Practical Session : Standard optimization problems

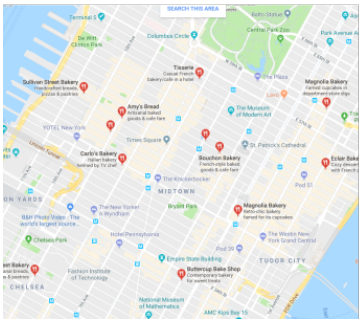
Rémi Flamary

1 Linear Program: Optimal Transport

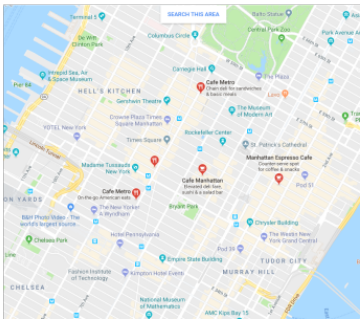
1.1 Dataset: Bakeries and cafés problem

We will solve the Bakery/Cafés problem of transporting croissants from a number of Bakeries to Cafés in a City (In this case Manhattan). We did a quick google map search in Manhattan for bakeries and Cafés and found the following list:

Bakeries



Cafés



We extracted from this search their positions and generated fictional production and sale number (that both sum to the same value). We provide in the file `manhattan.npz` to the position of Bakeries `bakery_pos` and their respective production `bakery_prod` which describe the source distribution. The Cafés where the croissants are sold are defined also by their position `cafe_pos` and `cafe_prod`. For fun we also provide an image map `Imap` that will illustrate the position of these shops in the city.

1. Load the file `manhattan.npz` in memory and recover all the vectors and matrices described above (`np.load`).
2. Plot the map `Imap` and the bakeries and cafés on the map as circles circle is proportional to their production (`pl.imshow,pl.scatter`).

1.2 Solving Optimal Transport

In this subsection we aim at solving the optimal transport problem between the bakeries and the cafés. The OT problem and how to express it as a standard problem are in the Linear Programming part of the course.

You will need to install the Python Optimal Transport toolbox POT. With anaconda with the following command:

```
conda install -c conda-forge pot
```

1. Compute the Euclidean distance matrix `C` between the bakeries and the cafés (`scipy.spatial.distance.cdist`).
2. Compute the matrices $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}, \mathbf{b}, \mathbf{G}, \mathbf{h}$ from exercise 4 in the second part of the course (`np.concatenate,np.ones`).

3. Solve the optimal transport optimization problem with the scipy linear program solver. Reshape the solution $\mathbf{x}$ as a matrix and check that the constraints are satisfied (`scipy.optimize.linprog`).
4. Change the solver method to simplex and compare the solution with the default interior point solver.
5. Solve the optimization problem using the POT toolbox and compare the solutions and computation times to the interior point and simplex solvers (`ot.emd,ot.tic,ot.toc`).

1.3 Interpreting the optimal transport solution

1. Plot a black line between the bakeries and cafés for which some croissants has to be transported (from the transport matrix).
2. Change the cost matrix to the 'cityblock' distance (also called manhattan distance). Is the solution the same?

2 Quadratic Program: Support Vector Machines

This part will require to use the python toolbox `cvxopt`. In order to install it execute the following command on anaconda:

```
conda install -c anaconda cvxopt
```

2.1 Dataset creation

1. Use the function `make_blobs` from `sklearn` to create a toy 2D example with `n=40` samples with parameters `n_samples=n, centers=2, random_state=6` (`sklearn.datasets.make_blobs`).
2. Convert binary classes to $\{-1, 1\}$ and plot the samples in 2D (`pl.scatter`).

2.2 Solving the SVM dual QP

We recall that the optimization problem for SVM can be expressed as the following.

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}_n^T\boldsymbol{\alpha} \tag{1}$$

$$\text{s.t.} \quad \mathbf{y}^T\boldsymbol{\alpha} = 0 \tag{2}$$

$$\mathbf{0}_n \leq \boldsymbol{\alpha} \leq C\mathbf{1}_n \tag{3}$$

where $G_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)y_iy_j$ with k the kernel and $\{\mathbf{x}_i, y_i\}_i$ the training samples. When the optimal solution $\boldsymbol{\alpha}$ is found the final binary classifier can be expressed as

$$f(\mathbf{x}) = \sum_i \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i)$$

All the samples $\mathbf{x}_i$ such that $\alpha_i > 0$ are called support vectors.

1. Compute the linear kernel matrix $\mathbf{K}$ such that $K_{i,j} = \mathbf{x}_i^T\mathbf{x}_j$.
2. Express the matrix $\mathbf{Q}, \mathbf{c}, \mathbf{A}, \mathbf{b}, \mathbf{G}, \mathbf{h}$ for the standard QP problem in the course corresponding to the SVM problem above. Compute them in Python for $C = 1$.
3. Solve the QP with the `qp` solver from `cvxopt`. Note that all the matrices and vectors have to be converted to the `matrix` type from cvxopt before calling the solver (`cvxopt.solvers.qp,cvxopt.matrix`).
4. Convert the solution of the problem back to `numpy` 1D array and plot the selected support vector (such that $\alpha_i > 0$) with the dataset.

2.3 Comparison with `sklearn`

1. Estimate an SVC classifier from `sklearn` with a linear kernel and $C = 1$ (`sklearn.svm.SVC`).
2. Reconstruct the original $\boldsymbol{\alpha}$ from classifier `clf` where `clf.support_` contains the index of the support vectors and `clf.dual_coef_` contains the corresponding values $\alpha_i y_i$.
3. Compare the two solutions.
4. Solve the SVM with your formulation and sklearn, for a Gaussian kernel and check that the solution is the same.

3 Solving the SVM primal QP

For a linear kernel one can solve the SVM in the primal. In this case the optimization problem is

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}, \mathbf{z}\in\mathbb{R}^n} \quad & C\sum_i z_i + \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\mathbf{x}_i^T\mathbf{w} + b) \geq 1 - z_i,\ \forall i \\ & \mathbf{z} \geq \mathbf{0} \end{aligned} \tag{4}$$

1. Express the corresponding standard QP matrices with $\mathbf{x} = [\mathbf{w}^T, b, \mathbf{z}^T]^T$.
2. Solve the QP and compare your estimated $\mathbf{w}$ to the one estimated using the dual solution with $\mathbf{w}^\star \sum_i y_i \mathbf{x}_i \alpha_i^\star$.