MATH 728D: Machine Learning
Project 3: Support Vector Machine, Multivariate Regression, Clustering

---

1 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:

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

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:

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

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

$$\begin{aligned} x \rightarrow wb(1:d+1,1) &\rightarrow \begin{pmatrix} w \\ b \end{pmatrix} \\ H(1:d+1,1:d+1) &\rightarrow \frac{1}{2}\begin{pmatrix} I_d & 0 \\ 0 & 0 \end{pmatrix} \\ f(1:d+1,1) &\rightarrow \begin{pmatrix} 0_{d+1} \end{pmatrix} \\ A(1:n,1:d+1) &\rightarrow -\operatorname{diag}(Y) * \begin{pmatrix} X & 1_n \end{pmatrix} \\ c(1:n,1) &\rightarrow \begin{pmatrix} -1_n \end{pmatrix} \end{aligned}$$

Here, I_m 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 * ___  + ___
```

2 Multivariate Regression

The file *insurance_train.txt* includes $n = 1000$ records about medical insurance. Each record lists, for a given person, 7 factors: age, sex ("female"=0, "male"=1), BMI, children, smoker ("no"=0, "yes"=1), region ("NE"=1, "NW"=2, "SW"=3, "SE"=4), total medical charges ($). We seek an affine formula that can estimate the charges based on the numerical values of age, sex, BMI, number of children, and smoker:

```
charges = c1 + c2 * age + c3 * sex + c4 * bmi + c5 * children + c6 * smoker
```

Because our techniques work for linear models, not affine ones, we make up an extra variable called "one" so that our problem has the form:

```
charges = c1 * one + c2 * age + c3 * sex + c4 * bmi + c5 * children + c6 * smoker
```

Our array A will have **six** columns, with the first being a vector of 1000 1's, followed by columns for age, sex, BMI, children, smoker. We seek values c that minimize the root mean square norm of $y - A * c$.

1. Read the training data from *insurance_train.txt*, extract columns 1, 2, 3, 4, 5, and 7, storing them in vectors named age, sex, bmi, children, smoker, and charges;
2. Build a matrix `A`, starting with a column of 1000 1's, then columns for age, sex, bmi, children, smoker;
3. Compute and tabulate the coefficients `c` that minimize the norm of the residual y-A*c;
4. Tabulate the rms norm of $y - A * c$, comparing your predicted and actual charges for the training data;
5. Tabulate the actual and estimated charges for patients 200, 400, 600, 800 and 1000;
6. Read the testing data from *insurance_test.txt*;
7. Create the appropriate matrix `A` for your new set of data, and use the values `c` that you computed from the training data, to estimate the charges for the testing data; Report the rms norm of the residual when estimating the testing data;

```
Table 2  Multivariate regression
-------------------------------------------------------------------------
charges = ___ * one + ___ * age + ___ * sex + ___ * bmi + ___ * children + ___ * smoker
RMS of residual for training data:
rms = ___
Estimated and actual charges for some patients:
 #200   ___  ___
 #400   ___  ___
 #600   ___  ___
 #800   ___  ___
#1000   ___  ___
RMS of residual for testing data:
 rms = ___
```

If you compare the testing and training rms values, what do you observe? Notice that one of the regression coefficients is negative. What does this suggest?

3 Clustering

Clustering is used if we believe our n data values fall naturally into k separate groups, or *clusters*. To classify the data, we want to assign a cluster index `cluster(i)` to each data item `data(i,:)`. Assuming the clusters are roughly circular, then one way to do this is to pick a representative central point `center(j,:)` for each

cluster and assign data items to the cluster with the nearest center. Mathematically, we imagine the vector `cluster` reports the assignment of each data item:

$$cluster(i) = \arg\min_{1 \le j \le k} (data(i,:) - center(j,:))^2$$

In MATLAB, it may be more convenient to list all the data items that belong to a given cluster. Thus, the indices of the elements in cluster 1 might be listed in a vector `c1`, whose length also tells us how many elements are in cluster 1. For a two cluster problem, with `d1` and `d2` the distances of each data item to the two cluster centers, we could compute `c1` in MATLAB by:

```
c1 = find ( d1 < d2 );
```

Given the centers, we can compute `E`, the cluster "energy" or cluster variance:

$$E = \sum_{i=1}^{n} (data(i,:) - center(cluster(i),:))^2$$

A good clustering has a low value of `E` and an optimal clustering minimizes it.

In MATLAB, we might compute the individual cluster energies, as in:

```
e1 = sum ( d1(c).^2 );
```

and in our two-dimensional case, we can then easily compute `E=e1+e2`.

Notice that we are replacing n data values by k representatives, the cluster centers, so in some sense clustering is a kind of dimension reduction. In this case, we are not reducing the spatial dimension d, but the dimension n counting the number of data values.

Once we have the optimal cluster assignment, we can usually get better results by recomputing the centers as the average of the points belonging to them. By changing the centers, we might actually tempt some data points to switch clusters, and so the process of updating cluster assignments and centers can involve an iteration.

We will begin with some simple clustering efforts of our own, and then turn to the k-means algorithm, which can return the optimal cluster centers and cluster assignments by a single function call. Keep in mind that our measure of optimality is a small total cluster energy, `E`.

We will work with a data file of $n = 272$ pairs of values (d=2), representing the length of an eruption of the old Faithful geyser, and the length of the pause that follows.

1. Get the file *faithful.txt* and load the data;
2. Standardize your data, so that both columns range between 0 and 1;
3. A scatterplot of your data will show it naturally divides into $k = 2$ clusters;
4. Define a pair of centers:

```
center = [ 0.40, 0.60;     <-- row 1 is center #1
           0.60, 0.30 ];           2 is center #2
```

5. Define a vector `d1`, the distance of each data item to cluster center 1; similarly define `d2`;
6. Define a vector `c1` which indexes data points closer to center 1 than to center 2; similarly define `c2`;
7. A scatterplot would show how you have clustered the data:

```
hold on
plot ( data(c1,1), data(c1,2), 'b.' );
plot ( data(c1,1), data(c1,2), 'r.' );
```

```
plot ( center(1,1), center(1,2), 'bo' );
plot ( center(2,1), center(2,2), 'ro' );
hold off
```

8. Compute your cluster energies `e1`, `e2`, and `E=e1+e2 and report these energies in the table; e1` is the sum of the squares of `d1` for the data indexed in `c1`; `e2` is computed similarly;
9. Replace each cluster center by the average of its data elements, and report these centers in the table.
10. Using these new centers, update `d1` and `d2`, `c1` and `c2`, and the cluster energies `e1`, `e2`, and `E=e1+e2`, and report these energies in the table.
11. If we repeat these steps, the total energy will decrease, and the clustering may improve significantly. Instead, call `kmeans()` (or, in Python, you could use `scipy.cluster.vq.kmeans()`) to get the best clustering centers in one step:

    ```
    k = 2;
    [ c, center ] = kmeans ( data, k );
    c1 = c(1);
    c2 = c(2);
    ```

    Compute the energies `e1`, `e2`, and `E=e1+e2`; Report the kmeans centers and energies in the table.

```
Table 3  Clustering
-------------------------------------------------------------------------
Clusters using center1 = (0.4,0.6) and center2 = (0.6,0.3):
  e1 = ___  e2 = ___  E = ___
Clusters using averaged centers = (___, ___) and center2 = (___, ___):
  e1 = ___  e2 = ___  E = ___
Clusters using kmeans centers = (___, ___) and center2 = (___, ___):
  e1 = ___  e2 = ___  E = ___
```

Plotting the data suggested that it formed two clusters. Performing the kmeans clustering gives optimal representatives of these two clusters.

4 What to Turn in:

The project should be completed and submitted by the beginning of class on April 25th. You should submit a report, roughly 3-4 pages in length, stored as a single PDF, that includes:

1. **Team information:** The names of the team members. Teams can involve from 1 to 4 members. Each team member should participate in the project, and should be able to explain the process involved in getting the results of the exercises.
2. **Problem Description:** For each exercise, make any comments that come to mind about the problem, the programming, and the results.
3. **Tables:** You should turn in neatly formatted versions of Tables 1, 2, and 3.
4. **Plots:** Your PDF should include the plot associated with the SVM calculation.
5. **Send report to:** *burkardt@mailbox.sc.edu*

Keep copies of the programs used to get your results, as they may be requested for review if there is an issue with your results.