The Ticket Restaurant Assignment Problem

Gabriele Cerizza

Università degli Studi di Milano
gabriele.cerizza@studenti.unimi.it
https://github.com/gabrielecerizza/orc_project

Introduction

In this report we detail the results of our experiments on the ticket restaurant assignment problem, pursuant to the project specifications set out for the Operational Research Complements course of the Università degli Studi di Milano[1].

In Section **??** we illustrate the dataset used in the experiments. In Section 4 we briefly describe the algorithm and its implementation. In Section 5 we show the results of our experiments and provide comments on them. Finally, Section 6 contains our concluding remarks.

1 Ticket Restaurant Assignment Problem

1.1 Definition

The ticket restaurant assignment problem (TRAP) is defined as follows. A ticket company (TC) possesses two kinds of restaurant tickets: the low-profit tickets and the high-profit tickets. TC gives a certain amount of tickets to customer companies (CC). Each CC receives only one kind of tickets. Each CC has different sets of employees and each set uses the tickets to buy meals in a specific restaurant.

For each restaurant with which TC has a deal, a given ratio between low-profit and high-profit tickets must be observed. TC must maximize the profit while complying with this constraint. Thus, maximizing the profit amounts to minimizing the number of low-profit tickets while ensuring that a given amount of low-profit tickets is assigned for each considered restaurant.

1.2 Formalization

Let $I = \{1, \dots, m\}$ be a set of restaurants, $J = \{1, \dots, n\}$ be a set of customer companies, $b \in \mathbb{Z}_+^m$ be the vector representing the minimum amount of low-profit tickets to be assigned for each restaurant, and $A \in \mathbb{Z}_+^{m \times n}$ be the matrix representing the amount of low-profit tickets assigned to each given customer

[1] https://homes.di.unimi.it/righini/Didattica/ComplementiRicercaOperativa/ComplementiRicercaOperativa.htm

company for each given restaurant, so that a_{ij} is the amount of low-profit tickets assigned to customer company j for restaurant i. Then, the integer linear programming model of the ticket restaurant assignment problem is the following:

$$\begin{aligned} \min \quad & z = \sum_{j \in J} \left(\sum_{i \in I} a_{ij} \right) x_j && && (1) \\ \text{subject to} \quad & \sum_{j \in J} a_{ij} x_j \geq b_i && i = 1, \ldots, m\,, \\ & x_j \in \{0,1\} && j = 1, \ldots, n\,, \end{aligned}$$

where each binary variable x_j indicates whether customer company j is assigned low-profit tickets.

From (1), the following Lagrangean relaxation is obtained:

$$\begin{aligned} \min \quad & z_{\mathrm{LR}} = \sum_{j \in J} \left(\sum_{i \in I} (1 - \lambda_i) a_{ij} \right) x_j + \sum_{i \in I} \lambda_i b_i && && (2) \\ \text{subject to} \quad & x_j \in \{0,1\} && j = 1, \ldots, n\,, \\ & \lambda_i \geq 0 && i = 1, \ldots, m\,, \end{aligned}$$

where λ_i are the Lagrangean multipliers.

Finally, the dual of the linear programming relaxation of (1) is:

$$\begin{aligned} \max \quad & w = \sum_{i \in I} b_i y_i && && (3) \\ \text{subject to} \quad & \sum_{i \in I} a_{ij} y_i \leq c_j && j = 1, \ldots, n\,, \\ & y_i \geq 0 && i = 1, \ldots, m\,, \end{aligned}$$

where $c_j = \sum_{i \in I} a_{ij}$ for all $j \in J$.

1.3 Related Works

The problem at hand is equivalent to the set covering problem (SCP) when matrix A is binary and b is an all-ones vector. The SCP has been treated extensively in literature and many of the ideas developed within that context can be leveraged to solve the TRAP. For the SCP, both exact and heuristic algorithms have been devised (see the survey in [1]). We will focus only on exact algorithms for our problem.

A problem more closely related to the TRAP is the one called multicovering problem (MCP) [2], defined as follows:

$$\begin{aligned} \min \quad & z = c^\top x \\ \text{subject to} \quad & Ax \geq b\,, \\ & x_j \in \{0,1\} \qquad \forall j\,, \end{aligned}$$

where A is a binary matrix and b is a vector of positive integers. This problem differs from the one at hand in that A is binary and not simply non-negative.

Furthermore, a generalization of the TRAP may be identified in the so-called covering integer problem (CIP) [3], defined as follows:

$$\begin{aligned} \min \quad & z = c^\top x \\ \text{subject to} \quad & Ax \geq b\,, \\ & x_j \geq 0 \qquad \forall j\,, \end{aligned}$$

where all the entries in A, b and c are non-negative. This problem differs from the one at hand in that x is not binary.

2 Branch-and-bound

2.1 Primal Heuristics

2.2 Lower Bounds

2.3 Branching

2.4 Reduction

3 Computational Results

When searching for the best Lagrangean multipliers in (2), we cannot assume that the Lagrangean multipliers can be confined within the range $[0, 1]$.

Lemma 1. *The optimal Lagrangean multipliers may not lie in the range* $[0, 1]$.

Proof. Consider a matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 4 \\ 2 & 2 & 2 \end{bmatrix}$ and a vector $b = \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$.

Our analysis was carried out on the USPS dataset[2], comprising 9298 16×16 grayscale images. These images depict handwritten digits ranging from 0 to 9. A sample of these digits can be seen in Figure 1. The classes do not present severe imbalance issues, as shown in Figure **??**. The values of the examples are scaled in the interval $[0, 1]$.

To gauge the complexity of multiclass classification on this dataset, we performed dimensionality reduction by way of t-distributed Stochastic Neighbor Embedding (t-SNE) [5] and projected the examples onto a two-dimensional space. The result is given in Figure **??**. Within this figure the examples are clustered into ten clearly distinguishable groups, thus suggesting a low complexity classification task.

The USPS dataset is made available with predetermined training and test sets. In our experiments, however, we merged these sets and run a 5-fold cross-validation on the whole dataset, as illustrated in Section 5.

[2] https://www.kaggle.com/datasets/bistaumanga/usps-dataset

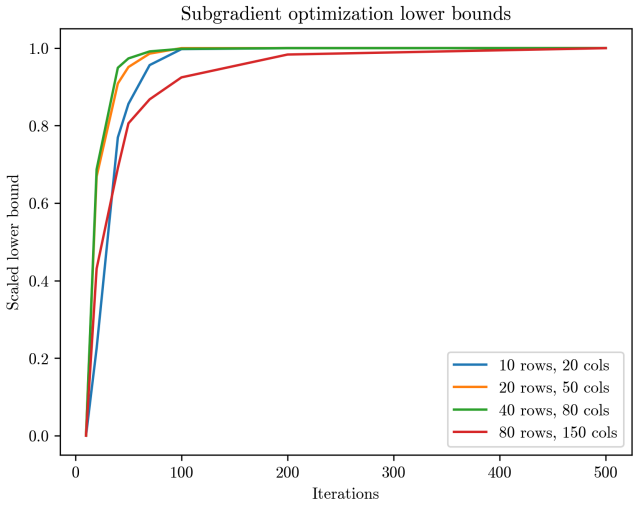

Fig. 1. Sample of images from the USPS dataset. The label is indicated below each image.

4 Pegasos Algorithm

In the present section we describe the Pegasos algorithm (Section 4.1), the kernel functions employed in our experiments (Section 4.2), and some implementation details to reduce the runtime of the algorithm (Section 4.3).

4.1 Description

The Pegasos algorithm [4] was introduced to solve the Support Vector Machine (SVM) [6] convex optimization problem for binary classification, which can be formalized as follows.

Let $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ be a training set with $\boldsymbol{x}_t \in \mathbb{R}^d$, $y_t \in \{-1, 1\}$ $\forall t = 1, \ldots, m$; and let $\phi_K : \mathbb{R}^d \to \mathcal{H}_K$ be a function such that $\dim(\mathcal{H}_K) \gg d$ and $\langle \cdot, \cdot \rangle_K$ is the inner product in $\mathcal{H}_K$. The optimization problem aims to find the maximum margin separating hyperplane in $\mathcal{H}_K$, which separates the examples according to their labels and maximizes the distance between the hyperplane and the closest example. This yields the following quadratic programming problem:

$$\begin{aligned} \min_{g \in \mathcal{H}_K, \boldsymbol{\xi} \in \mathbb{R}^m} \quad & \frac{\lambda}{2}\|g\|_K^2 + \frac{1}{m}\sum_{t=1}^{m} \xi_t \,, \\ \text{subject to} \quad & y_t \langle g, \phi_K(\boldsymbol{x}_t) \rangle_K \geq 1 - \xi_t \qquad t = 1, \ldots, m \,, \\ & \xi_t \geq 0 \qquad\qquad\qquad\quad t = 1, \ldots, m \,, \end{aligned}$$

where λ is a regularization coefficient dictating the relative importance of the two terms in the objective function; and ξ_t are slack variables that allow, but penalize, examples lying on the wrong side of the hyperplane.

This formulation can be turned into the following unconstrained optimization problem:

$$\min_{g \in \mathcal{H}_K} \quad \frac{\lambda}{2}\|g\|_K^2 + \frac{1}{m}\sum_{t=1}^{m} h_t(g)\,,$$

where $h_t(g) = [1 - y_t\langle g, \phi_K(\boldsymbol{x}_t)\rangle_K]_+$ is the hinge loss.

Pegasos solves this optimization problem using a stochastic gradient descent approach. The algorithm performs T iterations, each of which consists in drawing a training example uniformly at random and updating g through a gradient step. In particular, at iteration t Pegasos computes

$$g_{t+1} = \frac{1}{\lambda t}\sum_{r=1}^{t} \mathbb{I}\{h_{s_r}(g_r) > 0\} y_{s_r} K(\boldsymbol{x}_{s_r}, \cdot)\,, \tag{4}$$

where $\mathbb{I}$ is the indicator function, s_r is the index of the training example randomly drawn at iteration r, and $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel function such that $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi_K(\boldsymbol{x}), \phi_K(\boldsymbol{x}')\rangle_K$.

Note that g_{t+1} is a function, not a vector. Indeed, the dot in $K(\boldsymbol{x}_{s_r}, \cdot)$ stands for a missing argument, corresponding to the example on which the function is evaluated. Therefore, unlike vectors, g_{t+1} cannot be stored in memory and used to perform inference on new examples, since we have to evaluate g_{t+1} on them first.

In order to simplify the process, it was observed in [4] that (4) can be rewritten as

$$g_{t+1} = \frac{1}{\lambda t}\sum_{j=1}^{m} \boldsymbol{\alpha}_{t+1}[j] y_j K(\boldsymbol{x}_j, \cdot)\,,$$

where $\boldsymbol{\alpha}_{t+1} \in \mathbb{R}^m$ is a vector representing how many times each example $\boldsymbol{x}_j$ in the training set has been drawn and resulted in a non-zero loss, that is

$$\boldsymbol{\alpha}_{t+1}[j] = |\{r \leq t : s_r = j \wedge y_j\langle g_r, \phi_K(\boldsymbol{x}_j)\rangle_K < 1\}|\,.$$

The vector $\boldsymbol{\alpha}_{t+1}$ can be kept in memory and used to quickly evaluate g_{t+1} on new examples during both training and test. Algorithm 1 shows the pseudocode for the kernelized Pegasos algorithm using the abovementioned approach.

Once the vector $\boldsymbol{\alpha}_{T+1}$ has been obtained, the label of a new example $\boldsymbol{x}$ can be predicted by computing

$$\hat{y} = \operatorname{sgn}\left(\sum_{j=1}^{m} \boldsymbol{\alpha}_{T+1}[j] y_j K(\boldsymbol{x}_j, \boldsymbol{x})\right)\,. \tag{5}$$

It is not necessary to multiply the sum by $\frac{1}{\lambda T}$, since scaling does not affect the sign.

So far we illustrated the Pegasos algorithm for binary classification. In order to handle multiclass classification, we can adopt the one-vs-all multiclass strategy. We prepare a different binary classifier for each class. These binary

Algorithm 1: Pegasos algorithm in a kernel space

Hyperparameters: T, $\lambda > 0$
Data: Training set $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$
$\boldsymbol{\alpha}_1 \leftarrow (0, \ldots, 0)$
for $t = 1$ **to** T **do**
  Choose $s_r \in \{0, \ldots, |S|\}$ uniformly at random
  if $y_{s_r} \frac{1}{\lambda t} \sum_{j=1}^{m} \boldsymbol{\alpha}_t[j] y_j K(\boldsymbol{x}_j, \boldsymbol{x}_{s_r}) < 1$ **then**
    $\boldsymbol{\alpha}_{t+1}[s_r] = \boldsymbol{\alpha}_t[s_r] + 1$
  end
end
return $\boldsymbol{\alpha}_{T+1}$

classifiers are trained using labels encoded as 1 for the examples belonging to the class of the classifier, and encoded as -1 for the examples belonging to all the other classes.

Finally, to determine the class of a new example $\boldsymbol{x}$, we first compute

$$h_i = \sum_{j=1}^{m} \boldsymbol{\alpha}_{T+1}[j] y_j K(\boldsymbol{x}_j, \boldsymbol{x}) \tag{6}$$

for each classifier c_i, and then predict with

$$\hat{y} = \arg\max_i h_i \,.$$

Also in this case scaling h_i is unnecessary, given that $\frac{1}{\lambda T}$ is always positive and therefore does not change the order of the h_i values.

4.2 Kernel functions

In our experiments with the Pegasos algorithm, we employed two kernel functions: the Gaussian kernel and the polynomial kernel. The Gaussian kernel is defined as

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2\gamma} \|\boldsymbol{x} - \boldsymbol{x}'\|^2\right), \tag{7}$$

with $\gamma > 0$. The polynomial kernel is defined as

$$K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^\top \boldsymbol{x}')^n \,, \tag{8}$$

where $n \in \mathbb{N}$ is the degree of the polynomial.

4.3 Implementation

In this section we expound on two implementation details that improve the runtime of the algorithm.

Kernel matrix. In order to perform multiclass classification, we need to train multiple classifiers, each running the Pegasos algorithm over the same training set. Within this framework, the kernel function between two given examples may be computed multiple times, resulting in redundant computations.

Moreover, at each iteration the Pegasos algorithm computes the kernel function between a randomly drawn training example and all the other training examples. Drawing the same example multiple times results in further redundant computations.

Redundant computations adversely affect the runtime of the algorithm and should therefore be avoided. To this effect, we opted to compute ahead of time the kernel matrix $K \in \mathbb{R}^{m \times m}$, such that

$$K_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad \forall i, j = 1, \ldots, m\,.$$

This matrix was then dispatched to each classifier and used in place of the kernel function evaluation.

The drawback of the kernel matrix lies in its memory footprint. However, given the number of examples in the USPS dataset, such a matrix would require at most $\frac{9298 \times 9298 \times 32}{2^3 \times 10^9} \approx 0.34$ gigabytes of memory, which in modern machines constitutes a negligible amount.

Vectorization. The summations within the Pegasos algorithm can be vectorized, thus allowing to exploit the highly optimized array operations made available by math libraries.

In particular, the condition of the `if` statement within Algorithm 1 can be rewritten as

$$y_{s_r} \frac{1}{\lambda t} (\boldsymbol{\alpha}_t \odot y)^\top K_{s_r} < 1\,,$$

where $\odot$ is the Hadamard product and K is the kernel matrix between the training examples. Likewise, the formula for h_i in (6) can be rewritten as

$$\boldsymbol{h}_i = (\boldsymbol{\alpha}_{T+1} \odot y)^\top K'\,,$$

where K' is the kernel matrix between the test examples and the training examples and $\boldsymbol{h}_i$ is the vector containing the predictions for all the test examples returned by classifier c_i.

Also the computation of the kernel matrices can be vectorized. In the case of the polynomial kernel, the vectorization is trivial, since the dot product between vectors in (8) directly translates into a matrix product. In the case of the Gaussian kernel, however, an analogous approach would fail, because subtracting the training set matrix from itself in (7) would result in a null matrix. We therefore followed another approach.

Let $S \in \mathbb{R}^{n \times m}$ be a training set. In order to vectorize the kernel matrix with respect to the Gaussian kernel, we need to compute the difference between each row in S and all the rows in S. To accomplish this, we first turn S into a

tensor $T \in \mathbb{R}^{n \times 1 \times m}$, in which each row T_i is a $1 \times m$ matrix containing only the corresponding row S_i of the original matrix. Then, we simply compute

$$R = T - S\,,$$

where $R \in \mathbb{R}^{n \times n \times m}$ is a tensor having $R_i = S_i - S \; \forall i = 1, \ldots, n$, in which S_i is casted into a $\mathbb{R}^{n \times m}$ matrix using broadcasting rules[3]. Once R is obtained, the Gaussian kernel formula in 7 can be applied straightforwardly. The same procedure can be applied when computing the kernel matrix between the test and training examples.

One caveat of this approach is that R has a much larger memory footprint than the $n \times n$ kernel matrix. At most, the required memory for the USPS dataset would be $0.34 \times 256 \approx 87$ gigabytes. For this reason, it is necessary to split S into chunks, apply the procedure on each chunk and then concatenate the results.

Comparison. The training runtime reduction obtained by applying the above-mentioned optimizations on a single binary classifier is given in Figure **??**. The shown improvement becomes even more prominent when accounting the fact that, for multiclass classification, multiple classifiers need to be trained.

5 Experiments

In this section we show the results of our experiments on the multiclass classification task using the Pegasos algorithm. The experiments were run on a machine with 16 gigabytes of RAM and a CPU Intel(R) Core(TM) i7-9700K 3.60GHz with 8 cores.

Cross-validation. The algorithm performance was evaluated with a 5-fold cross-validation on the entire USPS dataset. The folds were stratified so as to retain the proportion of the classes found in the complete dataset.

Hyperparameters. We investigated the behavior of the algorithm for different choices of the hyperparameters T and λ, respectively representing the number of iterations and the regularization coefficient.

Concerning T, we gauged the performance of the algorithm over an increasing number of iterations, starting from a number much smaller than the cardinality of the training set and ending with a number several times bigger than the cardinality of the training set. More precisely, we chose the following values: 1000, 5000, 10000, 25000, 50000.

Concerning λ, in [4] the Pegasos algorithm was evaluated on the USPS dataset using λ equal to 1.36×10^{-4}. It was also observed therein how, given a fixed number of iterations, the performance of the algorithm was inversely

[3] https://numpy.org/doc/stable/user/basics.broadcasting.html

proportional to the value of λ, where λ ranged from 1e-7 to 1e-3. The authors remarked that this behavior of the Pegasos algorithm for low values of λ was known and attempts were being made to improve the performance for these values of the hyperparameter. We followed this insight, avoiding very low values of λ, and took the analysis one step further, experimenting with even higher values of λ, ranging from 1e-5 to 10.

We ran the algorithm with the polynomial and Gaussian kernel. The polynomial kernel was experimented upon with n equal to 2, 3, 4 and 7, in order to assess the effectiveness of the algorithm at both low and high degrees of the polynomial. The Gaussian kernel was run with γ equal to 0.25, as suggested in the project requirements, and also with γ equal to 0.75 and 2. We note that in [4] the authors used γ equal to 2 on the USPS dataset.

6 Conclusions

In this work we investigated the performance of the Pegasos algorithm described in Section 4 on a multiclass classification task over the USPS dataset described in Section **??**. The algorithm was evaluated with 5-fold stratified cross-validation across various settings of hyperparameters. The best performance (0.026 test error with zero-one loss) was achieved by using a polynomial kernel of degree 3, 50000 iterations and a regularization coefficient of 1. A similar performance (0.027 test error) was achieved by using a Gaussian kernel with γ equal to 2, 25000 iterations and a regularization coefficient of 1e-5.

The algorithm proved to be an effective classification tool, although the task was not particularly challenging. Moreover, the algorithm showed a noteworthy rate of convergence, bordering a test error of 0.070 after only 1000 iterations when using the Gaussian kernel with γ equal to 2.

Further research might explore a wider set of hyperparameters, especially in the direction of lower regularization coefficient values and higher values of γ when using a Gaussian kernel.

References

1. Caprara, A., Toth, P., and Fischetti, M. Algorithms for the Set Covering Problem. *Annals of Operations Research 98*, 1 (December 2000), 353–371.
2. Hall, N. G., and Hochbaum, D. S. The multicovering problem. *European Journal of Operational Research 62*, 3 (1992), 323–339.
3. Kolliopoulos, S. Approximating covering integer programs with multiplicity constraints. *Discrete Applied Mathematics 129* (08 2003), 461–473.
4. Shalev-Shwartz, S., Srebro, N., and Cotter, A. Pegasos: Primal estimated sub-gradient solver for svm. *Math. Program. 127* (03 2011), 3–30.
5. van der Maaten, L., and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research 9* (11 2008), 2579–2605.
6. Vapnik, V. N. *The Nature of Statistical Learning Theory*, second ed. Springer, November 1999.