DATA12001 Advanced Course in Machine Learning

Exercise set 1 **Due April 9, 23:59**

In this exercise set we focus on fundamental concepts such as statistics, optimization, and linear algebra (lecture slides 2–4).

Submit pen&paper (as JPG or PDF) and application exercises (as PDF) to Moodle. Submit TMC exercises to TMC.

---

1 Risk

pen & paper + application, 5p

Let us consider one-dimensional regression problem with model $\hat{y} = \alpha x$, where α is the parameter of the model. We assume the data-generating distribution $p(y, x)$ is known, and corresponds to the process that first samples x uniformly between -3 and 3 and then y conditional on x uniformly between $2x - 0.5$ and $2x + 0.5$. We want to evaluate the risk for the loss

$$\mathcal{L}(y, x, \alpha) = (y - \hat{y})^2 = (y - \alpha x)^2$$

for any given α.

(a) Write down the definition of risk $R(\alpha)$ and compute it analytically with pen and paper. Which α gives the smallest risk?

(b) Write a small piece of code for estimating the risk with Monte Carlo, using

$$\hat{R}(\alpha) \approx \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}(y_m, x_m, \alpha),$$

where $\{x_m, y_m\} \sim p(y, x)$ are samples from the data distribution.

(c) Plot numerical approximation for $R(1.5)$ as a function of M and compare it against the true value $R(1.5)$. Include the figure and the code, and submit your answer to moodle.

2 Multivariate calculus in practice

pen & paper, 3p

Assume we have N data points $\mathbf{x}_n$ represented by D-dimensional real-valued column vectors and corresponding scalar outputs y_n. We predict y_n using the model

$$\hat{y}_n = e^{\mathbf{x}_n^T \boldsymbol{\theta} + b}, \tag{1}$$

where $\boldsymbol{\theta}$ is also a D-dimensional column vector and b is a scalar. Assume we want to minimize (w.r.t. to $\boldsymbol{\theta}$ and b) the weighted sum of squared errors

$$\mathcal{L} = \sum_{n=1}^{N} w_n (y_n - \hat{y}_n)^2$$

for some weights $w_n > 0$ provided for the individual data points.

(a) Write the loss for whole data collection in matrix notation. Introduce suitable matrices and vectors and provide their dimensions, checking that the expression is valid. Note that the exponentiation in (1) is done for a scalar – you can use notation $e^{\mathbf{A}}$ to denote element-wise exponentiation of a matrix.

(b) Write down the derivative w.r.t. $\boldsymbol{\theta}$ and b using the chain rule of differentiation.

(c) Provide the actual gradient by figuring out each term in the chain rule.

3 Stochastic gradient descent

TMC, 5p

Complete the implementation of stochastic gradient for least squares (TMC exercise: `part1/01.gradient`).

Complete the following functions:

(a) `step_size(self)`, determines the step size of the gradient. Two modes should be supported

- Deterministic schedule (`self.mode == 'determ'`): during the tth iteration, the step size is a *scalar*, equal to $\alpha = \frac{\eta}{1+\eta\tau t}$ with two parameters η and τ controlling the initial step size and the decay speed, respectively. Use `self.eta`, `self.tau`, and `self.round` for η, τ, and t.
- AdaGrad for adaptive selection (`self.mode == 'adagrad'`): during the tth iteration, the step size is a *vector* α, defined element-wise as

$$\alpha_d = \frac{\eta}{\tau + \sqrt{\sum_{i=1}^{t-1} g_{id}^2}},$$

where g_i are the gradients from the previous rounds. Use `self.eta`, `self.tau` for η and τ. Use `self.grads_squared` for the sum $\sum_{i=1}^{t-1} g_i^2$ so that you do not have to store the gradients, or compute the sum.

(b) `update_step(self, grad)`, updates the parameters `self.round` and `self.grads_squared` used by `step_size`.

(c) `gradient(self, X, y, w)`, returns the gradient

$$\nabla_w \left\|\mathbf{X}\mathbf{w} - \mathbf{y}\right\|^2 / n,$$

where n is the number of data points (=first dimension of X).

Hints: Be aware of of matrix shapes. In Numpy, `*` is element-wise multiplication and `@` is the strandard matrix multiplication.

(d) `epoch(self, X, y, w, M)`, performs gradient descent over the whole data with mini-batches. The algorithm should split the data points into batches of size M (last batch may be smaller than M). Then the algorithm should apply the gradient descent on each batch using `gradient` and `step_size`, and `update_step`.

Hints: You should implement the functions in the described order. The tests will test the functions independently.

Submit your code to the TMC system.

4 Principal component analysis

TMC, 5p

Principal component analysis (PCA) is a linear method used for dimensionality reduction. Given a set of N data (row) vectors collected in a matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ the goal is to learn a set of L orthonormal basis vectors $\mathbf{W} \in \mathbb{R}^{D \times L}$ such that we get the best possible L-dimensional representation for the data vectors by projecting them onto that basis with $\mathbf{z}_n = \mathbf{x}_n \mathbf{W}$.

The quality of the representation is measured by reconstruction error. The reconstruction of a sample is obtained by projecting the low-dimensional vector $\mathbf{z}_n$ back to the observation space with $\widehat{\mathbf{x}}_n = \mathbf{W}\mathbf{z}_n$, and we use the squared error loss

$$\left\|\mathbf{X} - \widehat{\mathbf{X}}\right\|_F^2 = \left\|\mathbf{X} - \mathbf{Z}\mathbf{W}^T\right\|_F^2 = \left\|\mathbf{X} - \mathbf{X}\mathbf{W}\mathbf{W}^T\right\|_F^2 = \sum_{i=1}^{N}\sum_{j=1}^{D}(x_{ij} - \widehat{x}_{ij})^2.$$

We can show that $\mathbf{W}$ minimizing the error is formed by the first[1] L eigenvectors of the covariance matrix of $\mathbf{X}$.

Complete the implementation of PCA (TMC exercise: `part1/02.pca`).

Complete the following functions:

(a) `covariance_matrix(X, bias=False)`, computes the empirical covariance matrix of $\mathbf{X}$ of size $n \times k$. There are two widely-used versions of the empirical covariance matrices

$$\frac{1}{n}\sum_i (\mathbf{x}_i - \boldsymbol{\mu})^T(\mathbf{x}_i - \boldsymbol{\mu})$$

and

$$\frac{1}{n-1}\sum_i (\mathbf{x}_i - \boldsymbol{\mu})^T(\mathbf{x}_i - \boldsymbol{\mu}),$$

where $\boldsymbol{\mu}$ is average of $\mathbf{x}_i$. Note that $(\mathbf{x}_i - \boldsymbol{\mu})^T(\mathbf{x}_i - \boldsymbol{\mu})$ is not an inner product but a matrix of size $k \times k$. Implement both versions, if `bias == False`, then use $n-1$, otherwise use n as the normalization.[2]

(b) `pca(X)`, computes PCA for $L = 2$ components. Use `covariance_matrix` and `numpy.linalg.eig` function (already imported) to compute the eigenvectors. Remember to read the documentation of `eig` first. Note that you need to select 2 eigenvectors that have the largest eigenvalues, they are typically the first two vectors, but not always. You can use `np.argsort` to find the indices of the largest eigenvalues.

When doing PCA, it is customary to scale the dimensions, do *not* do it in this exercise.

Submit your code to the TMC system.

5 Political map and PCA

application, 2p

The dataset `elec2022.txt` contains the answers for Helsingin sanomat vaalikone by the candidates for 2022 county elections from western Uusimaa county. The first column in the dataset contains the party id, and the corresponding party names can be found in `parties.txt`. The remaining columns are the answers.

[1] By first we mean eigenvectors associated with the largest eigenvalues.

[2] For PCA, it will not matter which version is used since the eigenvectors will not change.

Apply PCA with 2 dimensions to the data (remove the party id column). You can use the code from the previous exercise, or you can use PCA provided by `sklearn` package.

Scatter plot the party *averages* of the principal components. Annotate the points with party names.

Include the figure and the code, and submit your answer to moodle.