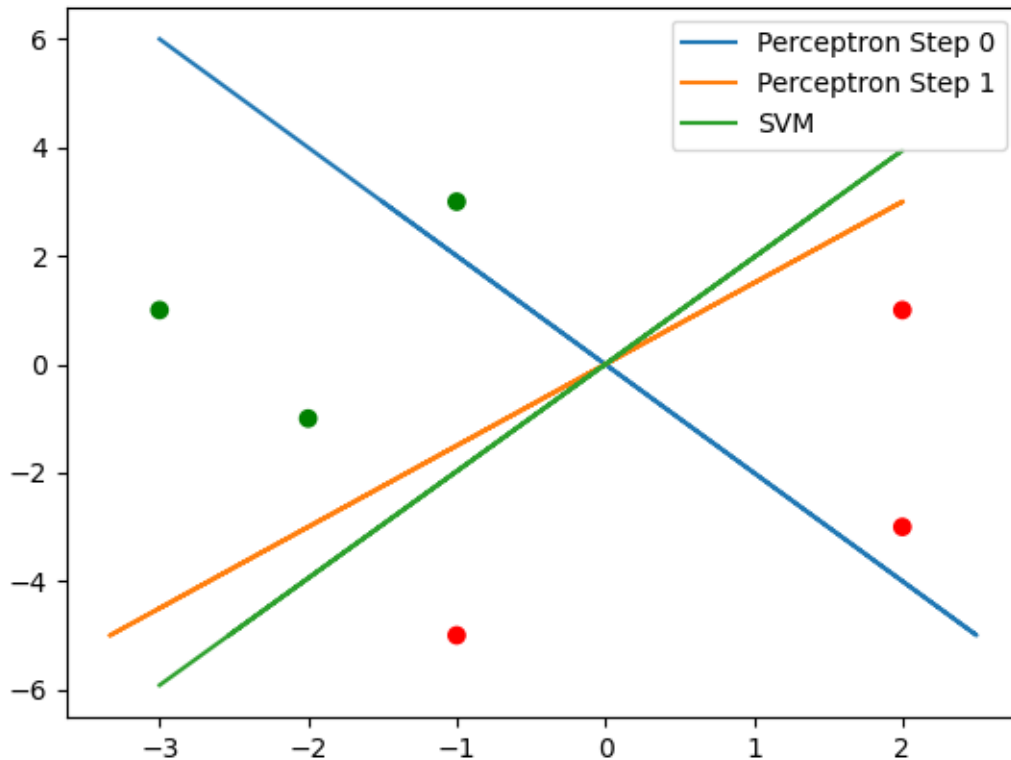# Exercise 1

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: 1.1094003924504583

(b) SVM Learning:

$w^* : (-0.607481409886559, 0.30790724037405226)$
Margin: 1.331824259573374

Exercise 2

(a) $\hat{w} = \mathbf{1} = (1, \ldots, 1) \in \{1\}^n$ 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 $-1'$. 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.

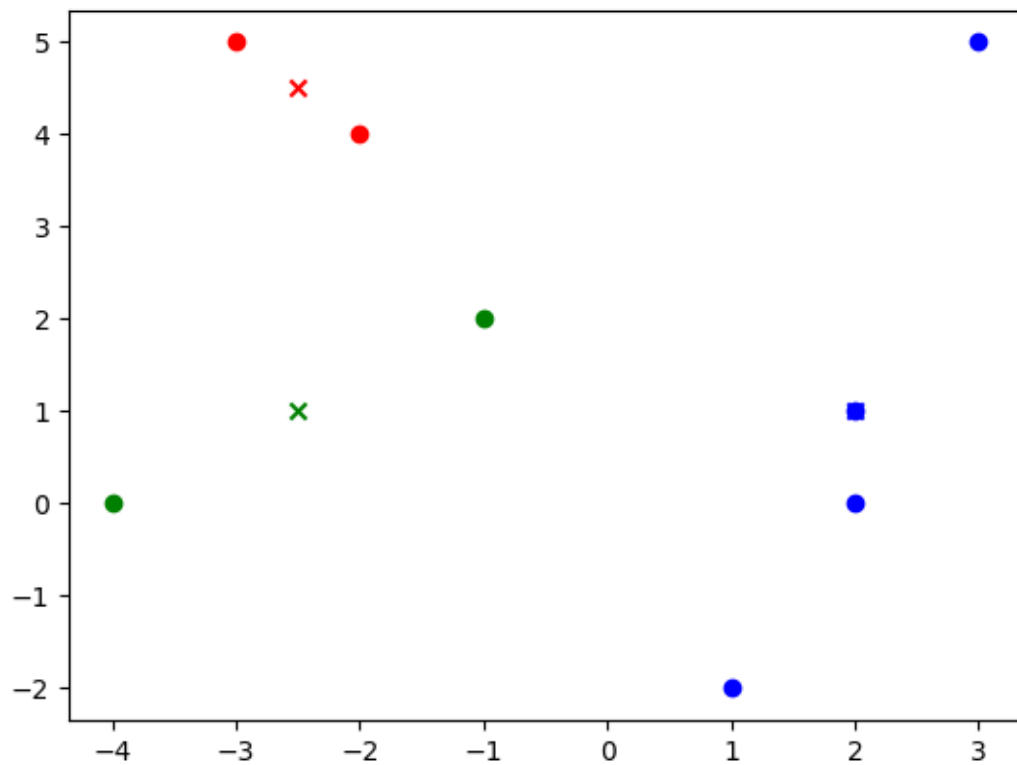# Exercise 4

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)]]
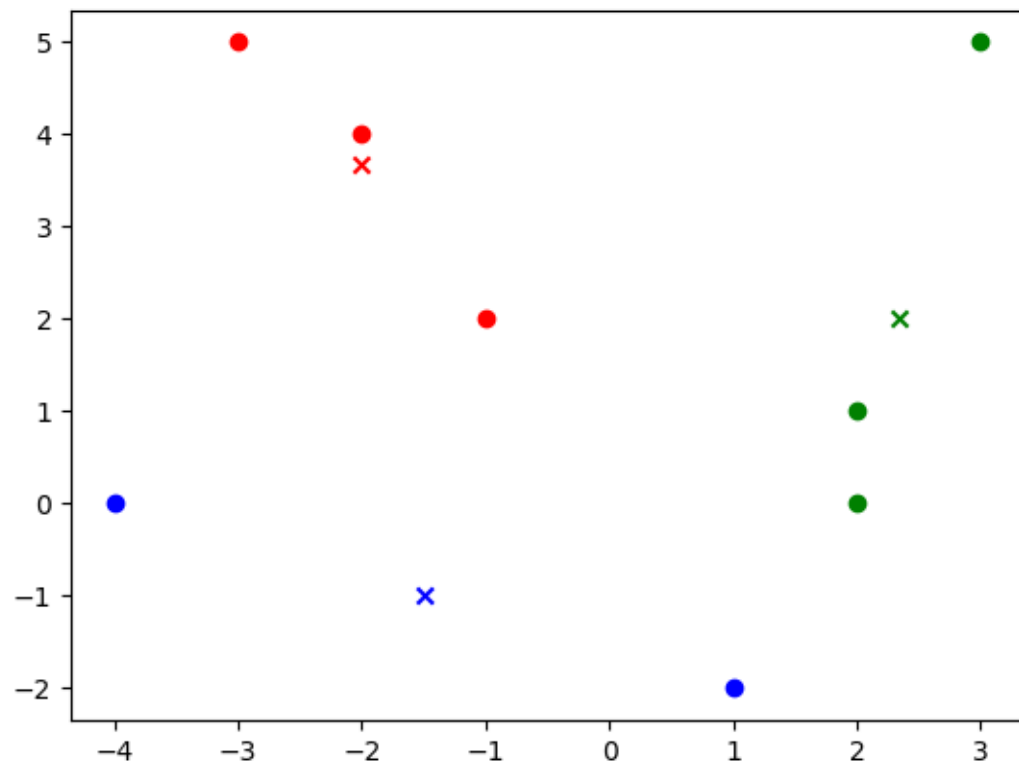Final Centers: [(-2.5, 4.5), (-2.5, 1.0), (2.0, 1.0)]

April 30, 2022

Exercise 02
Algorithmic Foundations of Data Science

Fabian Grob
Simon Michau
Til Mohr

(b)

(c)

(d)

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

produces following clustering:

April 30, 2022

Exercise 02
Algorithmic Foundations of Data Science

Fabian Grob
Simon Michau
Til Mohr

(e) Yes, for example $z^1 = \begin{bmatrix} -4 \\ 3 \end{bmatrix}$, $z^2 = \begin{bmatrix} -1.5 \\ 3 \end{bmatrix}$, $z^3 = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$,

April 30, 2022

Exercise 02
Algorithmic Foundations of Data Science

Fabian Grob
Simon Michau
Til Mohr

Exercise 5

Exercise 6

(a)

$$H(X|Y) + H(X) \Leftrightarrow \left(\sum_{y \in rg(Y)} Pr(Y = y) \left(\sum_{x \in rg(X)} Pr(X = x|Y = y) \cdot \log\left(\frac{1}{Pr(X = x|Y = y)}\right) \right) \right)$$

$$+ \left(\sum_{y \in rg(Y)} Pr(Y = y) \cdot \log\left(\frac{1}{Pr(Y = y)}\right) \right)$$

$$\Leftrightarrow \left(\sum_{y \in rg(Y)} \sum_{x \in rg(X)} \left(Pr(X = x, Y = y) \cdot \log\left(\frac{Pr(Y = y)}{Pr(X = x, Y = y)}\right) \right) \right)$$

$$+ \left(\sum_{y \in rg(Y)} Pr(Y = y) \cdot \log\left(\frac{1}{Pr(Y = y)}\right) \right)$$

$$\Leftrightarrow \left(\sum_{y \in rg(Y)} \sum_{x \in rg(X)} \left(Pr(X = x, Y = y) \cdot (\log(Pr(Y = y) - \log(Pr(X = x, Y = y)))) \right) \right)$$

$$- \left(\sum_{y \in rg(Y)} Pr(Y = y) \cdot \log(Pr(Y = y)) \right)$$

$$\Leftrightarrow \left(\sum_{y \in rg(Y)} \sum_{x \in rg(X)} \left(Pr(X = x, Y = y) \cdot \log(Pr(Y = y)) \right) \right)$$

$$+ H(X, Y)$$

$$- \left(\sum_{\substack{y \in rg(Y) \\ x \in rg(X)}} Pr(Y = y, X = x) \cdot \log(Pr(Y = y)) \right)$$

$$\Leftrightarrow H(X, Y)$$

(b)

Appendix

Code for Exercise 1

```python
from matplotlib import pyplot
from sklearn.svm import LinearSVC
from math import sqrt

S = [
    ((2, 1), -1),
    ((-1, 3), 1),
    ((-3, 1), 1),
    ((-2, -1), 1),
    ((-1, -5), -1),
    ((2, -3), -1),
]


def plot(S, w_percs, w_svm):
    # scatter points
```

```python
17      x_values = [s[0][0] for s in S]
18      y_values = [s[0][1] for s in S]
19      colors = ['green' if s[1] == 1 else 'red' for s in S]
20
21      pyplot.scatter(x_values, y_values, c=colors)
22
23      # plot linear seperators
24      x_min = min(x_values)
25      x_max = max(x_values)
26      y_min = min(y_values)
27      y_max = max(y_values)
28
29      w_list = list(w_percs + [w_svm])
30
31      for i in range(len(w_list)):
32          w = w_list[i]
33          ortho_w = (-w[1], w[0])
34
35          p_1 = (x_min, ortho_w[1] * (x_min / ortho_w[0]))
36          p_2 = (x_max, ortho_w[1] * (x_max / ortho_w[0]))
37          p_3 = (ortho_w[0] * (y_min / ortho_w[1]), y_min)
38          p_4 = (ortho_w[0] * (y_max / ortho_w[1]), y_max)
39
40          p_x_values = (p_1[0], p_2[0], p_3[0], p_4[0])
41          p_y_values = (p_1[1], p_2[1], p_3[1], p_4[1])
42
43          pyplot.plot(p_x_values, p_y_values, label=('SVM' if w==w_svm else f'
    Perceptron Step {i}'))#
44
45      pyplot.legend()
46
47      # save to file
48      pyplot.savefig(f'exercise_01.png')
49
50
51  def sgn(value) -> int:
52      if value > 0:
53          return 1
54      elif value == 0:
55          return 0
56      else:
57          return -1
58
59
60  def dot_product(a, b) -> int:
61      return a[0] * b[0] + a[1] * b[1]
62
63
64  def check_consistency(S, w) -> bool:
65      for s in S:
66          if sgn(dot_product(s[0], w)) != s[1]:
67              return False
68      return True
69
70
71  def perceptron(S) -> list:
72      w_list = list()
73      w = (0, 0)
74      while not check_consistency(S, w):
75          for s in S:
```

```python
76              if sgn(dot_product(s[0], w)) != s[1]:
77                  w_old = w
78                  # w <- w + yx
79                  w_x = w[0] + s[1] * s[0][0]
80                  w_y = w[1] + s[1] * s[0][1]
81                  w = (w_x, w_y)
82                  w_list.append(w)
83                  # printing formatted for latex. Just copy and paste
84                  print(f'Updating vector $w={w_old}$ using $(x,y)={s}$ \\\\')
85                  print(
86                      f'$w={w} \\rightarrow w={w_old} + y={s[1]} * x={s[0]}$
    \\\\ \n\\bigskip \n')
87       return w_list
88
89
90  def margin(S, w) -> float:
91      distances = [abs(dot_product(w, s[0]))/sqrt(dot_product(w, w)) for s in
    S]
92      distances = sorted(distances)
93      return distances[0]
94
95
96  def svm(S) -> tuple:
97      classifier = LinearSVC(fit_intercept=False) # force heterogenous (
    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
102  if __name__ == '__main__':
103      print(f'Perceptron Learning: \\\\ \n\\bigskip \n')
104      w_percs = perceptron(S)
105      print(f'Margin: ${margin(S,w_percs[-1])}$')
106
107      print(f'SVM Learning: \\\\ \n\\bigskip \n')
108      w_svm = svm(S)
109      print(f'$w^*: {w_svm}$ \\\\')
110      print(f'Margin: ${margin(S, w_svm)}$')
111
112      plot(S, w_percs, w_svm)
```

Code for Exercise 4

```python
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):
```

```python
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} \\\\\n')
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
```

```python
79        Z = [(sum([x[0] for x in C_[j]])/len(C_[j]), sum([x[1] for x in C_[j
   ]])/len(C_[j])) if len(C_[j]) != 0 else Z[j] for j in range(k)]
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} \\\\\n')
87    plot(C, Z, k=3)
```