ML Exam Questions 28/8/2020

Question 1 (Linear Regression)

Tell if the following statements about Linear Regression are true or false. Motivate your answers.

In linear regression models, the target is linear with respect to model parameters and to the problem variables.
FALSE, linear regression models require the target to be linear only with respect to the model parameters.

In linear regression it is possible to extend the model with additional terms that are polynomial in the model parameters.
FALSE, linear regression models require the target to be linear with respect to the model parameters.

When using Gaussian basis functions, we introduce additional model parameters (i.e., the mean (\mu) and standard deviation (\sigma) of the gaussian) that must be trained from data.
FALSE, \mu and \sigma are not model parameters to train but constants.

The Gaussian basis functions differ from polynomial and sigmoidal ones because their contributions include the whole input domain instead of being limited to a range.
FALSE, it is the opposite: gaussian function is significantly greater than zero only in a limited subset of input domain.

In a regression problem with multiple outputs, least squares cannot be applied because the computational cost increases exponentially with the number of outputs.
FALSE, the most expensive computation (i.e., computing the Hat Matrix) is performed only once and can be used for the other outputs.

Ordinary least squares solution can become computationally infeasible when the number of features grows significantly.
TRUE, because the computational cost of Ordinary Least Squares is cubic with respect to the number of features.

Lasso and Ridge regression both allow closed-form optimization but they involve different constraints on the weights vector.
FALSE, Lasso does not allow closed form optimization.

As long as we assume the targets are generated by a linear model with a zero-mean Gaussian noise, the maximum likelihood approach provides a better solution than the one found solving ordinary least squares.
FALSE, computing the likelihood function and maximizing it with respect to the weights of the model, we get exactly the same solution computed with OLS.

In order to apply the maximum likelihood approach to solve a linear regression problem, it is not necessary to make any assumption on the prior distribution of the weights of the model.
TRUE, maximum likelihood approach does not require to model completely the posterior distribution and, hence, does not require to assume any prior distribution.

When an uninformative prior (infinitely broad) is used in the Bayesian Linear Regression framework, the maximum a posteriori is equivalent to the maximum likelihood solution.
TRUE, because assuming an infinitely broad prior, the posterior probability becomes equivalent to the likelihood (except for the marginal likelihood that acts as normalizing constant).

The Bayesian Linear Regression framework can be applied in a sequential learning setting.
TRUE, by simply using as a prior the posterior computed with past data, it is possible to use the Bayesian framework in a sequential learning setting.

The Bayesian Linear Regression framework allows us to use regularization.
TRUE, indeed the Bayesian Linear Regression computes a solution equivalent to ridge regression where the regularization coefficient depends from the variance of data and from the variance of the prior (the smaller the variance of the prior, the stronger the regularization).

Question 2 (RL)

Tell whether the following statements about MDP and DP are true or false. Motivate your answers.

After the policy improvement step, the policy always changes.
FALSE, if the algorithm finds the optimal policy, then it remains the same after the policy improvement step.

Some sequential decision problems cannot be modeled as an MDPs.
TRUE, in order to model a problem as an MDP, we have to assume that the environment state is fully observable and Markovian, hence the current state of the environment should be completely determined by the current observation made by the agent. In some cases, we can "transform" the state of the environment to make it Markovian.

It is always better to perform Policy Evaluation using Dynamic Programming instead of solving a linear system.

FALSE, when the state space is small we can use the closed form solution. However, even if the linear system approach offers the exact solution, for very large problems it can be impractical to solve it, hence, we can resort to using the approximated solution offered by DP.

There are some tasks in which we may not need to discount rewards.
TRUE, for example in an episodic task, we may set the discount factor equal to 1.

The best approach to solve a large MDP, exploiting the knowledge of the one-step dynamics, is Dynamic Programming.
TRUE, an MDP can be solved with a brute force approach, using Dynamic Programming (DP) approaches (as Policy Iteration or Value iteration), or with Linear Programming (LP). Since the number of states is often very large (e.g., often growing exponentially with the number of state variables) it is impractical to use brute force (which scales exponentially in the number of states) or LP, which typically does not scale well on large problems. On the other hand, DP approaches have a polynomial dependence on the number of states and actions, hence, they can scale better on larger instances.

The only optimal policy of an MDP can be non-Markovian.
FALSE, we are guaranteed that a stationary deterministic Markovian optimal policy exists, but it might also exist a non-Markovian optimal policy.

We can solve Bellman Optimality Equation with a linear system.
FALSE, since the equation involves the computation of a maximum, which is not linear.

With the optimal value function, I can compute the optimal policy, even without knowing one-step dynamics.
FALSE, the optimal policy is the greedy policy w.r.t. the optimal Q-function. If I have the optimal value function, but not the one-step dynamics, I cannot recover the optimal Q-function. Moreover, since I do not know the model, I cannot use DP techniques, but I have to use RL algorithms.

In Policy Evaluation, we can converge to the correct value function, only if we properly initialize its values.
FALSE, it can be proved that the convergence is assured for any initialization of the value function (the Bellman Expectation Operator is a contraction).

In an MDP, the past actions you perform might influence the future rewards you will gain.
FALSE, past actions influence past rewards and the dynamics of the process that brought you to a certain state. However, rewards depend only on the current (Markovian) state and action, not on past actions.

Given a value function, there is only one policy that corresponds to it.
FALSE, a policy is mapped on a single value function, but the opposite is not true: different policies can obtain the same values in each state, but in different ways.

Value Iteration may not converge to the optimal value function.
FALSE, Value Iteration always converges to the optimal in the limit on infinite iterations.

Question 3 (Learning Theory + No Free Lunch Theorem)

Tell if the following statements about Computational Learning Theory are true or false. Motivate your answers.

When dealing with a classification problem, thanks to the No Free Lunch Theorem we know that the choice of a learning algorithm does not affect the final performance.
FALSE, the NFL states that learning algorithms are equivalent when averaging the performance over all the possible learning problems. Instead, for a specific problem a certain algorithm might be better suited than another.

For every learning problem on which our preferred algorithm performs well, we can always design a learning problem on which it performs equally bad.
TRUE, as a corollary to the NFL we have that, for any problem on which a learner has accuracy 0.5 + delta, there exists a problem on which the learner has accuracy 0.5 - delta

Given a hypothesis space H, to prove VC(H) = 3 we just need to show that there exists a set of three points that can be shattered by some hypothesis in H.
FALSE, we also need to show that it does not exist a set of four points that can be shattered by some hypothesis in H.

If a hypothesis space is finite, its VC dimension is finite.
TRUE, if a hypothesis space has finite cardinality |H|, its VC dimension is bounded from above: VC(H) <= log_2 (|H|).

If a hypothesis space has an infinite VC dimension, we cannot provide a finite bound over its sample complexity (N).
TRUE, if a hypothesis space has an infinite cardinality we might still have a finite sample complexity bound related to its VC dimension. But if the VC dimension is also infinite we cannot bound the value of N.

The higher the VC dimension of a hypothesis space H, the larger the set of concepts we can learn within H.
TRUE, the VC dimension is a measure of the expressive power of the hypothesis space.

The PAC-Learning bound on the true error for consistent hypotheses has a linear dependence on the cardinality of the hypothesis space.
TRUE, since Pr(L_true(h) >= epsilon) <= |H| exp(- epsilon * N) for any consistent hypothesis h in H.

The PAC bound for agnostic learning provides an alternative view of the bias variance dilemma.

TRUE, the PAC bound shows that a larger hypothesis space has a looser bound (higher variance) than a smaller one, but with the former we will likely have a lower training error (lower bias) than the latter.

Despite the so-called curse of dimensionality, the sample complexity (N) PAC bound grows linearly with the number of features (M).
FALSE, the PAC bound on N grows exponentially with the number of features M.

The PAC-Learning bounds with the VC dimension of an hypothesis space H are strictly tighter than the bounds with the cardinality of H.
FALSE, the bounds with the VC dimension can be either tighter or looser. In general, the bounds with the VC dimension are convenient when we consider infinitely large hypothesis spaces having finite VC dimensions.

Question 4 (Categorize ML problems)

Categorize the following machine learning problems. For each one of them suggest a technique to solve it and some features or actions/states pairs which might provide useful information for the specified technique.

Predict the outcome of a football match.
If we assume the outcome is binary (Win/Loss) it is a classification problem, and we can resort to SVM to solve it. If the outcome is the number of goals scored (for each team) it is a regression problem and we might use Kernel Regression to solve it. In both cases the features might be: home/away status, previous results between the teams, and the statistics of the players.

Select the students which are worthy of a scholarship.
We want to determine which student will get or not the scholarship, therefore it is a classification problem. We can solve it with logistic regression using the GPA as input. If the number of scholarships is limited, the problem becomes a ranking problem, which requires more sophisticated techniques to be solved.

Optimize the process of watering plants automatically.
Here we want to understand what is the best policy for watering plants, therefore it is a control problem. If the system is simple and fully observable we can rely on DP control techniques for MDP, otherwise we can use RL (for instance Q-learning). The action is the time and quantity of water to feed the plants and the states is the health and current humidity for each plant.

Evaluate the performance of a cleaning robot swarm in a specific building.
We are in an RL evaluation problem. We want to understand how good is a specific policy, which can be performed using either MC or TD estimation techniques. Here the actions are the movements of the robots, the cleaning/waiting actions and the states are the energetic state of each robot and their position in the building.

Automatically determine inappropriate photos to remove from Instagram.
Here we want to classify those photos that are inappropriate, thus a binary classification problem. We can use SVM or CNN and use the photo pixels as input features.

Identify groups of similar movies (in terms of content and genre) from a large database.
We want to generate some grouping of samples starting from an unlabeled dataset. This is a clustering problem. The technique one might use if K-means and the features are the movie length, the cast, the director, and information about the release.

Determine how to invest in the stock market.
If we want to estimate the value of a stock we are in a regression setting (linear regression, using correlated index or stocks as features). If we want to decide how to perform investments, we are in a RL control problem (SARSA with buy/sell as action and the portfolio situation as state).

Learn how to play board games.
This is an RL control problem. The method used might be SARSA or Q-learning. The actions are the actions each player can execute (to play a card, a meeple, or a tile) and the states are the pieces visible to the player (either on the board or at disposal of other players).

Identify the most relevant elements in an email that determine its importance.
This is a feature selection problem, in which we want to select only those words that are the most important for a regression task (degree of importance). In this case the most appropriate technique consists of a wrapper method, which uses directly the importance in the selection of the features. Here the word counting in each email constitutes the input of the problem.

Deciding the most promising treatment for a specific disease.
If we assume it is a clinical trial, it is a MAB problem, solvable with e.g., UCB1 using the different treatment as arms. Instead, if we want to classify patients and give different drugs to each of them, the problem is classification and we can use SVM using the patient clinical data as features.

Prediction of the number of new COVID19 cases for the next week in Italy.
Since the number of new cases has a topology, this is a regression problem. We can use linear regression to solve the problem. A set of possible features are the data about the virus diffusion of the previous day. Notice that if we think that the phenomenon is changing over time we are not allowed to use supervised learning techniques anymore, but we should resort to time series estimation techniques.

Select the best set of tyres for a specific track in a MotoGP race.
This consists in the selection of a kind of tyre among a set, therefore is a classification problem. We can use the perceptron to solve the problem using some features from the track (top speed, length, usual track temperature) and some about the driver and the motorcycle.

Determine the critical factors for the development of a contagion.
We are facing a problem to understand the features that are the most relevant for a phenomenon, which is a feature selection problem. We can use any wrapper or filtering techniques. Here the feature we want to test are the ones related to each potential patient and its geographical position.

Exercise 1 (MAB)

Consider the Thompson Sampling algorithm applied to an environment with Bernoulli rewards, in which the prior is a Beta distribution initialized as a uniform one: Beta(1,1). For each arm a_i in A = {a_0, . . ., a_4}, you are provided its posterior distributions Beta_i(alpha_t; beta_t), the true reward mean mu_i, and the last sample extracted from the Beta r(a_i):
<data here>

1. How much pseudo-regret did the TS algorithm accumulate so far?
2. What would UCB1 have chosen for the next round?
3. Suppose the next reward can be chosen by an adversary, but keeping the same domain: how could this adversary behave if it knows that the agent is using TS? What about UCB1 instead? (no computations are required)

Answers:
1. see different versions
2. see different versions
3. An adversary knows which kind of algorithm the agent is applying, hence, it can run the same algorithm in parallel to understand what the agent is willing to do next. It will always manage to obtain a loss equal to 1 with UCB1, since it is deterministic. However, with TS, the adversary cannot know in advance which samples would be extracted, but, it can still estimate the probability of each arm to be chosen, knowing the distributions. Therefore, it can choose to put a 1 on the least probable action, in order to maximize the expected instantaneous regret the agent will incur.

==========
VERSION A

a_0: alpha_t = 8, beta_t = 6, r_t = 0.46, mu_0 = 0.6
a_1: alpha_t = 8, beta_t = 18, r_t = 0.34, mu_1 = 0.3
a_2: alpha_t = 6, beta_t = 9, r_t = 0.33, mu_2 = 0.4
a_3: alpha_t = 18, beta_t = 19, r_t = 0.4, mu_3 = 0.5
a_4: alpha_t = 4, beta_t = 14, r_t = 0.2, mu_4 = 0.2

The total number of rounds is: T = 12 + 24 + 13 + 35 + 16 = 100

The pseudo regret is T * \mu^* - \sum \mu_{i_t}, where \mu^* is the optimal expected reward and \mu_{i_t} is the expected reward of the arm chosen for turn t. Similarly, one might compute it as:

T * \mu^* - \sum_i (alpha_t+beta_t-2) * \mu_i

The pseudo-regret is: 60.0 - (7.2 + 7.2 + 5.2 + 17.5 + 3.2) = 60.0 - 40.3 = 19.7

The upper bounds computed with UCB1 are:

a_0: U_t = 7 / 12 + sqrt(2 * log(100) / 12) = 1.459
a_1: U_t = 7 / 24 + sqrt(2 * log(100) / 24) = 0.911
a_2: U_t = 5 / 13 + sqrt(2 * log(100) / 13) = 1.226
a_3: U_t = 17 / 35 + sqrt(2 * log(100) / 35) = 0.999
a_4: U_t = 3 / 16 + sqrt(2 * log(100) / 16) = 0.946

Hence, the algorithm is going to choose arm 0 with U = 1.459
==========
VERSION B

a_0: alpha_t = 8, beta_t = 2, r_t = 0.68, mu_0 = 0.9
a_1: alpha_t = 5, beta_t = 11, r_t = 0.44, mu_1 = 0.3
a_2: alpha_t = 10, beta_t = 6, r_t = 0.69, mu_2 = 0.7
a_3: alpha_t = 20, beta_t = 6, r_t = 0.81, mu_3 = 0.8
a_4: alpha_t = 9, beta_t = 33, r_t = 0.2, mu_4 = 0.2

The total number of rounds is: T = 8 + 14 + 14 + 24 + 40 = 100

The pseudo regret is T * \mu^* - \sum_t \mu_{i_t}, where \mu^* is the optimal expected reward and \mu_{i_t} is the expected reward of the arm chosen for turn t. Similarly, one might compute it as:
T * \mu^* - \sum_i (alpha_t+beta_t-2) * \mu_i

The expected pseudo-regret is: 90.0 - (7.2 + 4.2 + 9.8 + 19.2 + 8) = 90.0 - 48.4 = 41.6

The upper bounds computed with UCB1 are:

a_0: U_t = 7 / 8 + sqrt(2 * log(100) / 8) = 1.948
a_1: U_t = 4 / 14 + sqrt(2 * log(100) / 14) = 1.097
a_2: U_t = 9 / 14 + sqrt(2 * log(100) / 14) = 1.454
a_3: U_t = 19 / 24 + sqrt(2 * log(100) / 24) = 1.411
a_4: U_t = 8 / 40 + sqrt(2 * log(100) / 40) = 0.68

Hence, the algorithm is going to choose arm 0 with U = 1.948
==========
VERSION C

a_0: alpha_t = 3, beta_t = 4, r_t = 0.8, mu_0 = 0.4

a_1: alpha_t = 5, beta_t = 5, r_t = 0.25, mu_1 = 0.5
a_2: alpha_t = 18, beta_t = 9, r_t = 0.62, mu_2 = 0.7
a_3: alpha_t = 10, beta_t = 25, r_t = 0.3, mu_3 = 0.3
a_4: alpha_t = 27, beta_t = 4, r_t = 0.88, mu_4 = 0.9

The total number of rounds is: T = 5 + 8 + 25 + 33 + 29 = 100

The pseudo regret is T * \mu^* - \sum \mu_{i_t}, where \mu^* is the optimal expected reward and \mu_{i_t} is the expected reward of the arm chosen for turn t. Similarly, one might compute it as:
T * \mu^* - \sum_i (alpha_t+beta_t-2) * \mu_i

The pseudo-regret is: 90.0 - (2.0 + 4.0 + 17.5 + 9.9 + 26.1) = 90.0 - 59.5 = 30.5

The upper bounds computed with UCB1 are:

a_0: U_t = 2 / 5 + sqrt(2 * log(100) / 5) = 1.757
a_1: U_t = 4 / 8 + sqrt(2 * log(100) / 8) = 1.573
a_2: U_t = 17 / 25 + sqrt(2 * log(100) / 25) = 1.287
a_3: U_t = 9 / 33 + sqrt(2 * log(100) / 33) = 0.801
a_4: U_t = 26 / 29 + sqrt(2 * log(100) / 29) = 1.46

Hence, the algorithm is going to choose arm 0 with U = 1.757
==========
VERSION D

a_0: alpha_t = 5, beta_t = 11, r_t = 0.37, mu_0 = 0.3
a_1: alpha_t = 7, beta_t = 8, r_t = 0.47, mu_1 = 0.5
a_2: alpha_t = 17, beta_t = 3, r_t = 0.5, mu_2 = 0.9
a_3: alpha_t = 29, beta_t = 9, r_t = 0.74, mu_3 = 0.8
a_4: alpha_t = 4, beta_t = 17, r_t = 0.36, mu_4 = 0.2

The total number of rounds is: T = 14 + 13 + 18 + 36 + 19 = 100

The pseudo regret is T * \mu^* - \sum \mu_{i_t}, where \mu^* is the optimal expected reward and \mu_{i_t} is the expected reward of the arm chosen for turn t. Similarly, one might compute it as:
T * \mu^* - \sum_i (alpha_t+beta_t-2) * \mu_i

The pseudo-regret is: 90.0 - (4.2 + 6.5 + 16.2 + 28.8 + 3.8) = 90.0 - 59.5 = 30.5

The upper bounds computed with UCB1 are:

a_0: U_t = 4 / 14 + sqrt(2 * log(100) / 14) = 1.097

a_1: U_t = 6 / 13 + sqrt(2 * log(100) / 13) = 1.303
a_2: U_t = 16 / 18 + sqrt(2 * log(100) / 18) = 1.604
a_3: U_t = 28 / 36 + sqrt(2 * log(100) / 36) = 1.284
a_4: U_t = 3 / 19 + sqrt(2 * log(100) / 19) = 0.854

Hence, the algorithm is going to choose arm 2 with U = 1.604

Exercise 2 (Perceptron)

VERSION A
Consider a binary linear classifier defined by parameters w = [1, 2]. Answer the following questions related to the perceptron algorithm. Provide full calculations and clear motivations.

1) Predict the class of the data point x_1 = [0, 1] with the given classifier (+1 represents the positive class, -1 the negative class).

y_1 = w' * x_1 = [1, 2]' * [0, 1] = 2 > 0 → +1 positive class.

2) Consider the data point (x_2, t_2) = ([0, -1], -1). Update the classifier with the perceptron algorithm (alpha = 1).

y_3 = w' * x_3 = [1, 2]' * [0, -1] = -2 < 0 → -1 negative class.

The data point is correctly classified: we don't need to update the classifier.

3) Consider the data point (x_3, t_3) = ([3, 2], -1). Update the classifier with the perceptron algorithm (alpha = 1).

y_3 = w' * x_3 = [1, 2]' * [3, 2] = 7 > 0 → +1 positive class.

The data point is misclassified, thus we need to update the classifier:
w_new = w + x_3 * t_3 = [1, 2] + [-3, -2] = [-2, 0].

4) What can we say about the loss function of the perceptron after a single update to the classifier?
When we update the classifier for a given data point x, the classification error associated with x is guaranteed to decrease (- w_new * x < - w * x). However, we cannot say anything about the loss function computed on the whole dataset.

VERSION B
Consider a binary linear classifier defined by parameters w = [-2, 1]. Answer the following questions related to the perceptron algorithm. Provide full calculations and clear motivations.

1) Predict the class of the data point x_1 = [-1, 0] with the given classifier (+1 represents the positive class, -1 the negative class).

y_1 = w' * x_1 = [-2, 1]' * [-1, 0] = 2 > 0 → +1 positive class.

2) Consider the data point (x_2, t_2) = ([1, 0], -1). Update the classifier with the perceptron algorithm (alpha = 1).

y_3 = w' * x_3 = [-2, 1]' * [1, 0] = -2 < 0 → -1 negative class.

The data point is correctly classified: we don't need to update the classifier.

3) Consider the data point (x_3, t_3) = ([-2, 3], -1). Update the classifier with the perceptron algorithm (alpha = 1).

y_3 = w' * x_3 = [-2, 1]' * [-2, 3] = 7 > 0 → +1 positive class.

The data point is misclassified, thus we need to update the classifier:
w_new = w + x_3 * t_3 = [-2, 1] + [2, -3] = [0, -2].

4) What can we say about the loss function of the perceptron after a single update to the classifier?
When we update the classifier for a given data point x, the classification error associated with x is guaranteed to decrease (- w_new * x < - w * x). However, we cannot say anything about the loss function computed on the whole dataset.

VERSION C

Consider a binary linear classifier defined by parameters w = [-1, -2]. Answer the following questions related to the perceptron algorithm. Provide full calculations and clear motivations.

1) Predict the class of the data point x_1 = [0, -1] with the given classifier (+1 represents the positive class, -1 the negative class).

y_1 = w' * x_1 = [-1, -2]' * [0, -1] = 2 > 0 → +1 positive class.

2) Consider the data point (x_2, t_2) = ([0, 1], -1). Update the classifier with the perceptron algorithm (alpha = 1).

y_3 = w' * x_3 = [1, -2]' * [0, 1] = -2 < 0 → -1 negative class.

The data point is correctly classified: we don't need to update the classifier.

3) Consider the data point (x_3, t_3) = ([-3, -2], -1). Update the classifier with the perceptron algorithm (alpha = 1).

y_3 = w' * x_3 = [-1, -2]' * [-3, -2] = 7 > 0 → +1 positive class.

The data point is misclassified, thus we need to update the classifier:
w_new = w + x_3 * t_3 = [-1, -2] + [3, 2] = [2, 0].

4) What can we say about the loss function of the perceptron after a single update to the classifier?
When we update the classifier for a given data point x, the classification error associated with x is guaranteed to decrease (- w_new * x < - w * x). However, we cannot say anything about the loss function computed on the whole dataset.

VERSION D

Consider a binary linear classifier defined by parameters w = [2, -1]. Answer the following questions related to the perceptron algorithm. Provide full calculations and clear motivations.

1) Predict the class of the data point x_1 = [1, 0] with the given classifier (+1 represents the positive class, -1 the negative class).

y_1 = w' * x_1 = [2, -1]' * [1, 0] = 2 > 0 → +1 positive class.

2) Consider the data point (x_2, t_2) = ([-1, 0], -1). Update the classifier with the perceptron algorithm (alpha = 1).

y_3 = w' * x_3 = [2, -1]' * [-1, 0] = -2 < 0 → -1 negative class.

The data point is correctly classified: we don't need to update the classifier.

3) Consider the data point (x_3, t_3) = ([2, -3], -1). Update the classifier with the perceptron algorithm (alpha = 1).

y_3 = w' * x_3 = [2, -1]' * [2, -3] = 7 > 0 → +1 positive class.

The data point is misclassified, thus we need to update the classifier:
w_new = w + x_3 * t_3 = [2, -1] + [-2, 3] = [0, 2].

4) What can we say about the loss function of the perceptron after a single update to the classifier?

When we update the classifier for a given data point x, the classification error associated with x is guaranteed to decrease (- w_new * x < - w * x). However, we cannot say anything about the loss function computed on the whole dataset.