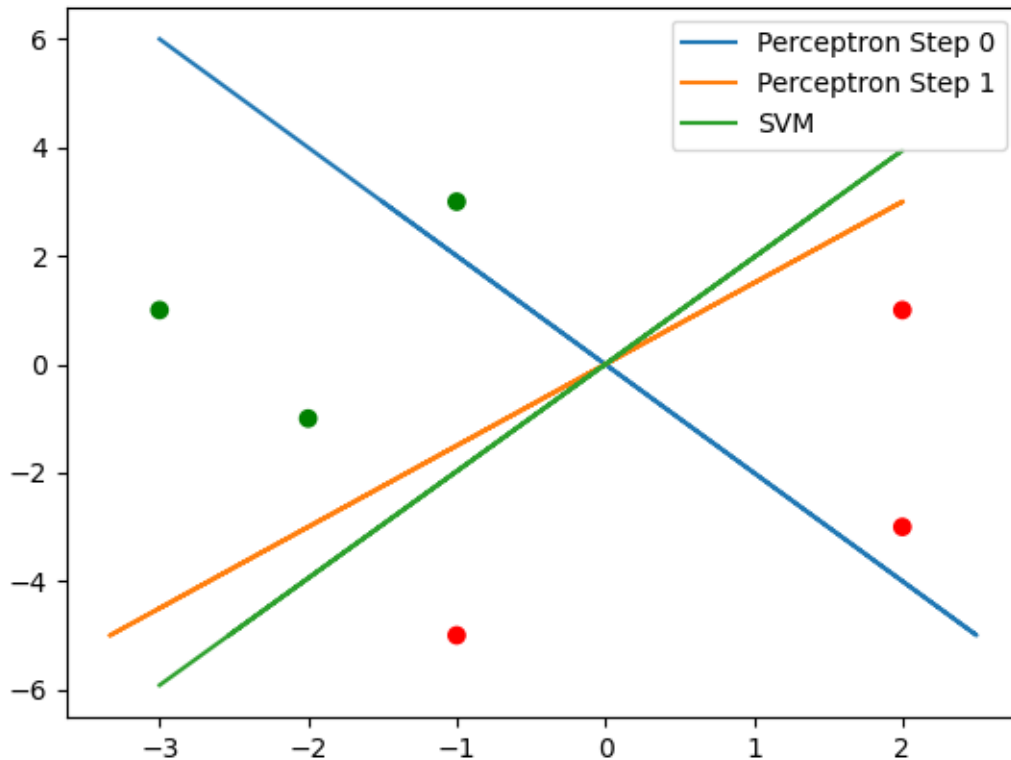


Exercise 1

See Referencesappendix for code.



(a) Perceptron Learning:

Updating vector $w = (0, 0)$ using $(x, y) = ((2, 1), -1)$
 $w = (-2, -1) \rightarrow w = (0, 0) + y = -1 * x = (2, 1)$

Updating vector $w = (-2, -1)$ using $(x, y) = ((-1, 3), 1)$
 $w = (-3, 2) \rightarrow w = (-2, -1) + y = 1 * x = (-1, 3)$

Margin: 0.3076923076923077

(b) SVM Learning:

$w^* : (-0.6074814098863559, 0.30790724037405226)$
 Margin: 1.9555332420486629

Exercise 2

- (a) $\hat{w} = \mathbf{1} = (1, \dots, 1) \in \{1\}^n$ is a suitable weight vector, since $\langle \hat{w}, x \rangle$ is only positive, iff x contains more 1's than -1's.
- (b) $\lambda = n$, since $\|x\|$ is maximum when x consists of either only 1's or only -1's.
 $\gamma = \frac{1}{n}$ since the margin is minimal for a x which consists of an by one number off amount of 1's and -1's. Thus, $\frac{|\langle w, x \rangle|}{\|w\|} = \frac{1}{n}$
 Using Theorem 1.13 we can derive that the perceptron algorithm finds a linear separator after at most $\left(\frac{\lambda}{\gamma}\right)^2 = \left(\frac{n}{\frac{1}{n}}\right)^2 = n^4$ updates.

Exercise 4

See Referencesappendix for code.

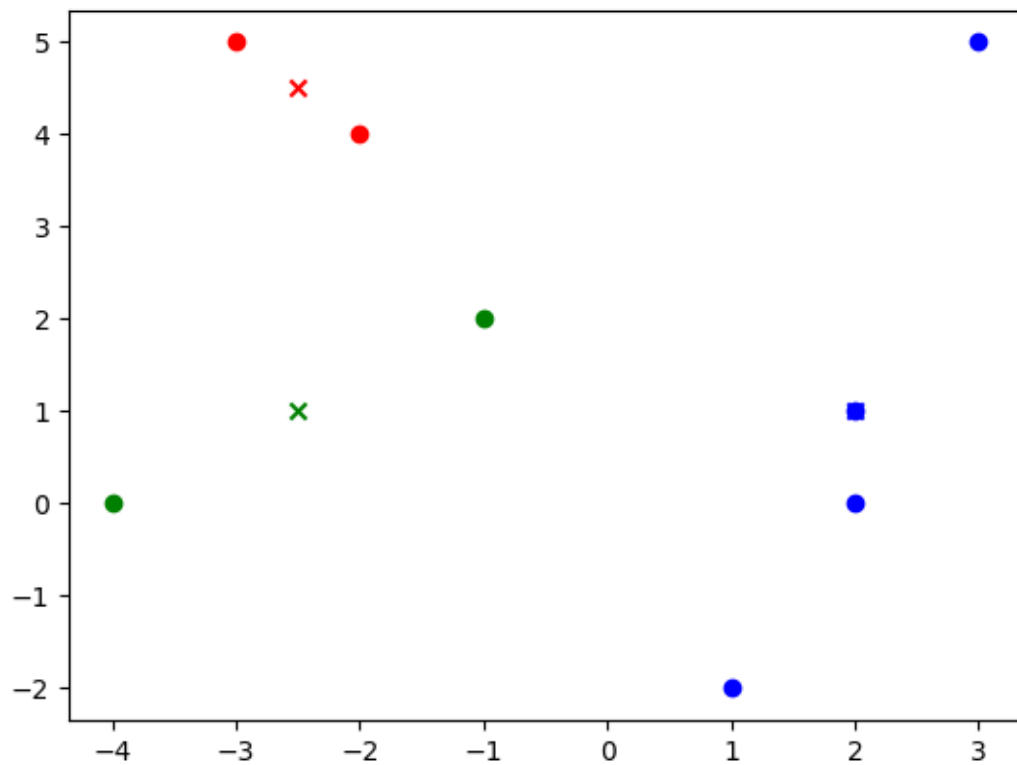
- (a) Clusters: $[[], [], []]$
 Centres: $[(-3, 5), (-2, 4), (-1, 2)]$

Clusters: $[[-3, 5], [-2, 4], [-1, 2], (-4, 0), (1, -2), (2, 0), (2, 1), (3, 5)]]$
 Centres: $[(-3.0, 5.0), (-2.0, 4.0), (0.5, 1.0)]$

Clusters: $[[-3, 5], [-2, 4], (-4, 0), [-1, 2], (1, -2), (2, 0), (2, 1), (3, 5)]]$
 Centres: $[(-3.0, 5.0), (-3.0, 2.0), (1.4, 1.2)]$

Clusters: $[[-3, 5], (-2, 4), [-1, 2], (-4, 0)], [(1, -2), (2, 0), (2, 1), (3, 5)]]$
 Centres: $[(-2.5, 4.5), (-2.5, 1.0), (2.0, 1.0)]$

Final Clusters: $[[-3, 5], (-2, 4)], [-1, 2], (-4, 0)], [(1, -2), (2, 0), (2, 1), (3, 5)]]$
 Final Centres: $[(-2.5, 4.5), (-2.5, 1.0), (2.0, 1.0)]$



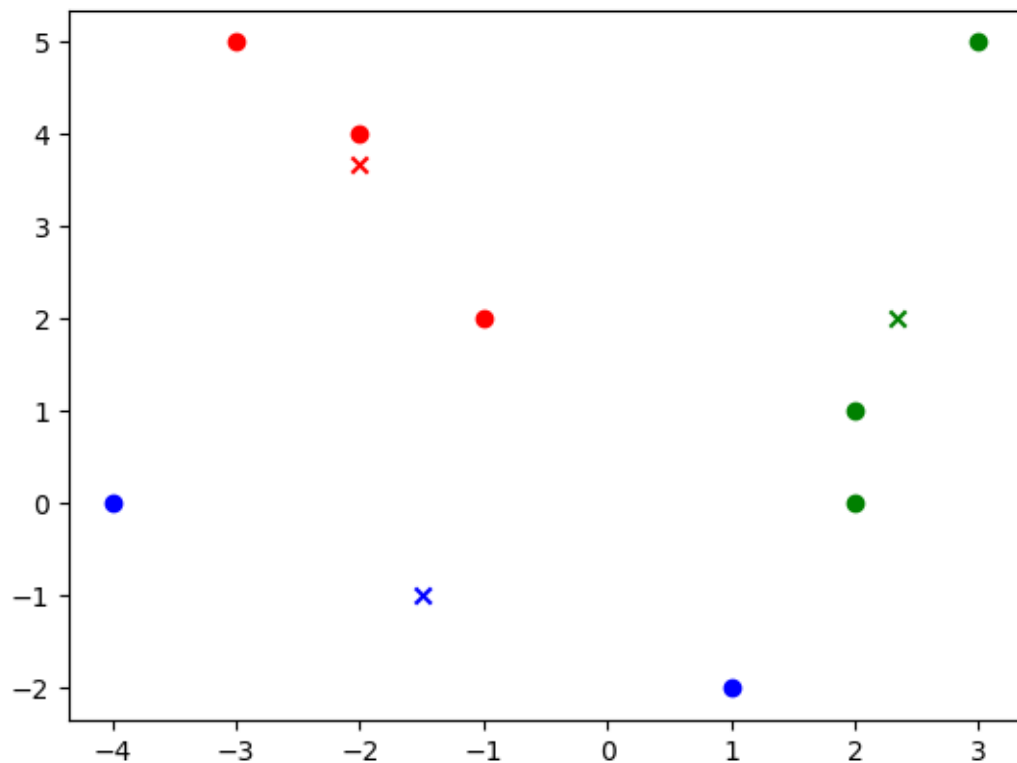
(b)

(c)

(d)

$$z^1 = x_1 = (-3, 5), z^2 = x_8 = (3, 5), z^2 = x_4 = (-4, 0)$$

produces following clustering:



(e)

Appendix

Code for Exercise 1

```

1 from matplotlib import pyplot
2 from sklearn.svm import LinearSVC
3
4 S = [
5     ((2, 1), -1),
6     ((-1, 3), 1),
7     ((-3, 1), 1),
8     ((-2, -1), 1),
9     ((-1, -5), -1),
10    ((2, -3), -1),
11 ]
12
13
14 def plot(S, w_percls, w_svm):
15     # scatter points
16     x_values = [s[0][0] for s in S]
17     y_values = [s[0][1] for s in S]
18     colors = ['green' if s[1] == 1 else 'red' for s in S]
19
20     pyplot.scatter(x_values, y_values, c=colors)
21
22     # plot linear separators
23     x_min = min(x_values)

```

```
24     x_max = max(x_values)
25     y_min = min(y_values)
26     y_max = max(y_values)
27
28     w_list = list(w_percs + [w_svm])
29
30     for i in range(len(w_list)):
31         w = w_list[i]
32         ortho_w = (-w[1], w[0])
33
34         p_1 = (x_min, ortho_w[1] * (x_min / ortho_w[0]))
35         p_2 = (x_max, ortho_w[1] * (x_max / ortho_w[0]))
36         p_3 = (ortho_w[0] * (y_min / ortho_w[1]), y_min)
37         p_4 = (ortho_w[0] * (y_max / ortho_w[1]), y_max)
38
39         p_x_values = (p_1[0], p_2[0], p_3[0], p_4[0])
40         p_y_values = (p_1[1], p_2[1], p_3[1], p_4[1])
41
42         pyplot.plot(p_x_values, p_y_values, label=('SVM' if w==w_svm else f'
Perceptron Step {i}'))#
43
44     pyplot.legend()
45
46     # save to file
47     pyplot.savefig(f'exercise_01.png')
48
49
50 def sgn(value) -> int:
51     if value > 0:
52         return 1
53     elif value == 0:
54         return 0
55     else:
56         return -1
57
58
59 def dot_product(a, b) -> int:
60     return a[0] * b[0] + a[1] * b[1]
61
62
63 def check_consistency(S, w) -> bool:
64     for s in S:
65         if sgn(dot_product(s[0], w)) != s[1]:
66             return False
67     return True
68
69
70 def perceptron(S) -> list:
71     w_list = list()
72     w = (0, 0)
73     while not check_consistency(S, w):
74         for s in S:
75             if sgn(dot_product(s[0], w)) != s[1]:
76                 w_old = w
77                 # w <- w + yx
78                 w_x = w[0] + s[1] * s[0][0]
79                 w_y = w[1] + s[1] * s[0][1]
80                 w = (w_x, w_y)
81                 w_list.append(w)
82     # printing formatted for latex. Just copy and paste
```

```

83         print(f'Updating vector $w={w_old}$ using $(x,y)={s}$ \\\\'
84         print(
85             f'$w={w}$ \\\rightarrow w={w_old} + y={s[1]} * x={s[0]}$
86         \\\ \n\\bigskip \n')
87         return w_list
88
89 def margin(S, w) -> float:
90     distances = [abs(dot_product(w, s[0]))/(dot_product(w, w)) for s in S]
91     distances = sorted(distances)
92     return distances[0]
93
94
95 def svm(S) -> tuple:
96     classifier = LinearSVC(fit_intercept=False) # force heterogenous (
97     fit_intercept=False)
98     classifier.fit([[s[0][0], s[0][1]] for s in S], [s[1] for s in S])
99     return (classifier.coef_[0][0], classifier.coef_[0][1])
100
101 if __name__ == '__main__':
102     print(f'Perceptron Learning: \\\ \n\\bigskip \n')
103     w_perceps = perceptron(S)
104     print(f'Margin: ${margin(S, w_perceps[-1])}$')
105
106     print(f'SVM Learning: \\\ \n\\bigskip \n')
107     w_svm = svm(S)
108     print(f'$w^*: {w_svm}$ \\\')
109     print(f'Margin: ${margin(S, w_svm)}$')
110
111     plot(S, w_perceps, w_svm)

```

Code for Exercise 4

```

1 from math import sqrt
2 from matplotlib import pyplot
3
4 X = [
5     (-3, 5),
6     (-2, 4),
7     (-1, 2),
8     (-4, 0),
9     (1, -2),
10    (2, 0),
11    (2, 1),
12    (3, 5),
13 ]
14
15 Z = [X[0], X[1], X[2]] # a+b
16 #Z = [X[0], X[7], X[3]] # d
17
18 def plot(C, Z, k=3):
19     colors = ['red', 'green', 'blue']
20     for j in range(k):
21         x_values = [cj[0] for cj in C[j]]
22         y_values = [cj[1] for cj in C[j]]
23         pyplot.scatter(x_values, y_values, c=colors[j])
24         pyplot.scatter([Z[j][0]], [Z[j][1]], marker='x', c=colors[j])
25
26     # save to file
27     pyplot.savefig(f'exercise_04.png')

```

```
28
29
30 def equals_list_of_lists(C_1, C_2) -> bool:
31     ll_1 = list([set(l_1) for l_1 in C_1])
32     ll_2 = list([set(l_2) for l_2 in C_2])
33     for l_1 in list(ll_1):
34         for l_2 in list(ll_2):
35             if l_1 == l_2:
36                 ll_1.remove(l_1)
37                 ll_2.remove(l_2)
38                 break
39
40     return len(ll_1) == 0 and len(ll_2) == 0
41
42
43 def dot_product(a, b) -> float:
44     return a[0] * b[0] + a[1] * b[1]
45
46
47 def k_means(X, Z, k=3) -> tuple:
48     C = [[] for j in range(k)]
49     C_ = list(C) # copy of C_
50     first_iteration = True
51     while not equals_list_of_lists(C, C_) or first_iteration:
52         first_iteration = False
53         C = list(C_)
54         C_ = [[] for j in range(k)]
55
56         print(f'Clusters: {C} \\\\'
57         print(f'Centres: {Z} \\\\'
58
59         for x in X:
60             distances = list()
61             for j in range(k):
62                 # x_i - z_j
63                 tmp = (x[0] - Z[j][0], x[1] - Z[j][1])
64                 distances.append((j, sqrt(dot_product(tmp, tmp))))
65             # get min j
66             min_distance = float('inf')
67             min_j = float('inf')
68             for d in distances:
69                 j = d[0]
70                 distance = d[1]
71                 if distance < min_distance:
72                     min_distance = distance
73                     min_j = j
74                 elif distance == min_distance and j < min_j:
75                     min_j = j
76             # add x_i to C_j
77             C_ = [C_[j] if j != min_j else C_[j] + [x] for j in range(k)]
78             # update z_j
79             Z = [(sum([x[0] for x in C_[j]])/len(C_[j]), sum([x[1] for x in C_[j]
80             return (C_, Z)
81
82
83 if __name__ == '__main__':
84     (C, Z) = k_means(X, Z, k=3)
85     print(f'Final Clusters: {C} \\\\'
86     print(f'Final Centers: {Z} \\\\'
```

87 `plot(C, Z, k=3)`