

# ME4 Machine Learning - Tutorial 4

In lecture 4 we looked at linear discriminant functions, as well as a section on scaling at the end. In this tutorial you will use Python to test out what linear discriminant functions look like, including the generalised form. Linear discriminant functions are underlying tools for a number of other applications in machine learning, so what we will be doing will be a bit more abstract than other tutorials, with no practical examples using them on actual data. This tutorial should help you to understand the concepts. However, initially, we will do a quick demo of scaling.

## 1 Scaling

Take the dataset [http://pogo.software/me4ml/tensile\\_strength.csv](http://pogo.software/me4ml/tensile_strength.csv). This contains various temperature values and various corresponding measured strengths, under the headings ‘Temperature (deg C)’ and ‘Ultimate tensile strength (Pa)’. Load this in using Pandas. As discussed in the lecture, calculate the means and standard deviations of each of the three parameters. You can use the functions `np.mean()` and `np.std()` for this - call your variables `t_mean` and `s_mean` for means of temperature and strength respectively, and use `t_std` and `s_std` for the standard deviations. Use these to scale the parameters. Then plot a histogram to check whether the scaling has worked as you expect; assuming that your scaled velocity is in the numpy array `s_scale`:

```
fig , ax = plt.subplots()
plt.hist(s_scale)
plt.show()
```

Do these match what you would expect? Try plotting a histogram of the original too and see what it looks like.

In some cases, you may want to generate these scaling values from the training dataset then apply them to a test dataset in a different script. Therefore you need to save the scaling parameters. Try saving these with the code:

```
scArray = np.array([[t_mean, s_mean],[t_std, s_std]])
np.savetxt('scaleParams.txt',scArray)
```

*#download it from the Colab interface:*

```
from google.colab import files
files.download('scaleParams.txt')
```

Also, check that you can load this in with the following code, and that the numbers make sense:

```
loadedScales = np.loadtxt('scaleParams.txt')
```

This capability may be useful when doing the coursework. You can achieve this scaling in other ways too. Note that scikit-learn also includes tools for scaling you can use if you wish, rather than calculating manually.

## 2 Plotting linear discriminant functions

Following the definitions in the lecture, I now define a discriminant function in 2D space  $g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$  using  $\mathbf{w} = (-1, -3)^t$  and  $w_0 = 1$ . Plot this in a region  $0 < x_1 < 1$ ,  $0 < x_2 < 1$ . (Note that you will need to use meshgrid and contourf as in tutorial 1, and you may wish to add a colour bar with plt.colorbar()). Note that if you have lots of  $\mathbf{x}$  values you need to apply the same  $\mathbf{w}$  to then you can do this in a matrix multiplication. Assuming  $\mathbf{w}$  is size  $2 \times 1$  and  $\mathbf{X}$ , combining all the  $\mathbf{x}$  coordinates together, is size  $2 \times n$  for  $n$  coordinates, you can do a matrix multiplication:

$$\mathbf{g} = \mathbf{w}^t \mathbf{X} + w_0$$

where  $\mathbf{g}$  will now be a  $1 \times n$  vector. Numpy has the function np.matmul() for multiplying two matrices together.

Calculate the line which corresponds to the discrimination line  $g(\mathbf{x}) = 0$  and plot this on the same graph. Make sure the limits are set to the same size as the region  $0 < x_1 < 1$ ,  $0 < x_2 < 1$ .

## 3 Plotting classification areas

We define five linear discriminant functions. Defining a generalised function as  $g = \mathbf{a}^t \mathbf{y}$  where  $\mathbf{y} = (1, x_1, x_2)^t$  with  $\mathbf{a}$  being the corresponding weightings, as given in the lecture, we have the following weighting terms:  $\mathbf{a}^{(1)} = (1.3, -1, -3)^t$ ,  $\mathbf{a}^{(2)} = (-2, 1, 2)^t$ ,  $\mathbf{a}^{(3)} = (0.3, 0.1, -0.1)^t$ ,  $\mathbf{a}^{(4)} = (0, -1, 1)^t$  and  $\mathbf{a}^{(5)} = (-0.2, 1.5, -1)^t$ . Following the definition of a linear machine, we classify any point according to which corresponding function  $g$  is the largest at that point. Produce a 2D map showing the classification regions corresponding to these five functions across the region  $0 < x_1 < 1$ ,  $0 < x_2 < 1$ . Assuming you have defined a sampling grid (Xgrid) in an array of size (npx\*ncpy, 2) where npx and nply are the dimensions in x and y of the grid, and your vectors for each  $\mathbf{a}$  term defined as having shape (1, 3), you can use the following code to generate the classification values:

```
#Ygrid is defined as the same as Xgrid, except it has 1
#at the beginning - this therefore adds a column of ones to the left
Ygrid = np.concatenate([np.ones([npx * nply, 1]), Xgrid], axis=1)

#calculate each of the five functions as before
g1 = np.matmul(a1, Ygrid.T)
g2 = np.matmul(a2, Ygrid.T)
g3 = np.matmul(a3, Ygrid.T)
g4 = np.matmul(a4, Ygrid.T)
```

```

g5 = np.matmul(a5, Ygrid.T)
#combine all five functions together
gconc = np.concatenate([g1, g2, g3, g4, g5])

#find which of the columns is the largest
omega=np.argmax(gconc, axis=0)

#put back onto 2D grid so it can easily be plotted
omega = np.reshape(omega, [npx, npy])

```

### 3.1 Higher order functions

Now we update the values to  $\mathbf{y} = (1, x_1, x_2, x_1x_2)^t$ , with weightings  $a^{(1)} = (1.3, -1, -3, -10)^t$ ,  $a^{(2)} = (-1, 1.5, 3, -1)^t$ ,  $a^{(3)} = (0.4, -0.1, -0.1, 3)^t$ ,  $a^{(4)} = (0.5, -1, 1, -0.1)^t$  and  $a^{(5)} = (-0.2, 1.5, -1, 0.4)^t$ . The  $x_1x_2$  term will include some nonlinearity in the decision boundaries. Plot the values on a 2D map.

## 4 Conclusions

This tutorial has demonstrated linear discriminant functions and aimed to give you an appreciation for the way they behave. The examples we have looked at are fairly abstract, with us focusing on the mathematical and computational aspects of their use rather than applying them to any specific application or data. This is really a reflection on how they are more fundamental tools which are useful to understand and upon which other methods – SVMs and neural networks in particular – are based.