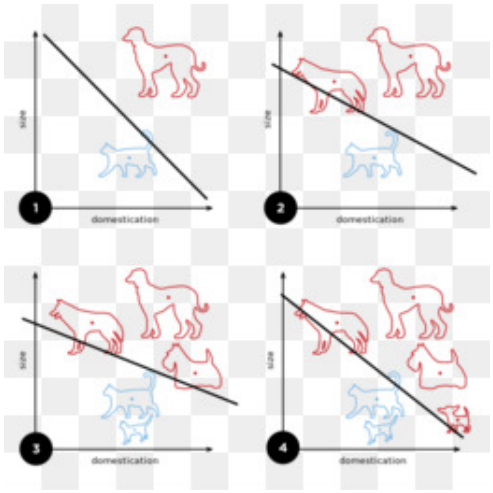
Lab #10
Support Vector Machines
ML_2022: Machine Learning

Location: https://people.sc.fsu.edu/~jburkardt/classes/ml_2022/svm_lab/svm_lab.pdf



Separate two sets of data with a line. And can you do it optimally?

---

1 The Perceptron Algorithm

We can think of the perceptron as an unambitious version of a support vector machine. While (as we will see) the SVM looks for the *optimal* linear formula that separates data, the perceptron iteration will stop when it finds *any* linear formula.

Assuming the data is two dimensional, and that it is indeed linearly separable, then the formula determined by the perceptron can be displayed graphically as a separating line.

To carry out this exercise, retrieve the file `generators.txt`, which contains $n = 56$ rows of data about a set of engines: the index, the RPM measurement, the vibration measurement, and a grade (0 = "defective", 1 = "working");

1. Read the file into `data`, and extract columns 2, 3 and 4 as vectors `rpm`, `vib`, and `grade`.
2. Compute the minimum and maximum values of `rpm` and `vib` and use them to define normalized variables `r` and `v` rescaled to the range $[0, +1]$;
3. Initialize the 3-dimensional column vector `w` to 1; Define the $n \times 3$ array `x` so that column 1 is entirely 1, the second column is the vector `r`, and third column is the vector `v`. Thus, our formula is $f(r, v) = [1, r, v] * w = w_1 + r * w_2 + v * w_3$;
4. Set the "learning rate" `alpha` to 0.01, and carry out the following iteration:

```
e = 1;
do while e is not 0
  e = 0
  for each data item i        % Determine the "grade" defined by w
    if 0 < x(i,1:3) * w then
```

```
      f = 1
    else
      f = 0
    end
    if grade(i) not equal to f  % Compare computed, desired grades
      e = e + 1
      w(1:3) = w(1:3) + alpha * x(i,1:3)' * ( grade(i) - f )
    end
  end
  w = w / norm ( w )              % Normalize w
end
```

Check your results. It should be the case that `x(i,1:3) * w` is positive if the i-th data item has a grade of 1, and negative otherwise. In other words, the perceptron has found weights `w` that classify the data `x`. If you plot the data, you might see that your line has "just barely" separated the data.

```
Table 1  Perceptron weights
-------------------------------------------------------------------------------
f(r,v) = [1,r,v] * w = ___  +  r * ___ + v * ___  (using normalized data)
```

2 Support Vector Machines

We are given a set of n pairs $\{x_i, y_i\}$, where each x_i is a d dimensional data value, and y_i is a classification, which, by SVM convention, will have the value -1 or +1.

We assume our data is linearly separable, that is, that at least one straight line can be drawn that splits the data into its two groups. If so, there will be many such lines. The SVM approach seeks the best such line, which maximizes the separation margin between the two data sets.

Our example will involve a dataset with spatial dimension $d = 2$, so we can easily visualize the results. In any dimension, the equation of the separating SVM line or hyperplane can be represented as

$$f(x) = x' * w + b = 0$$

where x represents a data value stored as a column vector, w is a d-vector of weights, and b is a real number which is a generalization of the y-intercept, and we want $f(x)$ to be positive or negative for data with a +1 or -1 classification respectively.

The SVM system can be rewritten as seeking w and b that:

$$\begin{aligned} \text{minimize: } & \frac{1}{2} w'w \\ \text{subject to: } & y_i(x_i'w + b) \geq 1 \text{ for } 1 \leq i \leq n \end{aligned}$$

In other words, the sign of $f(x)$ correctly classifies each data item x_i as "positive" or "negative".

This is a quadratic programming problem, for which MATLAB offers the function `quadprog()`. This expects a general problem of the form:

$$\text{minimize: } \frac{1}{2} x'Hx + f'x \text{ subject to: } \left\{ \begin{array}{l} A\,x \leq c \\ B\,x = e \text{ (we won't need this!)} \\ lb \leq x \leq ub \text{ (we won't need this!)} \end{array} \right\}$$

If we rearrange our SVM problem data, we can fit the `quadprog()` format:

$$
\begin{aligned}
x \rightarrow wb(1:d+1,1) &\rightarrow \begin{pmatrix} w \\ b \end{pmatrix} \\
H(1:d+1,1:d+1) &\rightarrow \begin{pmatrix} I_d & 0 \\ 0 & 0 \end{pmatrix} \\
f(1:d+1,1) &\rightarrow \begin{pmatrix} 0_{d+1} \end{pmatrix} \\
A(1:n,1:d+1) &\rightarrow -\operatorname{diag}(Y) * \begin{pmatrix} X & 1_n \end{pmatrix} \\
c(1:n,1) &\rightarrow \begin{pmatrix} -1_n \end{pmatrix}
\end{aligned}
$$

Here, I_d is the $d \times d$ identity matrix, $\operatorname{diag}(Y)$ is the $n \times n$ diagonal matrix whose diagonal entries are the values of the y data (the classifications), and X is the $n \times d$ matrix of data values.

Once we have defined the quadratic programming problem, we solve it with a command like:

```
wb = quadprog(H,f,A,c)
```

where the first d values of wb are the weights w and the last is the "intercept" b.

1. Create a data array by loading the file *jet_engines.txt*.
2. Create vectors `rpm`, `vib` and `grade` from columns 2, 3 and 4 of the data. Note that `grade` is +1 or -1.
3. Create the necessary arrays `H`, `f`, `A` and `c`;
4. Get `wb`, the weight and intercept data, by calling `quadprog()`;
5. Create a plot that displays the data and the separating line. You will need to work out the coordinates (px(1),py(1)) and (px(2),py(2)) needed to display the line. Hint: px(1) and px(2) can be the minimum and maximum rpm value, and then py(1) and py(2) can be found using your SVM separating line formula rpm * `w(1) + vib * w(2) + b = 0`:

```
good = find ( grade == +1 );
bad = find ( grade == - 1 );
px = [ min(rpm), max(rpm) ];
py = [ ?, ? ];
hold on
plot ( rpm(good), vib(good), 'b+' );
plot ( rpm(bad), vib(bad), 'ro' );
plot ( px, py, 'k-' );
hold off
```

Include this plot in your submission!

Your plot should show that the SVM has "optimally" separated the data.

Python users: The Python package `cvxopt` can solve quadratic programs. It can be downloaded by `pip install cvxopt`. Your program will need to include the `import` statements:

```
from cvxopt import matrix
from cvxopt import solvers
```

`cvxopt()` assumes a general quadratic problem of the form:

$$\text{minimize: } \frac{1}{2}x'Hx - f'x \text{ subject to: } Ax \leq c$$

which (aside from the minus sign on the `f`) is the same as we saw for MATLAB.

Once you have set up the `H`, `f`, `A` and `c` arrays, convert them each to the `matrix` format, as in:

```
H = matrix ( H )
```

Then call solvers.qp:

```
sol = solvers.qp ( H, f, A, c )
```

and then convert `sol` back to a `numpy` array:

```
wb = np.array ( sol['x'] )
```

Then plot the data and the SVM separating line, as suggested above.

If you have trouble with `cvxopt()`, please let me know!

```
Table 1  Support Vector Machine
-------------------------------------------------------------------------
f(x) = x' w + b = rpm * ___  + vib * ___  + ___
```