Examples of Linear Programming Problems
Pattern Classification

Σιώρος Βασίλειος
Ανδρινοπούλου Χριστίνα

Νοέμβριος 2019

1 Feature Vectors

Έστω **m** αντικείμενα, τα οποία επιθυμούμε να ταξινομήσουμε σε δύο διακριτές ομάδες, με κάθε αντικείμενο να ανήκει αυστηρά σε μία μόνο ομάδα.

Παραδείγματος χάριν φυτά, που επιθυμούμε να διαχωρίσουμε σε εσπεριδοειδή και μη.

Θεωρούμε πως τα αντικείμενα αυτά περιγράφονται επαρκώς από **n** το πλήθος χαρακτηριστικά.

Στην περίπτωση της ταξινόμησης φυτών, σε εσπεριδοειδή και μη, τα χαρακτηριστικά αυτά θα μπορούσαν να είναι το ύψος του δέντρου, αν είναι φυλλοβόλο ή όχι, δηλαδή αν ρίχνει τα φύλλα του το χειμώνα και σε τι κλίμα ευδοκιμεί.
Μπορούμε λοιπόν, να αναπαραστίσουμε κάθε αντικείμενο με ένα **n**-διάστατο διάνυσμα.

Παραδείγματος χάριν, έστω ένα στιγμιότυπο x της κλάσης των εσπεριδοειδών με τα εξής **3** χαρακτηριστικά

Ύψος : *8m*
Φυλλοβόλο : Όχι
Κλίμα : Τροπικό ∨ Υποτροπικό ∨ *Ε*ύκρατο

τότε, το παραπάνω αντικείμενο μπορεί να αναπαρασταθεί από το **3**-διάστατο διάνυσμα

$$x^* = [8m,\ \text{Όχι},\ \text{Τροπικό} \vee \text{Υποτροπικό} \vee \text{Εύκρατο}]$$

Είναι προφανές από το προηγούμενο παράδειγμα, ότι τα χαρακτηριστικά των αντικειμένων μπορεί να είναι μη αριθμητικά. Ωστόσο, υπάρχουν μέθοδοι μετατροπής τους σε αριθμητικά κι ως εκ τούτου, το παραπάνω διάνυσμα μπορεί εύκολα να απεικονιστεί στον $\mathbb{R}^3$. Λοιπόν, θα εστιάσουμε στην περίπτωση των αριθμητικών χαρακτηριστικών.

Εις το εξής, οι όροι αντικείμενο και διάνυσμα θα χρησιμοποιούνται ισοδύναμα.

2 Linear Separability

Δεδομένης της προηγουμένως ορισθείσας αναπαράστασης αντικειμένων, θεωρούμε τα σύνολα

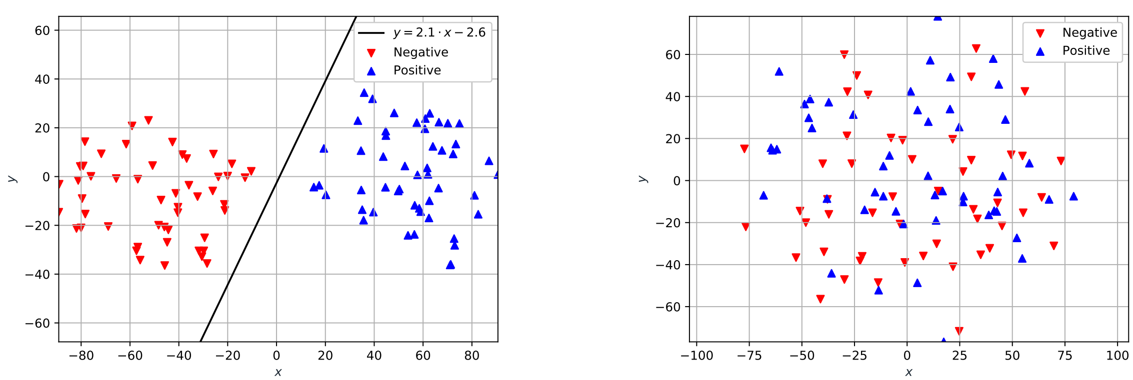
$$K = \{K_1, K_2, \ldots, K_{l_k}\} \qquad \text{όπου } K_i \in \mathbb{R}^n \ \forall i \in [1, \ldots l_k]$$
$$N = \{N_1, N_2, \ldots, N_{l_n}\} \qquad \text{όπου } N_i \in \mathbb{R}^n \ \forall i \in [1, \ldots l_n]$$

Γνωρίζουμε ότι, ένα υπερεπίπεδο διαχωρίζει ένα αφινικό χώρο σε δύο ημιχώρους.

Τα σύνολα K και N χαρακτηρίζονται γραμμικά διαχωρίσιμα, αν υπάρχει υπερεπίπεδο, τέτοιο ώστε τα αντικείμενα, που ανήκουν στο σύνολο K να εντοπίζονται στον έναν ημιχώρο και τα αντικείμενα, που ανήκουν στο σύνολο N στον άλλον ημιχώρο.

Πιο τυπικά θα λέγαμε ότι τα σύνολα K και N χαρακτηρίζονται γραμμικά διαχωρίσιμα, αν υπάρχουν $a \in \mathbb{R}^n$ και $b \in \mathbb{R}$, τέτοια ώστε

$$K \subseteq \{x \in \mathbb{R}^n : a^T \cdot x \geq b\}$$
$$N \subseteq \{x \in \mathbb{R}^n : a^T \cdot x < b\}$$



(a) Γραμμικά διαχωρίσιμα σύνολα

(b) Μη γραμμικά διαχωρίσιμα σύνολα

Figure 1: Παράδειγμα γραμμικής διαχωρισιμότητας στον $\mathbb{R}^2$

3 Pattern Classification via Linear Programming

3.1 An Alternative Characterization of Linear Separability

Αρχικά θα δείξουμε ότι τα σύνολα K και N είναι γραμμικά διαχωρίσιμα αν και μόνο αν υπάρχει $a \in \mathbb{R}^n$ και $b \in \mathbb{R}$, τέτοια ώστε

$$
\begin{aligned}
&a^T \cdot K_i - b \geq 1 && \forall i \in \{1, ..., l_k\} \\
&a^T \cdot N_j - b \leq -1 && \forall j \in \{1, ..., l_n\}
\end{aligned}
$$

Ο ισχυρισμός αυτός είναι ζωτικής σημασίας στην διατύπωση του προβλήματος ως πρόβλημα γραμμικού προγραμματισμού. Χάρις σε αυτόν η αυστηρή ανισότητα $a^T \cdot x < b$ απαλείφεται κι ως εκ τούτου το σύνολο δυνατών λύσεων είναι πλέον κλειστό. Αυτό συνεπάγεται ότι μπορούμε να βρούμε βέλτιση λύση στο προβλήμα.

<u>ΑΠΟΔΕΙΞΗ</u>

$\Leftarrow$ Αν υπάρχει $a \in \mathbb{R}^n$ και $b \in \mathbb{R}$ τέτοια ώστε

$$
\begin{aligned}
&a^T \cdot K_i - b \geq 1 && \forall i \in \{1, ..., l_k\} \\
&a^T \cdot N_j - b \leq -1 && \forall j \in \{1, ..., l_n\}
\end{aligned}
$$

τότε τα K και N είναι γραμμικά διαχωρίσιμα

Όπως είπαμε, τα K και N είναι γραμμικά διαχωρίσιμα όταν υπάρχουν $a \in \mathbb{R}^n$ και $b \in \mathbb{R}$, τέτοια ώστε

$$
\begin{aligned}
K \subseteq \{x \in \mathbb{R}^n : a^T \cdot x \geq b\} \\
N \subseteq \{x \in \mathbb{R}^n : a^T \cdot x < b\}
\end{aligned}
$$

Άρα, προφανώς τα K και N θα είναι γραμμικά διαχωρίσιμα και όταν

$$
\begin{aligned}
K \subseteq \{x \in \mathbb{R}^n : a^T \cdot x \geq b + 1\} \\
N \subseteq \{x \in \mathbb{R}^n : a^T \cdot x < b - 1\}
\end{aligned}
$$

$\Rightarrow$ Αν τα K και N είναι γραμμικά διαχωρίσιμα, τότε υπάρχει $a \in \mathbb{R}^n$ και $b \in \mathbb{R}$ τέτοια ώστε

$$
\begin{aligned}
&a^T \cdot K_i - b \geq 1 && \forall i \in \{1, ..., l_k\} \\
&a^T \cdot N_j - b \leq -1 && \forall j \in \{1, ..., l_n\}
\end{aligned}
$$

Αφού τα K και N είναι γραμμικά διαχωρίσιμα, υπάρχουν $c \in \mathbb{R}^n$ και $b \in \mathbb{R}$, τέτοια ώστε

$$
\begin{aligned}
&c^T \cdot x \geq b && \forall x \in K \\
&c^T \cdot x < b && \forall x \in N
\end{aligned}
$$

Θέτουμε

$$\begin{aligned} m &= \min_{x \in K} c^T \cdot x - \max_{x \in N} c^T \cdot x \\ a &= \frac{2}{m} \cdot c \\ b &= \frac{1}{p} \cdot \left(\min_{x \in K} c^T \cdot x + \max_{x \in N} c^T \cdot x \right) \end{aligned}$$

Έτσι έχουμε

$$\begin{aligned} &\max_{x \in N} a^T \cdot x = \max_{x \in N} \frac{2}{m} \cdot c^T \cdot x && \Leftrightarrow \\ &\max_{x \in N} a^T \cdot x = \max_{x \in N} \frac{1}{m} \cdot \left(c^T \cdot x + c^T \cdot x \right) && \Leftrightarrow \\ &\max_{x \in N} a^T \cdot x = \max_{x \in N} \frac{1}{m} \cdot \left(\min_{x \in K} c^T \cdot x - m + c^T \cdot x \right) && \Leftrightarrow \\ &\max_{x \in N} a^T \cdot x = \frac{1}{m} \cdot \left(\min_{x \in K} c_T \cdot x - m + \max_{x \in N} c^T \cdot x \right) && \Leftrightarrow \\ &\max_{x \in N} a^T \cdot x = \frac{1}{m} \cdot (m \cdot b - m) && \Leftrightarrow \\ &\max_{x \in N} a^T \cdot x = b - 1 \end{aligned}$$

και

$$\begin{aligned} &\min_{x \in K} a^T \cdot x = \min_{x \in K} \frac{2}{m} \cdot c^T \cdot x && \Leftrightarrow \\ &\min_{x \in K} a^T \cdot x = \min_{x \in K} \frac{1}{m} \left(c^T \cdot x + c^T \cdot x \right) && \Leftrightarrow \\ &\min_{x \in K} a^T \cdot x = \min_{x \in K} \frac{1}{m} \left(\max_{x \in N} c^T \cdot x + m + c^T \cdot x \right) && \Leftrightarrow \\ &\min_{x \in K} a^T \cdot x = \min_{x \in K} \frac{1}{m} \left(\max_{x \in N} c^T \cdot x + m + \min_{x \in K} c^T \cdot x \right) && \Leftrightarrow \\ &\min_{x \in K} a^T \cdot x = \frac{1}{m} \cdot (m \cdot b + m) && \Leftrightarrow \\ &\min_{x \in K} a^T \cdot x = b + 1 \end{aligned}$$

Αποδείξαμε επομένως ότι

$$\begin{aligned} a^T \cdot K_i - b &\geq 1 && \forall i \in \{1, \ldots, l_k\} \\ a^T \cdot N_j - b &\leq -1 && \forall j \in \{1, \ldots, l_n\} \end{aligned}$$

3.2 Linear Programming Formulation

Έστω το ακόλουθο πρόβλημα γραμμικού προγραμματισμού

$$\min \frac{1}{l_k}\left(k_1 + \ldots + k_{l_k}\right) + \frac{1}{l_n} \cdot \left(n_1 + \ldots + n_{l_n}\right)$$

υπό τους περιορισμούς

$$\begin{aligned}
k_i &\geq -a^T \cdot K_i + b + 1 && \forall i \in [1, \ldots l_k] \\
n_i &\geq a^T \cdot N_i - b + 1 && \forall i \in [1, \ldots l_n] \\
k_i &\geq 0 && \forall i \in [1, \ldots l_k] \\
n_i &\geq 0 && \forall i \in [1, \ldots l_n]
\end{aligned}$$

Αν υπάρχει βέλτιστη λύση στο παραπάνω πρόβλημα, η οποία, όπως είναι προφανές και από τους περιορισμούς, θα είναι το μηδέν, τότε έχουμε

$$\begin{aligned}
k_{opt} = 0 \Rightarrow 0 &\geq -a_{opt}^T \cdot K_i + b_{opt} + 1 \\
\Leftrightarrow a_{opt}^T \cdot K_i &\geq b_{opt} + 1 && \forall i \in [1, \ldots l_k]
\end{aligned}$$

$$\begin{aligned}
n_{opt} = 0 \Rightarrow 0 &\geq a_{opt}^T \cdot N_i - b_{opt} + 1 \\
\Leftrightarrow a_{opt}^T \cdot N_i &\leq b_{opt} - 1 && \forall i \in [1, \ldots l_n]
\end{aligned}$$

Σε αυτή την περίπτωση, τα σύνολα K και N είναι γραμμικά διαχωρίσιμα και το υπερεπίπεδο, που τα διαχωρίζει, περιγράφεται από την εξίσωση

$$a_{opt}^T \cdot x + b_{opt} = 0$$

4 Modeling Pattern Classification in Python

Σε αυτό το σημείο πρέπει να επισημάνουμε τα εξής:

- Ο γραμμικός ταξινομητής που, θα παρουσιαστεί στη συνέχεια, σχεδιάστηκε στα πλαίσια της παρουσίασης με στόχο την κατανόηση από άτομα που δεν έχουν εντρυφήσει στο θέμα και δεν αποτελεί αναγκαστικά την αποδοτικότερη εκδοχή του.

- Η ταξινόμηση θα εφαρμοστεί σε σημεία του $\mathbb{R}^2$, των οποίων οι συντεταγμένες προκύπτουν βάσει ομοιόμορφης κατανομής. Περιοριστήκαμε στον $\mathbb{R}^2$, καθώς η οπτικοποίηση του είναι ευκολότερη σε σχέση με χώρους μεγαλύτερων διαστάσεων.

4.1 Generating Random Points

Οι μέθοδοι, που θα αναπτύξουμε στη συνέχεια, καλούνται αρχικά με αρνητικό *multiplier*, ο ρόλος του οποίου θα αποσαφηνιστεί στη συνέχεια, για να παράξουν τα σημεία της μίας κλάσης, και έπειτα με θετικό *multiplier*, για να παράξουν τα σημεία της δεύτερης κλάσης.

4.1.1 Linearly Separable Dataset

Αρχικά, αναπτύσσουμε την μέθοδο *get_dispositioned_point*, η οποία δεδομένης μίας ευθείας $f(x) = a \cdot x + b$, ενός σημείου p με συντεταγμένες (x, y) και μίας ποσότητας *distance* υπολογίζει τις συντεταγμένες του σημείου p_d, το οποίο απέχει απόσταση *distance* απο το σημείο p και βρίσκεται επί της ευθείας y^T, που είναι κάθετη στην δοθείσα ευθεία και διέρχεται από το σημείο (x, y).

Σημειώνουμε πως ο όρος απόσταση χρησιμοποιείται αρκετά ελαστικά, στην περιγραφή της ποσότητας *distance*, καθώς η ποσότητα *distance* δύναται να λάβει τόσο θετικές όσο και αρνητικές τιμές.

```python
x, y = point

if a == 0.0:

    return (x, y + distance)

a = -1.0 / a

b = y - a * x

v = np.subtract([xmax, a * xmax + b], [xmin, a * xmin + b])
v = v / np.linalg.norm(v)

return np.asarray(point) + distance * v
```

Listing 1: Η μέθοδος *get_dispositioned_point*

Στη συνέχεια αναπτύσσουμε τη μέθοδο *generate_separable_group*, η οποία αρχικά παράγει σημεία επί της ευθείας $f(x) = a \cdot x + b$, τα οποία στη συνέχεια μετατοπίζει καταλλήλως μέσω της μεθόδου *get_dispositioned_point*.

```
f = lambda x: line[0] * x + line[1]

offset = int(percentage * number)
number = offset if mulitplier < 0 else number - offset

midpoint = (axis[0] + axis[1]) * percentage
bounds   = (axis[0], midpoint) if mulitplier < 0 else (midpoint, axis[1])

xs, ys = [], []

for _ in range(number):

    x = random.uniform(bounds[0], bounds[1])
    y = f(x)

    d = mulitplier * random.uniform(distance[0], distance[1])

    xx, yy = get_dispositioned_point(line[0], (x, y), d, axis[0], axis[1])

    xs.append(xx)
    ys.append(yy)

return xs, ys
```

Listing 2: Η μέθοδος *generate_separable_group*

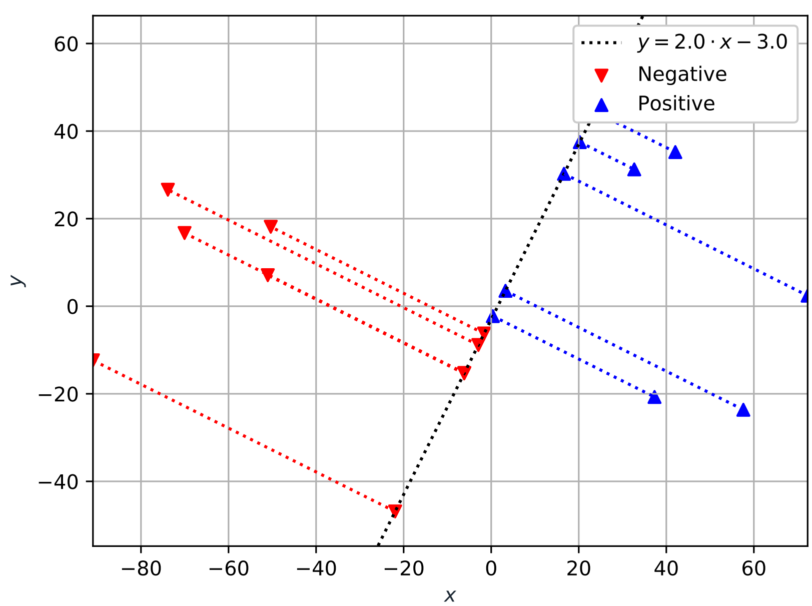


Figure 2: Calling *generate_separable_group* twice to generate 10 points

4.1.2 Non Linearly Separable Dataset

Το πολικό σύστημα συντεταγμένων είναι ένα δισδιάστατο σύστημα συντεταγμένων, στο οποίο η θέση οποιουδήποτε σημείου σε ένα επίπεδο καθορίζεται από την απόσταση του σημείου αυτού από ένα αυθαίρετα επιλεγμένο σημείο αναφοράς και τη γωνία από μία αυθαίρετα επιλεγμένη κατεύθυνση.

Η απόσταση ενός σημείου από το αυθαίρετα επιλεγμένο σημείο αναφοράς, για το οποίο είθισται να επιλέγεται η αρχή των αξόνων, ονομάζεται ακτινική συντεταγμένη, ενώ η γωνία που σχηματίζει η ακτίνα του σημείου με μία αυθαίρετα επιλεγμένη διεύθυνση, συνήθως έναν από τους δύο κύριους άξονες συντεταγμένων, ονομάζεται γωνιακή συντεταγμένη.

Αναπτύσσουμε τη μέθοδο *random_polar_coordinates*, η οποία υπολογίζει την ακτινική και γωνιακή συντεταγμένη βάσει ομοιόμορφης κατανομής, στο εύρος $[r_{min}, r_{max}]$ και $[-\pi, \pi]$ αντίστοιχα, και στη συνέχεια επιστρέφει τις αντίστοιχες καρτεσιανές συντεταγμένες.

```python
r = random.uniform(rmin, rmax)

phi = random.uniform(-math.pi, +math.pi)

return r * math.cos(phi), r * math.sin(phi)
```

Listing 3: Η μέθοδος *random_polar_coordinates*

Στη συνέχεια αναπτύσσουμε τη μέθοδο *generate_random_group*, η οποία χρησιμοποιώντας τη μέθοδο *random_polar_coordinates*, παράγει το ζητούμενο πλήθος τυχαίων σημείων.

```python
offset = int(percentage * number)
number = offset if mulitplier < 0 else number - offset

xs, ys = [], []

for _ in range(number):

    x, y = random_polar_coordinates(distance[0], distance[1])

    xs.append(x)
    ys.append(y)

return xs, ys
```

Listing 4: Η μέθοδος *generate_random_group*

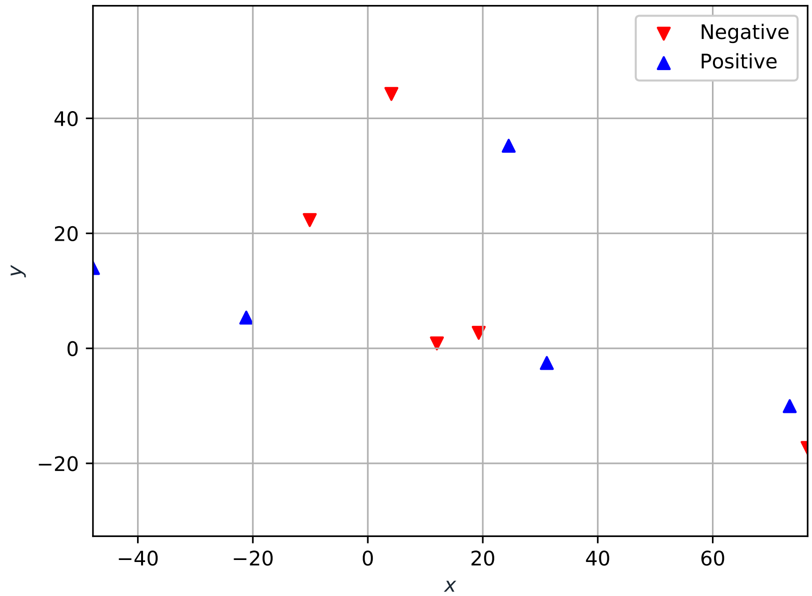


Figure 3: Calling *generate_random_group* twice to generate 10 points

4.2 Linear Classification in $\mathbb{R}^2$

4.2.1 Training the Linear Classifier

Ένας γραμμικός ταξινομητής αρχικά λαμβάνει ως είσοδο ένα συνόλο αντικειμένων, η κλάση καθενός από τα οποία είναι γνωστή εκ των προτέρων.

Βάσει αυτών των αντικειμένων υπολογίζονται τα $a \in \mathbb{R}^n$ και $b \in \mathbb{R}$, όπως περιγράφηκε στην προηγούμενη ενότητα.

Η ακόλουθη υλοποίηση, στην οποία όλοι οι περιορισμοί, που αναφέραμε, ενοποιούνται υπό το σύστημα $A \cdot x = b$, υποθέτει κάποια εξοικείωση με τη βιβλιοθήκη *SciPy* της *Python* και πιο συγκεκριμένα με τη μέθοδο linprog, παράδειγμα χρήσης της οποίας μπορείτε να βρείτε εδώ.

```python
A = []

for y in range(1, len(Y) + 1):

    l = [0] * (y - 1)
    r = [0] * (len(Y) + len(Z) - y)

    A.append(l + [-1] + r + [-1 * Y[y - 1][0], -1 * Y[y - 1][1], +1])

for z in range(1, len(Z) + 1):

    l = [0] * (len(Y) + z - 1)
    r = [0] * (len(Z) - z)

    A.append(l + [-1] + r + [+1 * Z[z - 1][0], +1 * Z[z - 1][1], -1])

for y in range(1, len(Y) + 1):

    l = [0] * (y - 1)
    r = [0] * (len(Y) + len(Z) - y)

    A.append(l + [-1] + r + [0, 0, 0])

for z in range(1, len(Z) + 1):

    l = [0] * (len(Y) + z - 1)
    r = [0] * (len(Z) - z)

    A.append(l + [-1] + r + [0, 0, 0])

A = np.asarray(A)

lower, upper = [0] * (len(Y) + len(Z)), [-1] * (len(Y) + len(Z))

b = np.asarray(upper + lower)
```

Listing 5: Calculating matrix **A** and vector **b**

```python
lower, upper = [0, 0, 0], [1 / len(Y)] * len(Y) + [1 / len(Z)] * len(Z)

c = np.asarray(upper + lower)

result = linprog(c, A_ub=A, b_ub=b, bounds=(None, None))
```

Listing 6: Defining vector **c** and finding a solution to the problem

```python
a, b, c = result.x[-3:]

a, b = -(a / b), (c / b)
```

Listing 7: Determining the slope and the y-intercept of the separating line

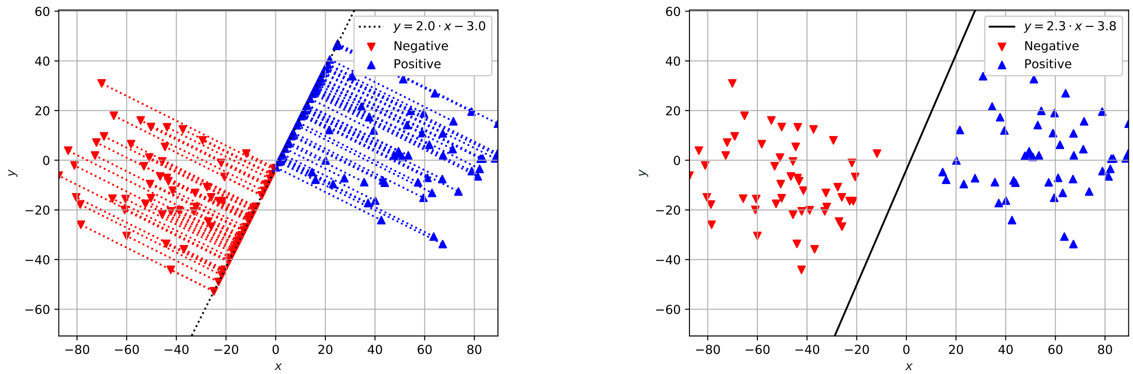


Figure 4: Παράδειγμα επι γραμμικά διαχωρίσιμων υποσυνόλων του $\mathbb{R}^2$

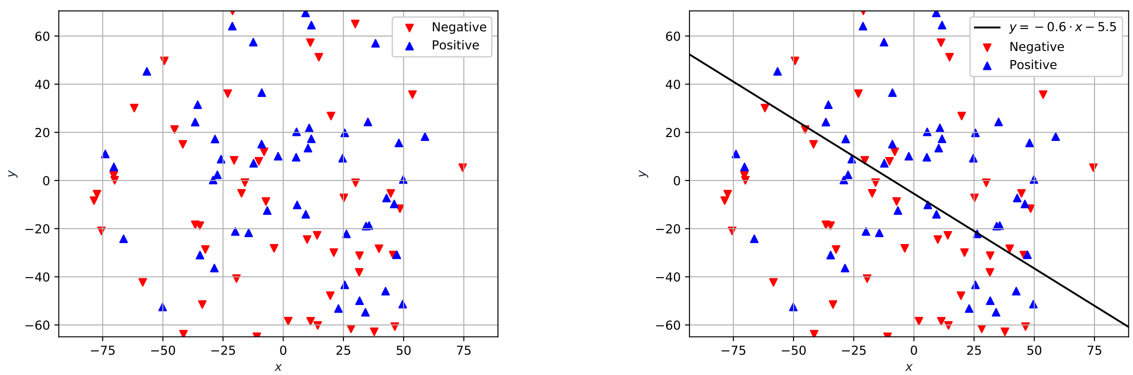


Figure 5: Παράδειγμα επι μη γραμμικά διαχωρίσιμων υποσυνόλων του $\mathbb{R}^2$

4.2.2 Testing the Linear Classifier

Χρησιμοποιούμε τυχαία στοιχεία του $\mathbb{R}^2$, για να ελέγξουμε την ακρίβεια του γραμμικού ταξινομητή, που αναπτύξαμε.

Για κάθε ένα, από αυτά τα αταξινόμητα αντικείμενα, x, το πρόσημο της παράστασης $a^T \cdot x - b$, υποδεικνύει την κλάση στην οποία είναι πιθανότερο να ανήκει το αντικείμενο αυτό.

```
xs, ys = [], []

for _ in range(number):

    xs.append(random.uniform(xaxis[0], xaxis[1]))
    ys.append(random.uniform(yaxis[0], yaxis[1]))

return xs, ys
```

Listing 8: Η μέθοδος *generate_random_points*

```
f = lambda x: a * x + b

xs, ys = generate_random_points((xmin, xmax), (ymin, ymax), args.extra)

xsa, ysa, xsb, ysb = [], [], [], []

for i in range(len(xs)):

    if (f(xs[i]) > ys[i]) == (f(xa[0]) > 0):

        xsa.append(xs[i])
        ysa.append(ys[i])

    else:

        xsb.append(xs[i])
        ysb.append(ys[i])
```

Listing 9: Η υλοποίηση της ταξινόμησης βάσει του προσήμου της παράστασης $a^T \cdot x - b$

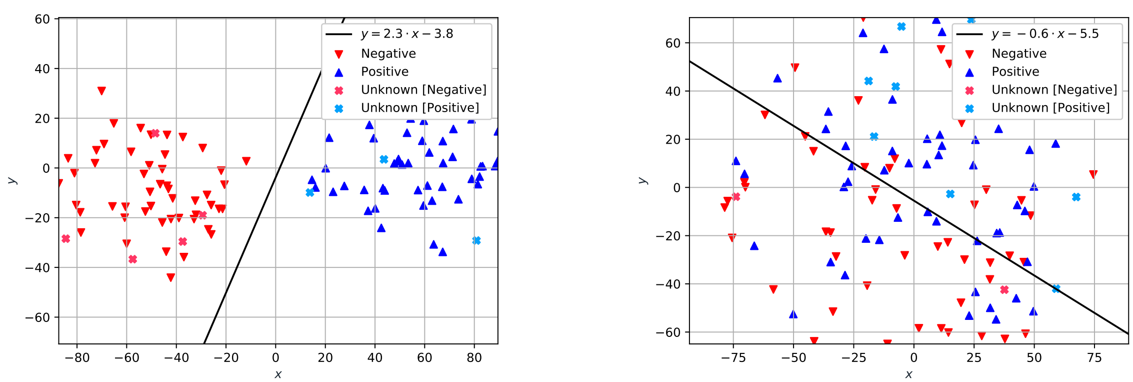


Figure 6: Ταξινόμηση σημείων των οποίων την κλάση δεν γνωρίζουμε εκ των προτέρων