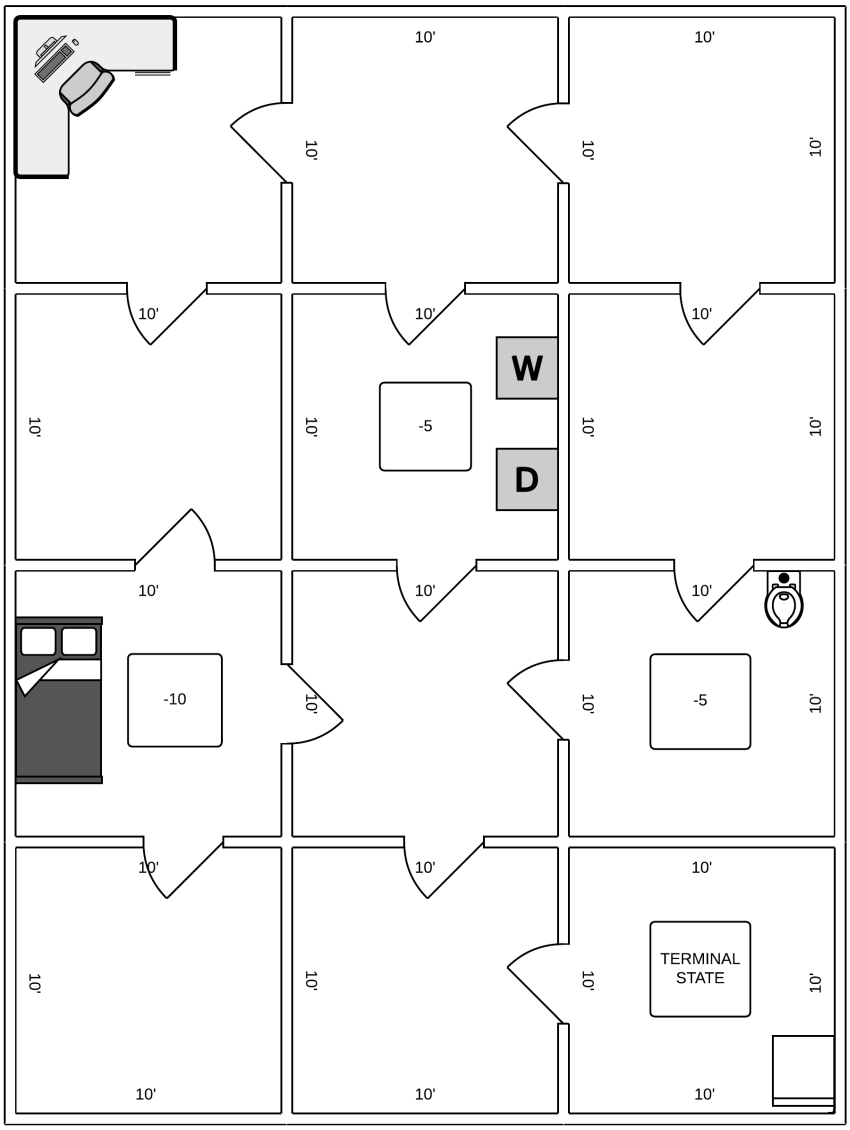
Make 10 repetitions of each experiment and for each experiment plot the average pseudo regret (over the 10 repetitions) as a function of time and the average pseudo regret + one standard deviation (over the 10 repetitions). Remember that in the i.i.d. setting the pseudo regret is defined as $\bar{R}_t = \sum_{s=1}^{t} \mathbb{E}[N_t(a)] \Delta(a)$, so each time an algorithm plays a suboptimal action a it accumulates $\Delta(a)$ regret.

Important: do not forget to add a legend and labels to the axes.

Break UCB1 Now design an adversarial sequence for which you expect UCB1 to suffer linear regret. Consider a version of UCB1 that breaks ties randomly. Explain how you design the sequence and execute UCB1 and EXP3 on it and report the observations (in a form of a plot). Even though the sequence is deterministic, you should make several repetitions (say, 10), because there is internal randomness in the algorithms. (If you fail to break UCB1 with randomized tie breaking, break UCB1 with deterministic tie breaking.)

You are welcome to try other settings (not for submission). For example, check what happens when μ^* is close to 1 or 0. Or what happens when the best arm is not static, but switches between rounds. Even though you can break UCB1, it is actually pretty robust, unless you design adversarial sequences that exploit the knowledge of the algorithm.

4 [Preview Question, not for submission. You will have it again in Home Assignment 7] Reinforcement learning



Let us consider the navigation task defined by depicted floorplan. The floorplan represents a 3×4 grid-world. Each of the 12 rooms corresponds to one state. The agent can go from one room to another if the rooms are connected by a door. The actions are the movements $\{\text{up}, \text{down}, \text{right}, \text{left}\}$. Every

movement (except in the terminal state) gives a negative reward of -1 (i.e., every movements costs 1). If you bump into a wall without a door, you stay in the same room, but the movement has still to be paid. There are three rooms that give you an *additional* negative reward of -5 and -10, respectively, *when you enter them*, see figure. The room in the bottom right is a terminal state. An episode ends when this room is reached, which can be modelled by every action in this state leaving the state unchanged and having no cost.

This exercise requires an implementation of the MDP. Note that all rewards and transitions are deterministic. They could be described by simple mappings $S \times A \to \mathbb{R}$ and $S \times A \to S$, respectively, which can be encoded by simple tables.

1. Use an algorithm presented in the lecture to compute the value function V^{rand} of the random policy, that is, the policy that chooses an action uniformly at random in every state. Report the 12 V^{rand} values. Provide the implementation of the algorithm.

2. Use an algorithm presented in the lecture to compute the optimal value function V^*. Report the 12 V^* values. Provide the implementation of the algorithm.

5 [Optional, not for submission] Empirical evaluation of algorithms for adversarial environments

Is it possible to evaluate experimentally the quality of algorithms for adversarial environments? If yes, how would you design such an experiment? If no, explain why it is not possible.

Hint: Think what kind of experiments can certify that an algorithm for an adversarial environment is good and what kind of experiments can certify that the algorithm is bad? How easy or hard is it to construct the corresponding experiments?

Good luck!
Yevgeny, Christian, Tobias & Julian