Algorithmic Operation Research
Homework 3

Theodora Panagea - 1115201400135
Anna-Aikaterini Kavvada - 1115201500050

8-11-2019

1 Choosing paths in a network

A communication network aims to create a liaison between two devices or more, a **receiver** and a **transmitter**. It consists of the two aforementioned parties and all in-between devices, called nodes, which are connected with **communication links**. Let's consider a set N that consists of all the nodes, and a set A, which includes all links available in our network. For example, if we take the nodes $i, j \in N$, then the ordered **pair** $(i, j) \in A$. We assume that the link (i, j) can carry up to u_{ij} bits per second. Along that link, a positive charge c_{ij} per bit is transmitted. Each node k generates data, at the rate of b^{kl} bits per second, that have to be transmitted to node l, either through a direct link (k, l), or by tracing a **sequence of links**.

Problem

The problem we have to face is to choose the right path, along which all data reach their intended destinations, while minimizing the cost.
Note: Data with the same origin and destination, can be split and transmitted along different paths.

Formulating as a Linear Programming Problem

We have x_{ij}^{kl} which indicates the amount of data with origin k and destination l, that traverse link (i, j). Let

$$b_i^{kl} = \begin{cases} b^{kl}, & \text{if } i = k \\ -b^{kl}, & \text{if } i = l \\ 0, & \text{otherwise} \end{cases}$$

Therefore, b_i^{kl} is the net inflow at node i, with origin k and destination l, from outside the network. We need to construct a function, which represents the cost we need to minimize. The cost will be the sum of all possible combinations of origin, destination and traverse links, and for each combination, the cost is the charge c_{ij} per bit per second of the traverse link (i, j), multiplied by x_{ij}^{kl}, the amount of data needed to be transmitted from k to l, through that link.

So, our main **goal** is:

$$\text{minimize} \quad \sum_{(i,j)\in A} \sum_{k=1}^{n} \sum_{l=1}^{n} c_{ij} x_{ij}^{kl}$$

$$\text{subject to} \quad \sum_{\{j|(i,j)\in A\}} x_{ij}^{kl} - \sum_{\{j|(j,i)\in A\}} x_{ji}^{kl} = b_i^{kl}, \qquad i,k,l = 1,\dots,n$$

$$\sum_{k=1}^{n} \sum_{l=1}^{n} x_{ij}^{kl} \leqslant u_{ij}, \qquad (i,j) \in A$$

$$x_{ji}^{kl} \leqslant 0, \qquad (i,j) \in A \quad k,l = 1,\dots,n$$

The first constraint is a flow conservation constraint at node i for data with origin k and destination l. The expression

$$\sum_{\{j|(i,j)\in A\}} x_{ij}^{kl}$$

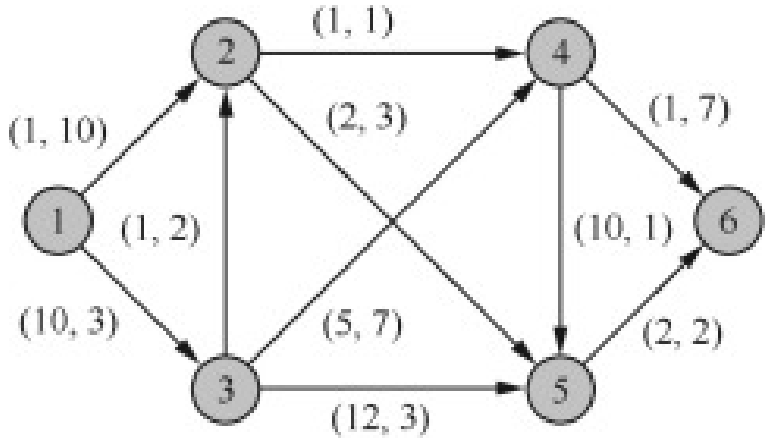
represents the amount of data with origin k and destination l, that **leave** node i along some link. The expression

$$\sum_{\{j|(j,i)\in A\}} x_{ji}^{kl}$$

represents the amount of data with the same origin and destination as the above, but **enter** node i along some link. Finally, b_i^{kl} is the net amount of such data that enter node i from outside the network. The second constraint function expresses the requirement that the total traffic through a link (i,j) cannot exceed the link's capacity.
This problem is a **multicommodity flow** problem. Routing and wavelength assignment (RWA) in optical burst switching of Optical Network would be approached via multicommodity flow formulas.

Extension

If we consider the network as a graph, where a link is an **edge** between two nodes, with weight equal to the positive charge it produces, then all we have to do is to find the shortest path between two nodes, while also checking that each edge can carry the asked amount of bits. There are several algorithms for these problems, such as **Dijkstra**, which is already in use, in terms of routing.
A small example is shown below, where a network is represented as a graph, and the pair above each edge, shows its cost and its capacity.



2 Pattern Classification

Let's consider m examples of **objects** and for each one, a description of its features in terms of an *n-dimensional* **vector**. Thus, the i-th object is described by the vector a_i.

For the above case, we set 2 *classes* of objects. For each example, we know the class it belongs to, thus considering the above set of data as the test set we will use for a classifier.

Goal

At this point our goal, is to design a classifier which, given a new object, suppose k, that does not belong to the test set, will be able to decide in which class k belongs, regarding its features stored in vector a_k.

What Is A Linear Classifier?

Definition: An n-dimensional vector x, which is considered the basis of a *linear combination* of the different features of an object, and a *scalar* x_{n+1}, that defines the range of each class taking in consideration all the features available.
Operation: Given a new object k, with feature vector a_k, the classifier declares it to be an object of the first class, if the following constraint is met:

- $\alpha' x \geq x_{n+1}$

Otherwise, in case the constraint below is met, the item belongs to the second class of objects:

- $\alpha' x < x_{n+1}$

Designing A Linear Classifier

So, how do we design a good linear classifier, using the available examples/test set? A reasonable starting point for that, is the requirement for the classifier to give the correct answer, meaning to classify the available example in the proper class. Let S be the set of examples of the first class. We are then looking for some x and x_{n+1} that satisfy the constraints

- $\alpha_i' x \geq x_{n+1} \quad i \in S$
- $\alpha_i' x < x_{n+1} \quad i \notin S$

Note: the second set of constraints involves a strict inequality and is not quite of the form arising in linear programming. **Resolve Issue:** We can resolve this issue by observing that if some choice of x and x_{n+1} satisfies all of the above constraints, then there exists some other choice if we multiply x and x_{n+1} by a suitably large scalar, that satisfies the below constraints:

- $\alpha_i' x \geq x_{n+1} \quad i \in S$
- $\alpha_i' x \leq x_{n+1} \quad i \notin S$

It is like curving a convex hull around the original scalar of the second class, in order to satisfy the demands of a linear program.

Conclusion

We conclude that the search for a linear classifier which is consistent with all available examples is a problem of finding a set of values applicable for the decision variable - the second class in this case. This set satisfies the entire constraints provided in the optimization problem (*feasible solution*).The feasible region of the optimization problem is defined by all the set of the feasible solutions. In most of the optimization algorithms first, an attempt is made to find the feasible solution and then another attempt is made to locate another feasible solution which will improve the objective function value. The process of making an attempt to search the improving feasible solution goes on until there are no chances of further improvement or any other criteria of stopping this process is met.

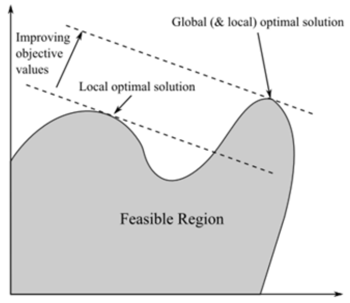


Figure 1:

Extension

Nowadays, we see many applications of Linear Classifiers in Computer Science (i.e. Data Mining, Machine Learning).Below follow two widely known competitive algorithms used to solve Pattern Classification Problems:

- **Multinomial Naive Bayes:**With a multinomial event model, samples (feature vectors) represent the frequencies with which certain events have been generated by a multinomial $(p_1, \ldots, p_n)$ where p_i is the probability that event i occurs (or K such multinomials in the multiclass case). A feature vector $\mathbf{x} = (x_1, \ldots, x_n)$ is then a histogram, with x_i counting the number of times event i was observed in a particular instance. This is the event model typically used for document classification, with events representing the occurrence of a word in a single document (see bag of words assumption). The likelihood of observing a histogram x is given by

  $$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i {p_{ki}}^{x_i}$$

  The multinomial naive Bayes classifier becomes a linear classifier when expressed in *log*-space:

  $$\begin{aligned} \log p(C_k \mid \mathbf{x}) &\propto \log \left(p(C_k) \prod_{i=1}^{n} {p_{ki}}^{x_i} \right) \\ &= \log p(C_k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki} \\ &= b + \mathbf{w}_k^\top \mathbf{x} \end{aligned}$$

  where $b = \log p(C_k)$ and $w_{ki} = \log p_{ki}$.

- **SVM-Support Vector Machine:**We are given a training dataset of n points of the form

  $$(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n),$$

  where the y_i are either 1 or -1, each indicating the class to which the point $\vec{x}_i$ belongs. Each $\vec{x}_i$ is a p-dimensional real vector. We want to find the "maximum-margin hyperplane" that divides the group of points $\vec{x}_i$ for which $y_i = 1$ from the group of points for which $y_i = -1$, which is defined so that the distance between the hyperplane and the nearest point $\vec{x}_i$ from either group is maximized.

Any hyperplane can be written as the set of points $\vec{x}$ satisfying

$$\vec{w}\dot{\vec{x}} - b = 0,$$

where $\vec{w}$ is the (not necessarily normalized) normal vector to the hyperplane. This is much like Hesse normal form, except that $\vec{w}$ is not necessarily a unit vector. The parameter $\frac{b}{\|\vec{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\vec{w}$.

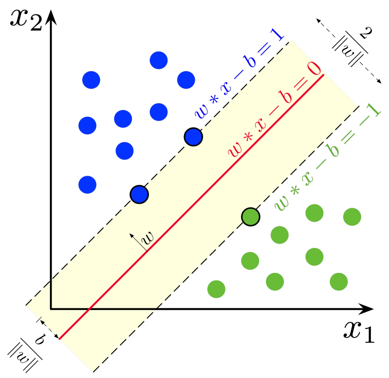


Figure 2: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

Hesse normal form: it is an equation used in analytic geometry, and describes a line in $\mathbb{R}^2$ or a plane in Euclidean space $\mathbb{R}^3$ or a hyperplane in higher dimensions.It is primarily used for calculating distances.
It is written in vector notation as

$$\vec{r} \cdot \vec{n}_0 - d = 0.$$