Data Mining & Machine Learning

Computer Exercise 8 - Support Vector Machines

Steinarr Hrafn Höskuldsson

October 2022

Section 1.1

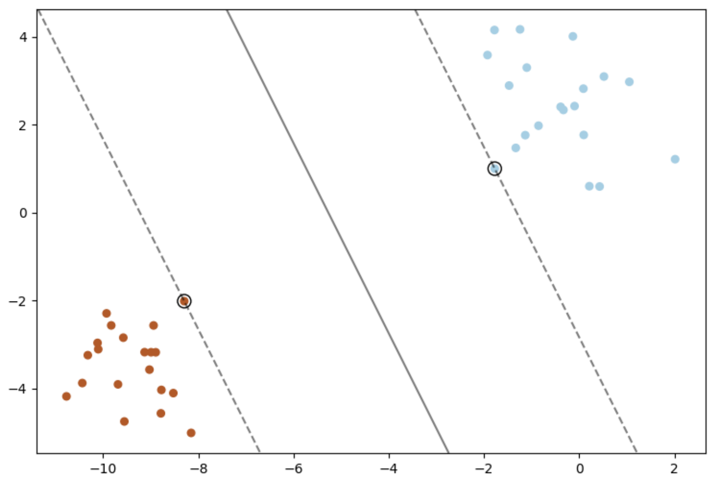

Figure 1: 1_1_1.png The decision boundary and Support Vector margins for generated data.

Section 1.2

In the support vector machine plotted in Figure 1 there is one support vector for each of the two classes. Thus the decision boundary is a straight line.

Section 1.3

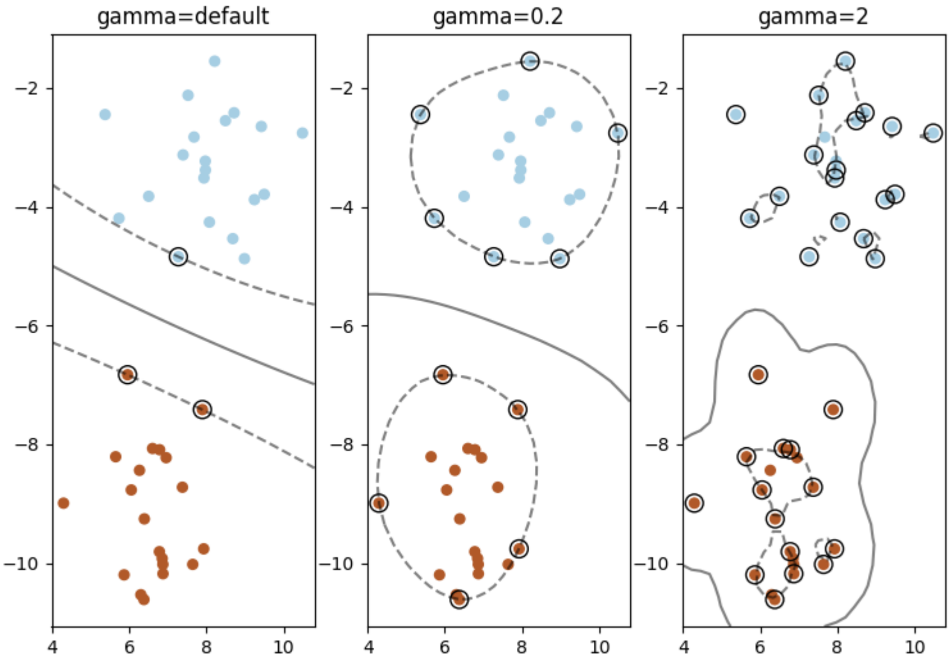

Figure 2: 1_3_1.png The decision boundary and Support Vector margins for three Support Vector machines that differ only in the value of *gamma*.

Section 1.4

When plotting the decision boundary for different values of gamma as can be seen in Figure 2 the amount of support vectors were printed:

```
For default gamma, number of support vectors is: [1 2]
For gamma=0.2, number of support vectors is: [6 5]
For gamma=2, number of support vectors is: [18 15]
```

As can be seen on the plots the shape of the decision boundary gets more and more complicated, With default gamma it is a simple curve with a constant curvature, with gamma=0.2 it appears to be a curve with changing curvature and with gamma=2 it is a very complicated shape.

Section 1.5

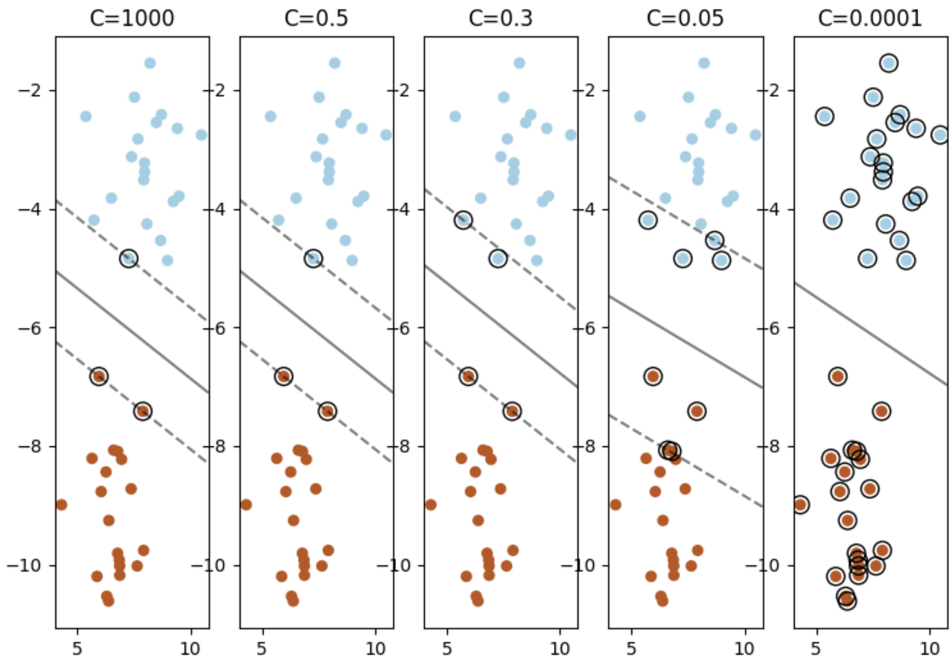

Figure 3: 1_5_1.png The decision boundary and Support Vector margins for five Support Vector machines that differ only in the value of C

Section 1.6

When plotting the decision boundary for different values of C as can be seen in Figure 3 the amount of support vectors were printed:

```
For C=1000, number of support vectors is: [1 2]
For C=0.5, number of support vectors is: [1 2]
For C=0.3, number of support vectors is: [2 2]
For C=0.05, number of support vectors is: [4 4]
For C=0.0001, number of support vectors is: [20 20]
```

As C decreases there are more support vectors inside the margins until at $C = 0.0001$ all the datapoints are support vectors inside the margins. No support vectors are misclassified although the soft margin classifier being used does allow for that. But since the dataset is linearily seperable it does not occur.

Section 2.2

The different kernels were tested on the breast cancer dataset. The results can be found in Table 1

	Accuracy	Test Score	Recall
Linear	95.9%	95.4%	97.3%
Radial Basis	94.7%	94.7%	97.3%
Polynomial	93.6%	93.8%	96.4%

Table 1: The results of Support Vector machine with different kernels tested on the breast cancer dataset.

Looking at the results it looks like the linear kernel gave slightly better results.

Independent

A collection of monthly average weather observations in Reykjavík from 1949 to 2022 was collected from the Icelandic Meteorological Office's website. The objective being to train a classifier to guess the month based on Average weather observations. A short script was written to parse the values for Average Temperature, Average Pressure, Average Sun hours, Average Wind Speed and The number for the corresponding month. These were then parsed into a train test splitter.

A Support vector machine was trained on the training data and then made to predict the month from the test data. The Accuracy of the model was around 70%. Here the confusion matrix can be seen:

	Jan.	*Feb.*	*March*	*April*	*May*	*June*	*July*	*Aug.*	*Sept.*	*Oct.*	*Nov.*	*Dec.*
January	6	0	3	3	1	0	0	0	0	0	0	0
February	0	11	0	0	0	2	0	0	0	0	0	0
March	0	0	6	0	4	0	0	0	0	0	0	0
April	1	0	0	15	0	0	0	0	0	0	0	0
May	0	0	2	0	13	1	0	0	0	0	0	0
June	0	2	0	0	4	11	0	0	0	0	0	0
July	0	1	0	0	0	4	10	0	0	0	0	1
August	0	0	0	0	0	0	2	12	0	0	0	0
September	0	0	0	0	0	0	0	1	8	0	0	2
October	0	0	0	0	0	0	0	0	0	13	5	1
November	0	0	0	0	0	0	0	0	2	10	5	0
December	0	1	0	0	0	0	0	1	0	0	0	12

More than half inaccuracies are only off by one month and, interestingly, the model seems to have had particular issues with classifying between October and November.

Appendix

A Code

Listing 1: The code used

```
# Author:
# Date:
# Project:
# Acknowledgements:
#

# NOTE: Your code should NOT contain any main functions or code that is executed
# automatically.  We ONLY want the functions as stated in the README.md.
# Make sure to comment out or remove all unnecessary code before submitting.


from tools import plot_svm_margin, load_cancer
from sklearn import svm
from sklearn.datasets import make_blobs
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    confusion_matrix

import numpy as np
import matplotlib.pyplot as plt


def _plot_linear_kernel(show=True):
    X, t = make_blobs(40, centers=2)

    svc = svm.SVC(C=1000, kernel="linear")
    svc.fit(X, t)
    print(" number of support vectors for each class:", svc.n_support_)
    print("thus the shape of decision boundary is a line")

    plot_svm_margin(svc, X, t)
    if show:
        plt.show()

if __name__ == "_Q_main__":
    _plot_linear_kernel(show=False)
    plt.gcf().set_figheight(6)
    plt.gcf().set_figwidth(9)
    plt.savefig("08_SVM/1_1_1.png")
    plt.show()


def _subplot_svm_margin(
    svc,
    X: np.ndarray,
    t: np.ndarray,
    num_plots: int,
    index: int,
    title=None
):
    '''
    Plots the decision boundary and decision margins
    for a dataset of features X and labels t and a support
    vector machine svc.
```

```

    Input arguments:
    * svc: An instance of sklearn.svm.SVC: a C-support Vector
    classification model
    * X: [N x f] array of features
    * t: [N] array of target labels
    '''
    # similar to tools.plot_svm_margin but added num_plots and
    # index where num_plots should be the total number of plots
    # and index is the index of the current plot being generated
    plt.subplot(1, num_plots, index)
    if title is not None:
        plt.gca().title.set_text(title)
    plot_svm_margin(svc, X, t)


def _compare_gamma(show=True):
    X, t = make_blobs(n_samples=40, centers=2, random_state=6)

    clf = svm.SVC(C=1000, kernel="rbf")
    clf.fit(X,t)
    print(f"For default gamma, number of support vectors is: {clf.n_support_}")
    _subplot_svm_margin(clf, X, t, 3, 1, title="gamma=default")

    clf = svm.SVC(C=1000, kernel="rbf", gamma=0.2)
    clf.fit(X, t)
    print(f"For gamma=0.2, number of support vectors is: {clf.n_support_}")
    _subplot_svm_margin(clf, X, t, 3, 2, title="gamma=0.2")


    clf = svm.SVC(C=1000, kernel="rbf", gamma=2, )
    clf.fit(X, t)
    print(f"For gamma=2, number of support vectors is: {clf.n_support_}")
    _subplot_svm_margin(clf, X, t, 3, 3, title="gamma=2")

    if show:
        plt.show()

if __name__ == "_Q_main__":
    _compare_gamma(show=False)
    plt.gcf().set_figheight(6)
    plt.gcf().set_figwidth(9)
    plt.savefig("08_SVM/1_3_1.png")
    plt.show()


def _compare_C(show=True):
    X, t = make_blobs(n_samples=40, centers=2, random_state=6)

    Cs = [1000, 0.5, 0.3, 0.05, 0.0001]
    for i, C in enumerate(Cs):
        clf = svm.SVC(C=C, kernel="linear")
        clf.fit(X, t)
        print(f"For C={C}, number of support vectors is: {clf.n_support_}")
        _subplot_svm_margin(clf, X, t, len(Cs), i+1, title=f"{C=}")
    if show:
        plt.show()

if __name__ == "__main__":
    _compare_C(False)
```

```python
    plt.gcf().set_figheight(6)
    plt.gcf().set_figwidth(9)
    plt.savefig("08_SVM/1_5_1.png")
    plt.show()


def train_test_SVM(
    svc,
    X_train: np.ndarray,
    t_train: np.ndarray,
    X_test: np.ndarray,
    t_test: np.ndarray,
):
    '''
    Train a configured SVM on <X_train> and <t_train>
    and then measure accuracy, precision and recall on
    the test set

    This function should return (accuracy, precision, recall)
    '''
    svc.fit(X_train, t_train)
    y = svc.predict(X_test)
    return accuracy_score(t_test, y), precision_score(t_test, y), recall_score(t_test, y)

if __name__ == "_Q_main__":
    (X_train, t_train), (X_test, t_test) = load_cancer()

    kernels = ["linear", "rbf","poly"]
    for kernel in kernels:
        svc = svm.SVC(C=1000, kernel=kernel)
        print(f"Using {kernel=}, gives: {train_test_SVM(svc, X_train, t_train, X_test, t_test)}")

""" Independent"""
if __name__ == "__main__":


    def split_train_test(
        features: np.ndarray,
        targets: np.ndarray,
        train_ratio: float = 0.8
    ):
        '''
        Shuffle the features and targets in unison and return
        two tuples of datasets, first being the training set,
        where the number of items in the training set is according
        to the given train_ratio
        '''
        p = np.random.permutation(features.shape[0])
        features = features[p]
        targets = targets[p]

        split_index = int(features.shape[0] * train_ratio)

        train_features, train_targets = features[0:split_index, :],\
            targets[0:split_index]
        test_features, test_targets = features[split_index:-1, :],\
            targets[split_index: -1]

```

```python
        return (train_features, train_targets), (test_features, test_targets)


    def get_weather_data():
        temps = []
        pressures = []
        suns = []
        winds = []
        month = []
        with open("08_SVM/medalvedur_rvk.txt", 'r') as f:
            for line in f.readlines():
                stuff = line.split("\t")
                temps.append(stuff[3])
                pressures.append(stuff[14])
                suns.append(stuff[16])
                winds.append(stuff[17])
                month.append(stuff[2])
        return split_train_test(np.vstack([temps, pressures, suns, winds]).T, np.array(month).T)

    (X_train, t_train), (X_test, t_test) = get_weather_data()
    # kernels = ["linear", "rbf", "poly"]
    # for kernel in kernels:
    #     svc = svm.SVC(C=1000, kernel=kernel,)
    #     print(f"Using {kernel=}, gives: {train_test_SVM(svc, X_train, t_train, X_test, t_test)}")

    svc = svm.SVC(C=1000, kernel='linear')
    svc.fit(X_train, t_train),
    y = svc.predict(X_test)
    confuse = confusion_matrix(t_test, y)


    def format_matrix(matrix, environment="bmatrix", formatter=str):
        """Format a matrix using LaTeX syntax"""

        if not isinstance(matrix, np.ndarray):
            try:
                matrix = np.array(matrix)
            except Exception:
                raise TypeError("Could not convert to Numpy array")

        if len(shape := matrix.shape) == 1:
            matrix = matrix.reshape(1, shape[0])
        elif len(shape) > 2:
            raise ValueError("Array must be 2 dimensional")

        classes = ("January",
                   "February",
                   "March",
                   "April",
                   "May",
                   "June",
                   "July",
                   "August",
                   "September",
                   "October",
                   "November",
                   "December")
        body_lines = [classes[i] + " & " +
```

```
                    " & ".join(map(formatter, row)) for i, row in enumerate(matrix)]

        body = "\\\\\n".join(body_lines)
        return f"""\\begin{{{environment}}}
        {"(empty) &" + " & ".join(classes)}\\\\
        {body}
        \\end{{{environment}}}"""

    print(format_matrix(confuse))
    print(t_test.shape)
```