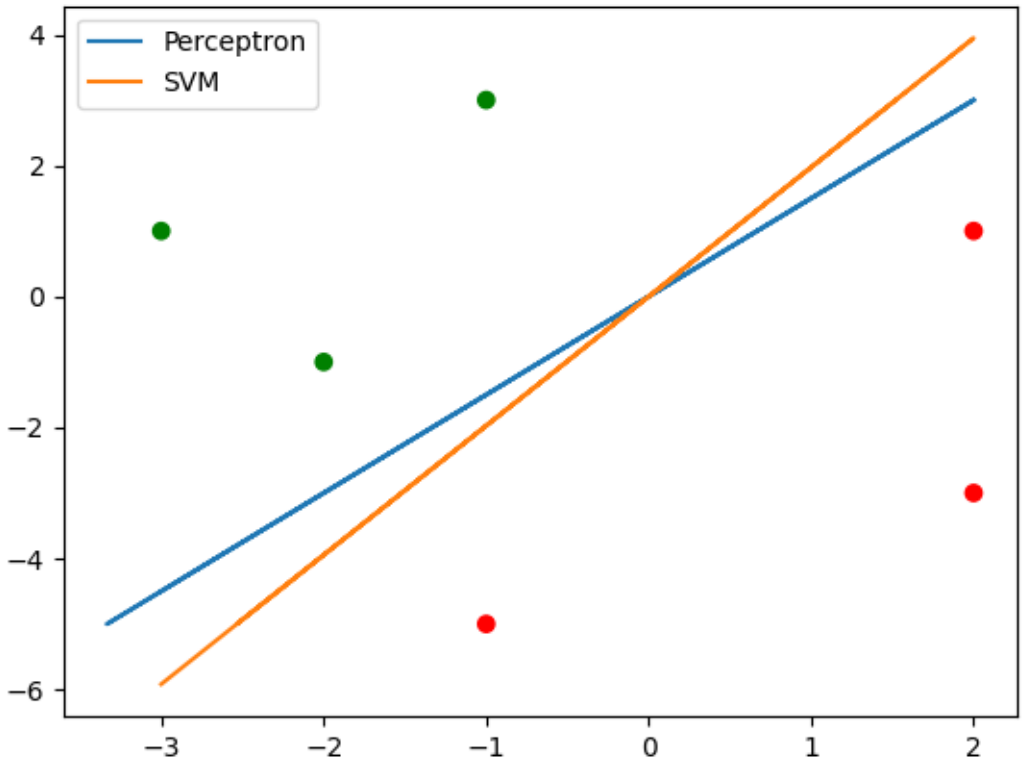
Exercise 1

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

Appendix

Code for Exercise 1

```python
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)
    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


```

```python
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


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)
```