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. 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 strength 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 $g(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + w_0$ using $\boldsymbol{w} = (-1, -3)^t$ 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 $\boldsymbol{x}$ values you need to apply the same $\boldsymbol{w}$ to then you can do this in a matrix multiplication. Assuming $\boldsymbol{w}$ is size 2×1 and $\boldsymbol{X}$, combing all the $\boldsymbol{x}$ coordinates together as rows, is size $m \times 2$ for m data points, you can do a matrix multiplication:

$$\boldsymbol{g} = (\boldsymbol{w}^t\boldsymbol{X}^t)^t + w_0$$

where $\boldsymbol{g}$ will now be a $m \times 1$ vector. Note that $\boldsymbol{w}$ is transpose as per the definition of the function, $\boldsymbol{X}$ is transpose to turn each row into a column, corresponding to each $\boldsymbol{x}$ which is a column, and we have transposed the final result to ensure that the first dimension of g loops through the m samples. This can also be written as $\boldsymbol{g} = \boldsymbol{X}\boldsymbol{w} + w_0$, which is mathematically identical and simpler, but is not quite as clear. Hint: you are advised to use print(w.shape) on $\boldsymbol{w}$, $\boldsymbol{X}$ and g to check that the dimensions are correct as described above. Numpy has the function np.matmul() for multiplying two matrices together.

Calculate the line which corresponds to the discrimination line $g(\boldsymbol{x}) = 0$ and plot this on the same graph – you should do this by hand and put the equation into Python rather than trying to use the code to perform the calculation. Make sure the limits are set to the same size as the region $0 < x_1 < 1$, $0 < x_2 < 1$.

We plot the function and the discriminant function together like this purely for illustration purposes, to help you understand what the function in the background looks like. You should have a sense of the form of the decision function and how it is produced, although in practice it is rare that you would actually plot the whole function like this. This is just what it looks like in the background.

3 Plotting classification areas

We define five linear discriminant functions. Defining a generalised function as $g = \boldsymbol{a}^t\boldsymbol{y}$ where $\boldsymbol{y} = (1, x_1, x_2)^t$ with $\boldsymbol{a}$ being the corresponding weightings, as given in the lecture, we have the following weighting terms: $a^{(1)} = (1.3, -1, -3)^t$, $a^{(2)} = (-2, 1, 2)^t$, $a^{(3)} = (0.3, 0.1, -0.1)^t$, $a^{(4)} = (0, -1, 1)^t$ and $a^{(5)} = (-0.2, 1.5, -1)^t$. 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*npy, 2) where npx and npy are the

dimensions in x and y of the grid, and your vectors for each $\boldsymbol{a}$ term defined as having shape (3, 1), you can use the following code to generate the classification values.

```
import sys
if a1.shape != (3, 1):
        print("Error!! Shape of a1 is incorrect.")
        sys.exit()
if Xgrid.shape != (npx*npy, 2):
        print("Error!! Shape of Xgrid is incorrect.")
        sys.exit()

#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 * npy,1]), Xgrid],axis=1)

#calculate each of the five functions as before
g1 = np.matmul(Ygrid,a1)
g2 = np.matmul(Ygrid,a2)
g3 = np.matmul(Ygrid,a3)
g4 = np.matmul(Ygrid,a4)
g5 = np.matmul(Ygrid,a5)

#combine all five functions together
gconc = np.concatenate([g1, g2, g3, g4, g5],axis=1)

#find which of the values is largest for each row
omega=np.argmax(gconc,axis=1)

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

This space represents 5 regions which classify any points which lie within them. You would train your linear discriminant functions such that these regions fit the training dataset, but here we are just illustrating the forms that they can take.

3.1 Higher order functions

Now we update the values to $\boldsymbol{y} = (1, x_1, x_2, x_1x_2)^t$, 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.