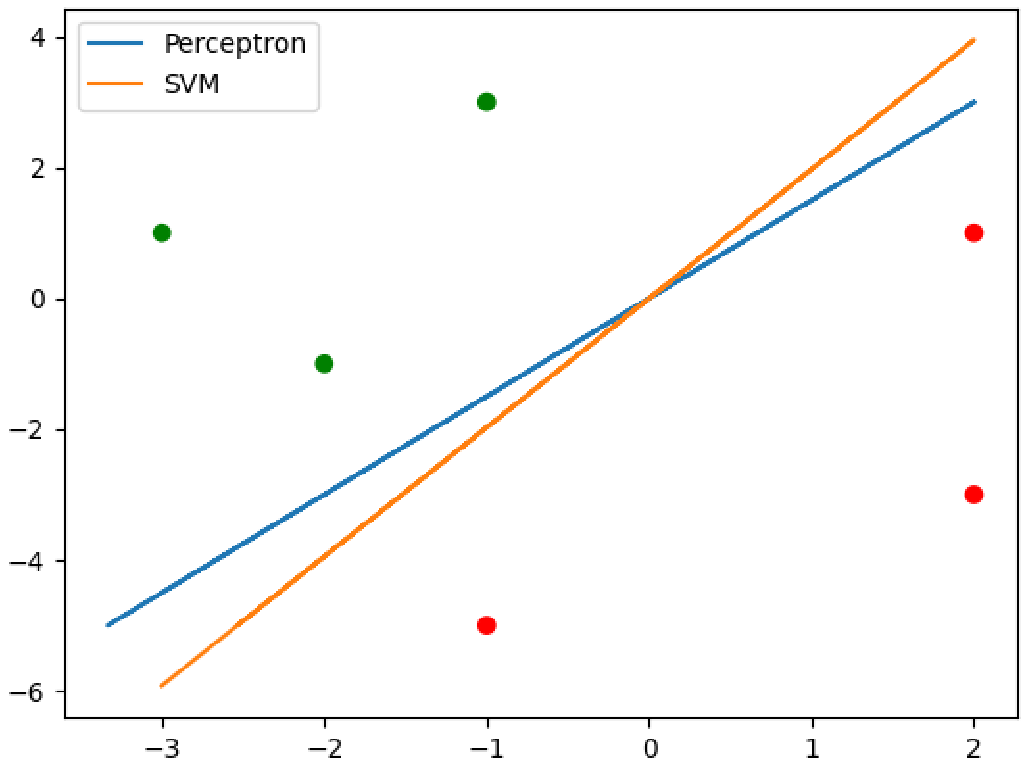
# Exercise 1

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:

0.0 $w^* : (-0.6074814098863559, 0.30790724037405226)$
Margin: 1.9555332420486629

# 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

See Referencesappendix for code.

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

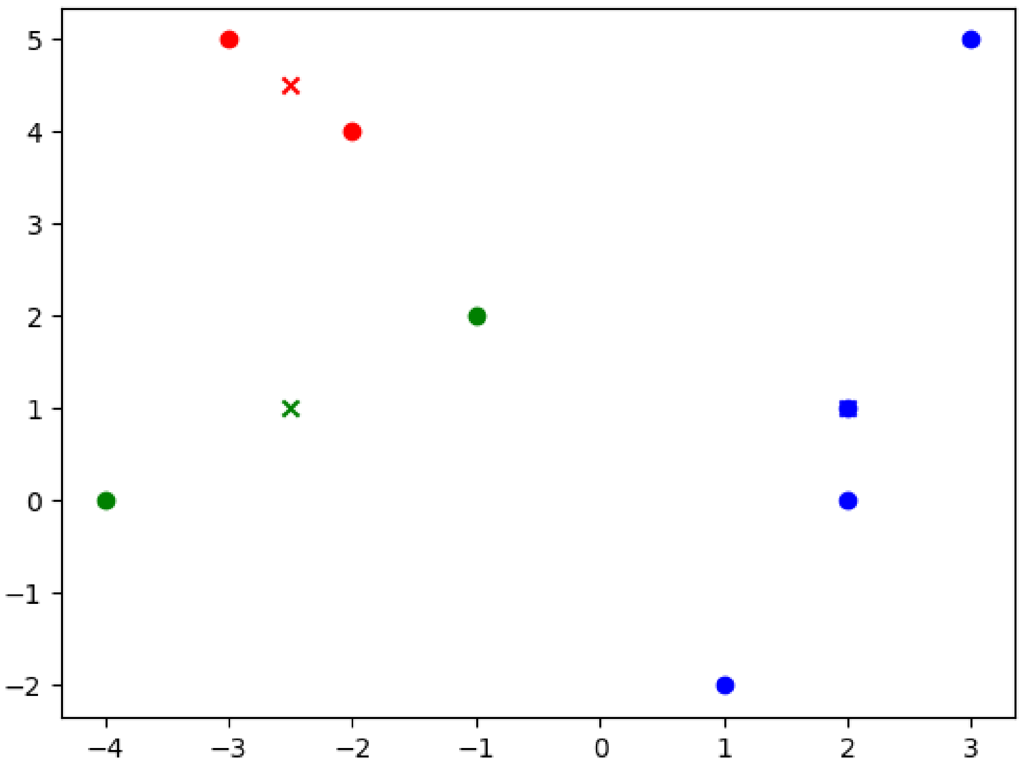
Clusters: [[(-3, 5)], [(-2, 4)], [(-1, 2), (-4, 0), (1, -2), (2, 0), (2, 1), (3, 5)]]
Centres: [(-3.0, 5.0), (-2.0, 4.0), (0.5, 1.0)]

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)]

Fabian Grob
Simon Michau
Til Mohr



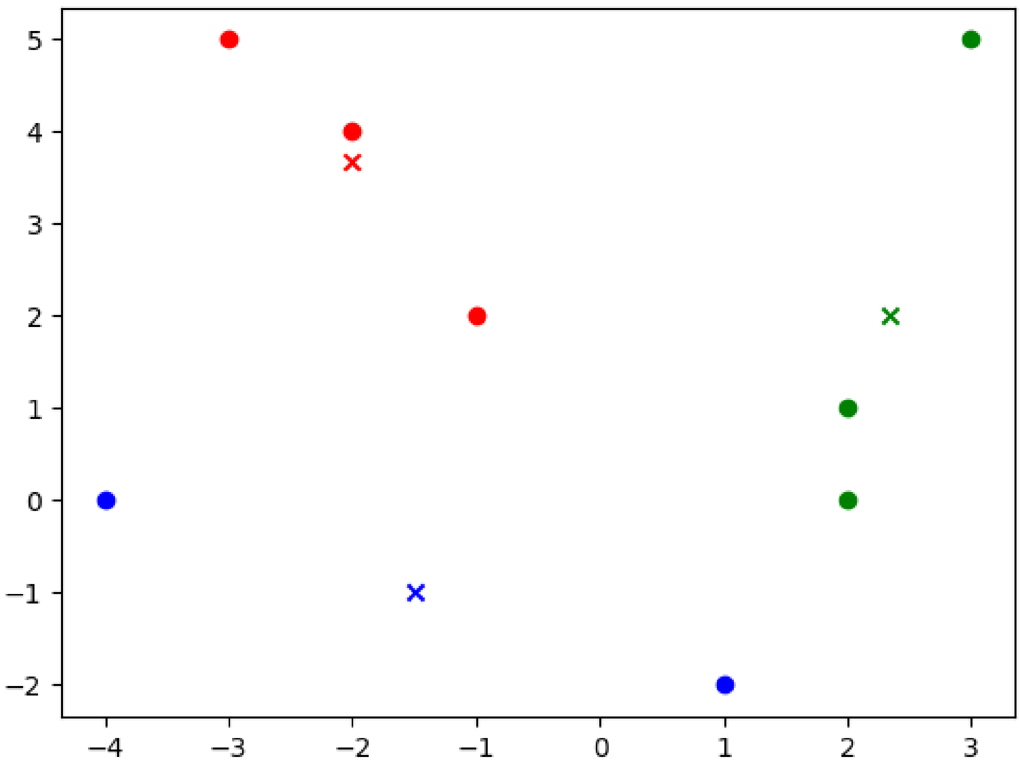

(b)

(c)

(d)

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

produces following clustering:




(e)

# Appendix

## Code for Exercise 1

```
from matplotlib import pyplot
from sklearn.svm import LinearSVC

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


def plot(S, w_perc, w_svm):
    # scatter points
    x_values = [s[0][0] for s in S]
    y_values = [s[0][1] for s in S]
    colors = ['green' if s[1] == 1 else 'red' for s in S]

    pyplot.scatter(x_values, y_values, c=colors)

    # plot linear seperators
    x_min = min(x_values)
```

```python
    x_max = max(x_values)
    y_min = min(y_values)
    y_max = max(y_values)

    for w in [w_perc, w_svm]:
        ortho_w = (-w[1], w[0])

        p_1 = (x_min, ortho_w[1] * (x_min / ortho_w[0]))
        p_2 = (x_max, ortho_w[1] * (x_max / ortho_w[0]))
        p_3 = (ortho_w[0] * (y_min / ortho_w[1]), y_min)
        p_4 = (ortho_w[0] * (y_max / ortho_w[1]), y_max)

        p_x_values = (p_1[0], p_2[0], p_3[0], p_4[0])
        p_y_values = (p_1[1], p_2[1], p_3[1], p_4[1])

        pyplot.plot(p_x_values, p_y_values, label=('Perceptron' if w==w_perc
     else 'SVM'))#

    pyplot.legend()

    # save to file
    pyplot.savefig(f'exercise_01.png')


def sgn(value) -> int:
    if value > 0:
        return 1
    elif value == 0:
        return 0
    else:
        return -1


def dot_product(a, b) -> int:
    return a[0] * b[0] + a[1] * b[1]


def check_consistency(S, w) -> bool:
    for s in S:
        if sgn(dot_product(s[0], w)) != s[1]:
            return False
    return True


def perceptron(S) -> tuple:
    w = (0, 0)
    while not check_consistency(S, w):
        for s in S:
            if sgn(dot_product(s[0], w)) != s[1]:
                w_old = w
                # w <- w + yx
                w_x = w[0] + s[1] * s[0][0]
                w_y = w[1] + s[1] * s[0][1]
                w = (w_x, w_y)
                # printing formatted for latex. Just copy and paste
                print(f'Updating vector $w={w_old}$ using $(x,y)={s}$ \\\\')
                print(
                    f'$w={w} \\rightarrow w={w_old} + y={s[1]} * x={s[0]}$
    \\\\ \n\\bigskip \n')
    return w
```

```python


def margin(S, w) -> float:
    distances = [abs(dot_product(w, s[0]))/(dot_product(w, w)) for s in S]
    distances = sorted(distances)
    return distances[0]


def svm(S) -> tuple:
    classifier = LinearSVC(fit_intercept=False) # force heterogenous (fit_intercept=False)
    classifier.fit([[s[0][0], s[0][1]] for s in S], [s[1] for s in S])
    print(classifier.intercept_)
    return (classifier.coef_[0][0], classifier.coef_[0][1])


if __name__ == '__main__':
    print(f'Perceptron Learning: \\\\ \n\\bigskip \n')
    w_perc = perceptron(S)
    print(f'Margin: ${margin(S,w_perc)}$')

    print(f'SVM Learning: \\\\ \n\\bigskip \n')
    w_svm = svm(S)
    print(f'$w^*: {w_svm}$ \\\\')
    print(f'Margin: ${margin(S, w_svm)}$')

    plot(S, w_perc, w_svm)
```

## Code for Exercise 4

```python
from math import sqrt
from matplotlib import pyplot

X = [
    (-3, 5),
    (-2, 4),
    (-1, 2),
    (-4, 0),
    (1, -2),
    (2, 0),
    (2, 1),
    (3, 5),
]

Z = [X[0], X[1], X[2]] # a+b
#Z = [X[0], X[7], X[3]] # d

def plot(C, Z, k=3):
    colors = ['red', 'green', 'blue']
    for j in range(k):
        x_values = [cj[0] for cj in C[j]]
        y_values = [cj[1] for cj in C[j]]
        pyplot.scatter(x_values, y_values, c=colors[j])
        pyplot.scatter([Z[j][0]], [Z[j][1]], marker='x', c=colors[j])

    # save to file
    pyplot.savefig(f'exercise_04.png')


def equals_list_of_lists(C_1, C_2) -> bool:
    ll_1 = list([set(l_1) for l_1 in C_1])
```

```python
    ll_2 = list([set(l_2) for l_2 in C_2])
    for l_1 in list(ll_1):
        for l_2 in list(ll_2):
            if l_1 == l_2:
                ll_1.remove(l_1)
                ll_2.remove(l_2)
                break

    return len(ll_1) == 0 and len(ll_2) == 0


def dot_product(a, b) -> float:
    return a[0] * b[0] + a[1] * b[1]


def k_means(X, Z, k=3) -> tuple:
    C = [[] for j in range(k)]
    C_ = list(C) # copy of C_
    first_iteration = True
    while not equals_list_of_lists(C, C_) or first_iteration:
        first_iteration = False
        C = list(C_)
        C_ = [[] for j in range(k)]

        print(f'Clusters: {C} \\\\')
        print(f'Centres: {Z} \\\\\n')

        for x in X:
            distances = list()
            for j in range(k):
                # x_i - z_j
                tmp = (x[0] - Z[j][0], x[1] - Z[j][1])
                distances.append((j, sqrt(dot_product(tmp, tmp))))
            # get min j
            min_distance = float('inf')
            min_j = float('inf')
            for d in distances:
                j = d[0]
                distance = d[1]
                if distance < min_distance:
                    min_distance = distance
                    min_j = j
                elif distance == min_distance and j < min_j:
                    min_j = j
            # add x_i to C_j
            C_ = [C_[j] if j != min_j else C_[j] + [x] for j in range(k)]
        # update z_j
        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)]
    return (C_, Z)


if __name__ == '__main__':
    (C, Z) = k_means(X, Z, k=3)
    print(f'Final Clusters: {C} \\\\')
    print(f'Final Centers: {Z} \\\\\n')
    plot(C, Z, k=3)
```