Machine Learning Unit

# What is machine learning?

Well it’s basically where we get a machine to observe examples of a task and generate its own set of rules that fit with those examples and allow it to do the task by itself.

For example, we could give the machine some data classification examples and then get the machine to generate a set of rules to classify that data.

Some data classification examples in the form of a table:

|  |  |  |
| --- | --- | --- |
| Feature: height in cm (input) | Feature: mass in kg (input) | Classification (output) |
| 60cm | 30kg | Child |
| 144cm | 45kg | Child |
| 150cm | 60kg | Child |
| 165cm | 65kg | Adult |
| 160cm | 58kg | Adult |
| 178cm | 65kg | Adult |
| 187cm | 95kg | Adult |

Machine learning algorithms ccould take this training data and find a rule to predict Adult or Child when given a brand-new **never previously seen** data example containing height and mass.

# Questions

Circle the most correct answer.

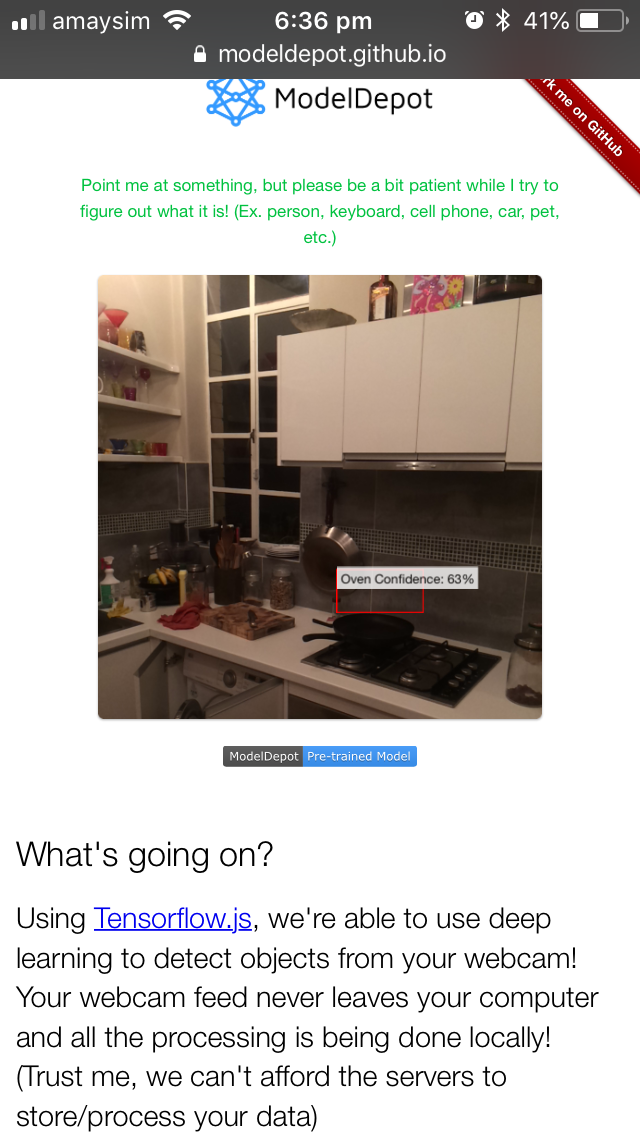
In machine learning, “Features” are:

1. Films that are special.
2. Outputs of the machine learning algorithm that classify data.
3. The answers to a machine learning problem.
4. Input data that describes the item being classified.

# What can machine learning do?

What can’t it do? Every day machine learning is being used to do something that was previously unable to be done by machines.

It’s not just limited to tables of data. Machines can learn to classify items in camera images:



Note that this JavaScript version of Tensorflow running in real time on an iPhone 5S has not quite hit the nail on the head when looking at my kitchen, but it’s impressive nonetheless, given it had no knowledge of where it was. Almost freaky.

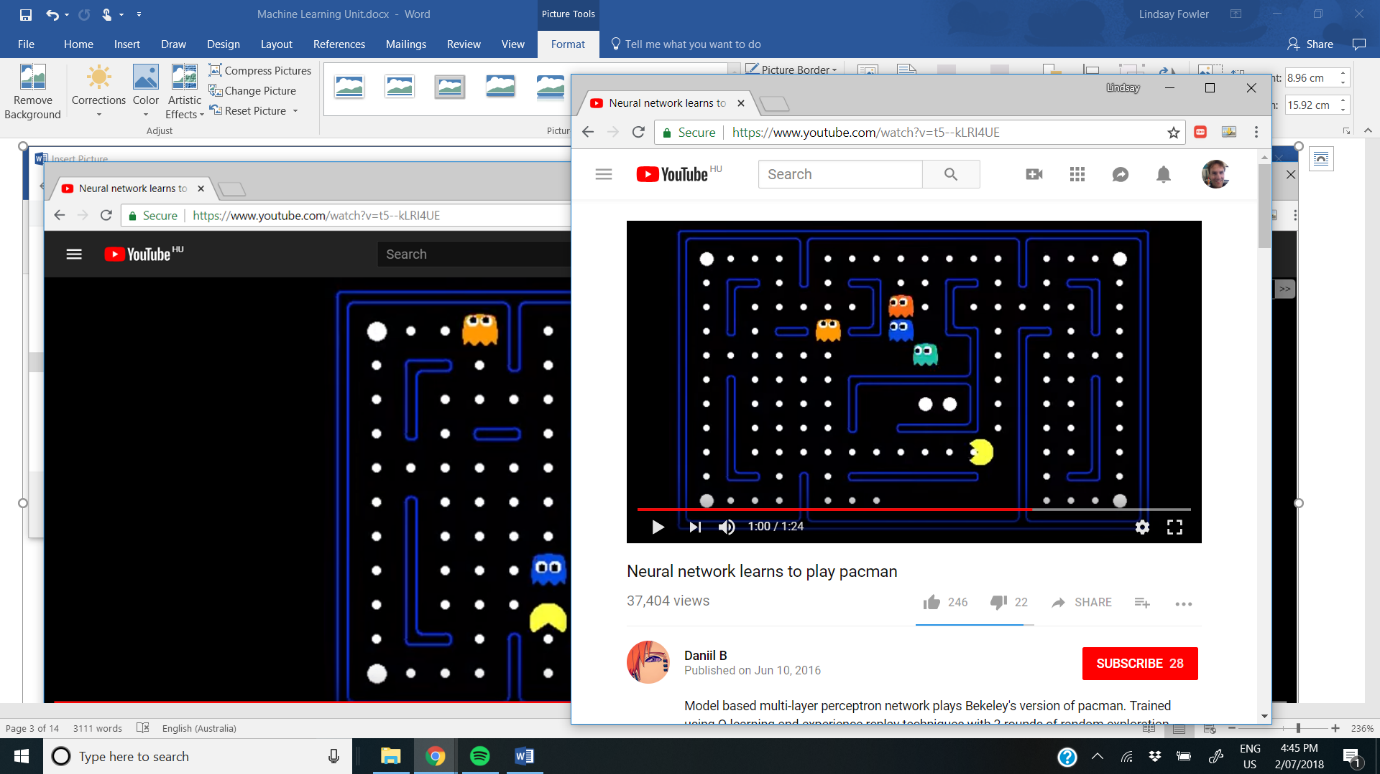
[](https://www.youtube.com/watch?v=W_gxLKSsSIE)

Machine learning can even observe examples of actions and learn to perform tasks such as flipping a pancake:

<https://www.youtube.com/watch?v=W_gxLKSsSIE>

Machine learning algorithms can be used to play video games and pull off some impressive moves, challenging for humans to replicate. Machine learning bots can often achieve higher scores than any human.

<https://youtu.be/t5--kLRI4UE?t=1m>

[](https://youtu.be/t5--kLRI4UE?t=1m)

Types of Machine Learning

There are two main types:

* Supervised Learning – where we tell the machine the answers and tell it to learn from them
* Reinforcement Learning – where the machine can act in a random way and learn when it achieves success (read Pong from Pixels for more info) – this works really well as long as the feedback is instant for not too many actions. <https://www.youtube.com/watch?v=JgvyzIkgxF0>

# Easy example with one feature

Here is some data which actually has two features (inputs) and one classification (output).

You should be able to see that the student’s name has nothing to do with making a decision about whether they passed or failed the examination – so really, this is an example with one input – we will ignore the Student Name feature in the data.

|  |  |  |
| --- | --- | --- |
| Feature: Student Name | Feature: Student Exam Score | Classification: |
| Freddy | 109 | Pass |
| Amanda | 53 | Fail |
| Gemma | 90 | Pass |
| Lindsay | 85 | Pass |
| Jim | 61 | Pass |
| Austin | 51 | Fail |
| Maurice | 87 | Pass |
| Jelena | 107 | Pass |
| Lucille | 90 | Pass |

We don’t know the total number of the marks available in the examination, but we could probably take a reasonable guess by looking at the data. We can see that scores of 53 or less were a fail and scores of 61 or more were a pass.

Let’s choose the number halfway between as the passing score: 57

Let’s write some pseudo code for a rule.

If StudentExamScore > = 57:

classify as Pass

Else:

classify as Fail

That’s a pretty good rule, even if the examination passing grade was actually 55 – our rule would still work really well most of the time. 😊

Now you try:

# Easy Activity – Make a rule for the data

Can you determine simple way of predicting whether someone is an adult or child, based on this data?

|  |  |
| --- | --- |
| Feature: year of birth (input) | Classification (output) |
| 2015 | Child |
| 2014 | Child |
| 2010 | Child |
| 1999 | Adult |
| 1990 | Adult |
| 1993 | Adult |
| 1977 | Adult |

Rule for being an adult:

|  |
| --- |
|  |

Will this rule always work?

If no, then explain when it wouldn’t work.

|  |
| --- |
|  |
|  |

# Slightly Harder Activity – Make a rule for this data with two features.

See if you can determine a simple way of predicting whether someone is an adult or child, based on this data which incorporates two features: height and weight?

|  |  |  |
| --- | --- | --- |
| Feature: height in cm (input) | Feature: mass in kg (input) | Classification (output) |
| 60cm | 30kg | Child |
| 144cm | 45kg | Child |
| 150cm | 60kg | Child |
| 165cm | 65kg | Adult |
| 160cm | 58kg | Adult |
| 178cm | 65kg | Adult |
| 187cm | 95kg | Adult |

Rule for being an adult \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Rule for being a Child \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Will these rules always work?

# How does machine learning solve problems?

At a very basic and general level, machine learning works by applying a rule and then seeing if it works. If the rule doesn’t work, it changes the rule a little bit and tries it again. It keeps doing this, thousands and thousands of times in the hope of getting a rule that is reliable.

For example, looking at the data in the set above a machine might try a rule like **“if the mass is more than 40kg it’s an adult**”. It would then apply this rule to all the rows in the table and see how often it got it right (green) and wrong (red):

**if the mass is more than 40kg it’s an adult**

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs or features | | Output | |
| height in cm | mass in kg | Target (what it should be) | Output using rule |
| 60cm | 30kg | Child | Child |
| 144cm | 45kg | Child | Adult |
| 150cm | 60kg | Child | Adult |
| 165cm | 65kg | Adult | Adult |
| 160cm | 58kg | Adult | Adult |
| 178cm | 65kg | Adult | Adult |
| 187cm | 95kg | Adult | Adult |

Hmm two wrong – not bad – let’s adjust the rule a bit:

**if the mass is more than 50kg it’s an adult**

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs or features | | Output | |
| height in cm | mass in kg | Target (actual data) | Output using rule |
| 60cm | 30kg | Child | Child |
| 144cm | 45kg | Child | Child |
| 150cm | 60kg | Child | Adult |
| 165cm | 65kg | Adult | Adult |
| 160cm | 58kg | Adult | Adult |
| 178cm | 65kg | Adult | Adult |
| 187cm | 95kg | Adult | Adult |

Improvement! Only one wrong! That’s excellent.

Let’s make a rule for height too:

**If the height is over 170cm it’s likely to be adult**

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs or features | | Output | |
| height in cm | mass in kg | Target (actual data) | Output using rule |
| 60cm | 30kg | Child | Child |
| 144cm | 45kg | Child | Child |
| 150cm | 60kg | Child | Child |
| 165cm | 65kg | Adult | Adult |
| 160cm | 58kg | Adult | Child |
| 178cm | 65kg | Adult | Adult |
| 187cm | 95kg | Adult | Adult |

Again we have one wrong classification – but it’s a different one this time…

Now, how do we use both our rules together to get a better result? Enter the neural network…

Let’s give each rule a strength. Like 60% of the outcome depends on the body-mass rule and 40% of the outcome depends on the height rule.

Let’s try it:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| inputs | | Ouputs |  |  |  |  |
| height | mass | Actual | >50kg (60%) | >170cm (40%) | Combined Score | Confidence of being adult |
| 60cm | 30kg | Child | 0 | 0 | 0 | 0% |
| 144cm | 45kg | Child | 0 | 0 | 0 | 0% |
| 150cm | 60kg | Child | 60 | 0 | 60 | 60% |
| 165cm | 65kg | Adult | 60 | 0 | 60 | 60% |
| 160cm | 58kg | Adult | 60 | 0 | 60 | 60% |
| 178cm | 65kg | Adult | 60 | 40 | 100 | 100% |
| 187cm | 95kg | Adult | 60 | 40 | 100 | 100% |

We still have one wrong, but we have the advantage of a confidence; the one that is wrong only had a 60% confidence so we knew the decision wasn’t so sure.

This simple combination of two rules could classify adults and children with a lowish error rate.

Obviously, this was a very simple example, just to get a gist of how machine learning is approached.

As the data we collect improves, and as we use better rules (not just greater than some threshold value), and as we adjust our rules many, many times, we can make machines that learn much more reliably.

# Neural Networks

A neural network is a collection of neurons connected together, like we connected the two rules above. Here is the same example drawn as a neural net. (Real neural nets will have many more neurons)

Mass

0.4

0.6

Height

Output Value



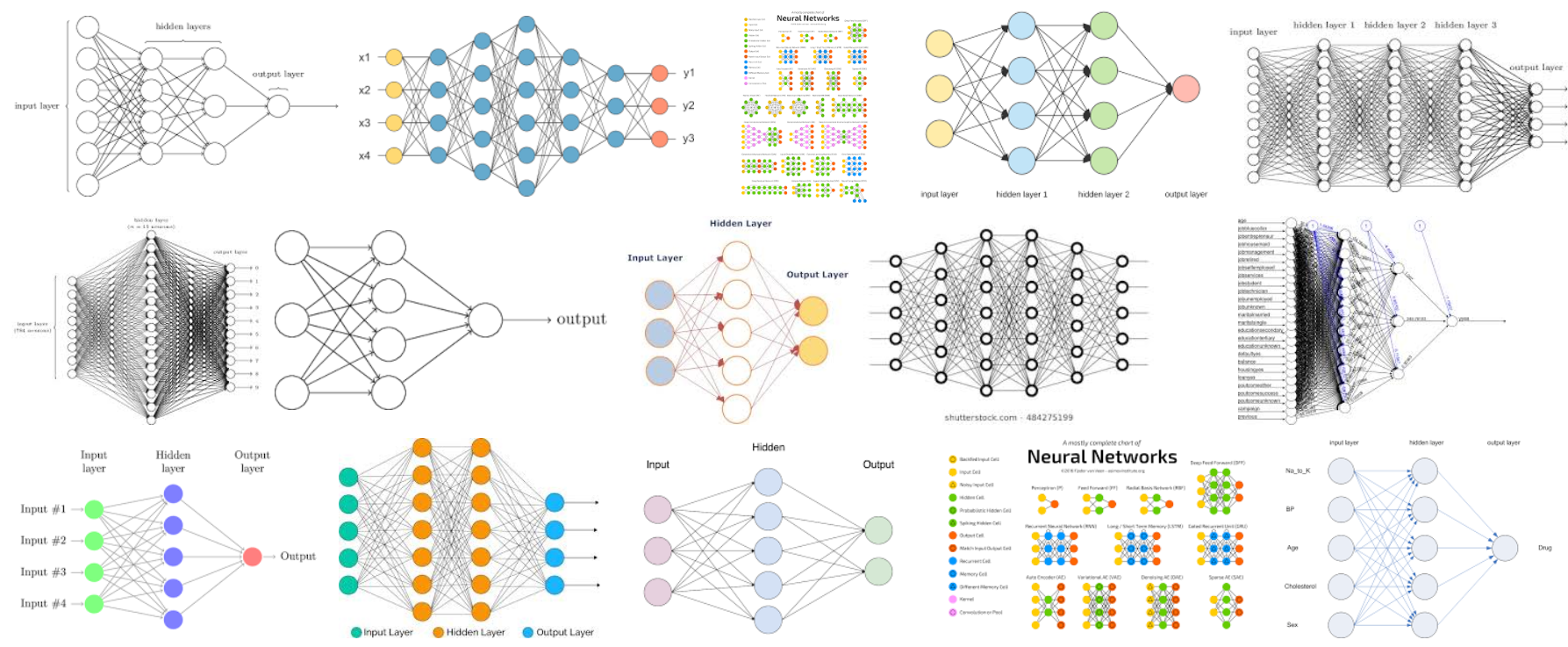
>170cm

>50Kg



The circles are called nodes or neurons and the arrows are called synapses. They are named after the bits and pieces that make up your brain. Notice the strength of influence (0.4 and 0.6) on the synapses. This means that the final node listens more to the value given by the weight node than the height node.

Let’s take a sneak peek at real neural nets by hitting Google Images:



You will notice that they have layers of nodes going from left to right. Generally, the left side is where the features (inputs) are detected and the right nodes are the outputs.

The middle layers are often called hidden layers because they are internal and don’t connect directly to inputs or outputs. You might also notice that every hidden node connects to every node in the previous layer and the next layer. The strength of these synapses (or perhaps better expressed: the level of influence of these connections) is adjusted iteratively as the network learns.

# Coding Up Some Machine Learning Examples

A lot of machine learning work has been done in Python and there are some libraries that make life a lot easier for getting started.

We will be using Python 3.6

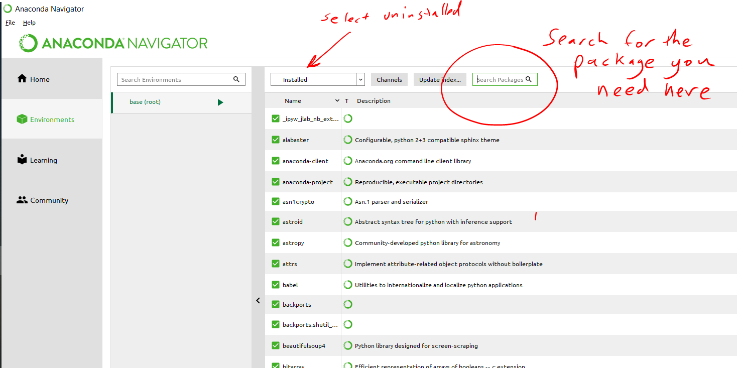
## Install Anaconda

Anaconda is probably the easiest way to install Python and provides the Spyder IDE which is probably the nicest development environment for Python that I have come across to date. It is still supremely daggy as far as IDEs go, what when compared to Visual Studio or X Code, but it’s the best we’ve got so I recommend using it. Install it now from <https://anaconda.org/anaconda/python>

## When it doesn’t work:

That’s right, expect it won’t work. Am I a negative Nancy? Well no, but whenever I teach any coding, half the class has errors and you need to expect to have things go wrong or you will give up at the first hurdle, so here’s some basic strategies to start you off and that you should come back to:

### Gotchas

1. Make sure you’re not using Python 2.7 or something too old.
2. Errors have line numbers to help you find them – but if it says line 13, the error could actually be on the previous line or the next line, so be aware.
3. Spelling errors – number one source of errors – check how you spelled everything.
4. Python code is case-sensitive, meaning import is different to Import.
5. Make sure you include all the brackets; each opening bracket ( should have a closing bracket )
6. Make sure all your quotes “ have closing quotes “
7. Python is extremely particular about the number of spaces you use to indent a line. Always use the tab key to indent a line 4 spaces at a time and you shouldn’t have a problem where a single space character ruins your code.
8. If it says “module does not exist” or something similar, it probably means you need to install the module. You can do this in Anaconda under Environments.

Or if you prefer using IDLE you will need to go to the command line and install modules with “pip”.

# Coding Example 01

Let’s predict the price of a car based on the brand, the year, the kilometres!

# Questions

Circle the most correct answer.

Machine learning is where:

1. The computer programmer teaches the machine something.
2. The computer programmer has to write a new program for each new task.
3. The computer programmer adjusts the weights of the neurons carefully by hand so that the computer can do a task.
4. The computer can use a standard program for a new task and reconfigure it by using an iterative process to adjust certain numbers that govern how it passes information from neuron to neuron.

Supervised learning is the branch of machine learning where:

1. The computer can be left unattended because it supervises itself.
2. The computer generates its own training data and a human supervises it.
3. The computer learns by looking at unlabelled training data.
4. The computer learns by looking at training data which has the outputs correctly labelled beforehand, so it effectively has examples to mimic.

How does a computer figure out if it is on the right track or not?

What are the columns of input data called?

1. Tables
2. Arrays
3. Files
4. Features

# Getting Datasets

Supposedly, the Australian Bureau of Stats is the place to go. I tried using their Table Builder but I found it super slow and was unable to come up with anything useful for machine learning.

American universities have some much better resources specifically for machine learning.

Here is a great archive from University of California Irvine with over 400 interesting datasets:

<https://archive.ics.uci.edu/ml/datasets.html>

Also try Stanford Machine Learning

MIT, Berkerley

Stanford’s Car dataset: <http://ai.stanford.edu/~jkrause/cars/car_dataset.html>

## Make your own

There are several options here:

1. You can collect your own data by conducting a survey and saving it as a CSV file!
2. You could ask your school or other organisation for some data (without student names).
3. You can also collect data electronically, if you’re good with and Arduino!
4. You could mine the data yourself using Python code that accesses a website!

# Tips for creating your own datasets

Try to ensure that you have roughly 10 times the data examples as you have features. So if you have 3 input features, you should have at least 30 examples in your dataset. Also try to get data examples that cover the range of examples you wish to test.

Feature Engineering: Choosing useful data. Adding irrelevant features to your dataset will make your model more complex and will more than likely make it less accurate.

Pre-process your dataset.

# Pre-processing

Binning – converting numerical values to true or false values

## Remove Duplicate Data

Weight in kg and weight in lb is the same info expressed in different units. This introduces added complexity without adding any value.

# Splitting Testing and Training Data

SKLearn’s train\_test\_split function allows you to split the input (x) data and labels (y) into a training set and a testing set. Import it like this:

from sklearn.model\_selection import train\_test\_split

use it like this:

(trainX, testX, trainY, testY) = train\_test\_split(data, labels,

    test\_size=0.25, random\_state=42)

The examples for each set are taken randomly; leaving the random state out will result in different values chosen each time; setting it to a number will result in the same random selection each time (useful for repeatability).

You can elect to make any percentage the testing set and the remaining will be used for the training set.

The output of the function is four lists: trainX, testX, trainY, testY.

One Hot Encoding – This is useful for categorical data and involves converting a feature column to several separate feature columns. E.g. A model to learn the re-sale value of cars might have a dataset with an RGB encoded colour column; this could be converted to separate columns: blue, black, red, yellow etc. While this might seem like it would make the dataset more complex, it actually removes irrelevant numerical info; a (0,0,195) blue car is unlikely to be any better than a (0,0,180) blue car when it comes to re-sale value – it’s more likely to be due to the time of day the phodto was taken.

SKLearn’s LabelBinarizer is used to do one hot encoding.

from sklearn.preprocessing import LabelBinarizer

If the contents of trainY is [“car”, “car”, “car”, “truck, “van”, “car”, etc…]

We call fit\_transform like so..

trainY = LabelBinarizer().fit\_transform(trainY)

..and trainY will now be [1 0 0], [1 0 0], [1 0 0], [0 1 0 ], [0 0 1], [1 0 0], etc..

note that each label is now a one hot encoded vector.

Caution:

fit\_transform looks at the data and tries to figure out how many categories there are and then make a vector for each category. It is unclear which order the categories will be in but I think it is alphabetical order. In my experience, fit\_transform will can refuse to work properly if you have a small amount of data <50 in any category or perhaps a real imbalance in categories, e.g. 100 cars and 3 trucks and 4 vans. But this kind of imbalance will never help you in Machine Learning anyway.

# Libraries

Numpy, Scikit learn, pandas (panel data (pages in a spreadsheet)) are the main machine learning libraries that we will use in Python.

We call a column of data a vector… wouldn’t it be a row?

SIMD – great speed increases for modern cpus <https://en.wikipedia.org/wiki/SIMD>

Read SciKitLearn documentation

Numpy takes advantage of SIMD features of CPUs and allows computation code to run much faster.

Vectorising code means not iterating through an array. Instead we apply an instruction to the whole “vector” or column. Again this is what Numpy is geared to do.

## Loss and Accuracy

Every time data gets fed into a neural network for classification, the network produces a probability for each class.

e.g.

In a network that takes in a face and determines if it is Mike Tyson, Cindy Crawford or someone else:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| One Hot Encoded Labels | | | Network Output Probabilities | | |  |  |
| Cindy Crawford | Mike Tyson | Other | Cindy Crawford | Mike Tyson | Other | Accuracy | Loss |
| 1 | 0 | 0 | 0.5 | 0.3 | 0.3 | 1 | 0.7884574 |

The accuracy is 1 or 100% because the highest score was the correct category.

But the loss is calculated in a special way and takes into account the confidence in each of the classes. If the correct class only beat the other classes by a narrow margin, then the loss is high.

## Using Keras to Calculate Loss

In the terminal window you can try calculating loss yourself:

(it’s a little messy and there are warnings of deprecated code)

Open Python by typing “python”:

PS D:\Documents\Dropbox\Machine Learning\ > python

>>> from keras import backend as K

>>> labels = K.variable([[1., 0., 0.]])

>>> prediction = K.variable([[.50, 0.3, 0.3]])

>>> loss = K.eval(K.categorical\_crossentropy(truth, prediction))

>>> print(loss)

[0.7884574]

<http://www.jussihuotari.com/2018/01/17/why-loss-and-accuracy-metrics-conflict/>

Image Classification

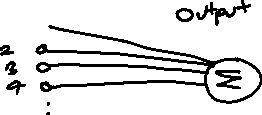
In image recognition, we use the image as data (before we just used a short list of numbers; an image is a very long list of numbers).

Each image is made up of pixels (coloured dots) and each pixel is made up of 3 numbers for Blue Green and Red light.

A tiny colour image that is 32 pixels by 32 pixels (about 8mm square on your screen) is a list of 32 x 32 x 3 numbers. For an 8 bit image (256 possible combinations), each of these numbers is between 0 and 255. These numbers get fed into the network inputs. So for this small image we will require a network with 3072 inputs!

You could start with a simple fully connected network: a support vector machine (SVM):

Of course, I haven’t drawn all 3072 input nodes connecting to the output.



This is an SVM, a single layer network. It multiplies every number in the image by a weight. There are 3072 pixels, so there are 3072 weights. The weights can be positive (if a large value for that pixel colour is desirable) or negative (if a large value for that pixel colour is not desirable).

As a simple example, we might find that in detecting pictures with cows, many of the surrounding pixels at the bottom would be usually be green, so the green inputs at the bottom will probably need positive weights. We might also give negative weights to red and blue pixels in the areas where grass might be expected.

Each multiplication is summed together to give a final score.

The bigger the score the more likely it is to be a picture of a cow!

You might think this is too simple for image classification and you would be correct, but it does work, albeit poorly and it would probably fail detecting a cow if the background was pink.

This model has one output, which makes it **binary.** The output can be high (above 0.5) or low (below 0.5). Models with multiple outputs are **categorical**.

Let’s give it a try

from keras.regularizers import l2

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(64, activation='relu'))

model.add(Dense(1), W\_regularizer=l2(0.01))

model.add(activation('linear'))

model.compile(loss='hinge',

optimizer='adadelta',

metrics=['accuracy'])

model.fit(X, Y\_labels)

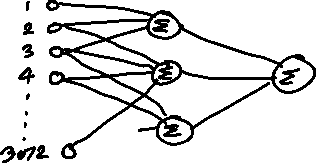
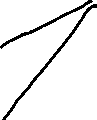
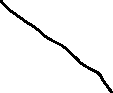
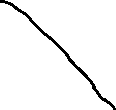
**Other important things to change/check in multiclass (compared to binary classification):**

Set class\_mode='categorical' in the generator() function(s).

Don't forget that the *last* dense layer must specify the number of labels (or classes):

model.add(layers.Dense(3, activation='softmax'))

By adding more layers to the network we can better results. For example, we might do something like this:



We then have more complex possibilities.

Using Google TensorFlow

<https://www.youtube.com/watch?v=Rgpfk6eYxJA>

# Install Tensor Flow GPU 1.4 Mark Jay

<https://www.youtube.com/watch?v=RplXYjxgZbw>

Without an Nvidia GPU you can simply install the CPU version.

Most promising free course to date. Suitable for Year 10s maybe 9s.

<https://developers.google.com/machine-learning/crash-course/ml-intro>

# A simple neural network with one neuron and one input



Using pyimage search – he wants $$$

These are my bing API keys

Key 1: 4a1f2d5713e44ceaa894b66eb0e8835b

Key 2: e8cc3137a2fd46f58c44c0a32749dcc8

On the raspberry pi:

pip install requests

pip install cv2

# Using Sentdex

This requires the use of python 2.7 which is a bit annoying.

Using Anaconda Navigator, I created a new environment, setting the Python version to 2.7.

It seems you then have to install spyder again… for that environment…

# Dataset Resources

Manythings.org has data sets with language translations

# Open AI gym

conda install -c akode gym