Regression Lab

January 2026

1 Objective

The objective of this lab is to gain hands-on experience with building machine learning models using scikit-learn, also known as sklearn. We will be building `Linear Regression Models` of some form throughout, and we will be assessing them too.

Assessment and interpreting results is just as, if not more, important than building the models. Having an understanding how each different type of model works will allow us to better interpret the results, evaluate the model and maybe improve the model in the future.

2 Data Structure

The models in sklearn and tensorflow typically expect data to be given to them all at once, that means your entire training dataset should be given altogether. Like the tuples of data I mentioned in class.

If our dataset has 5 features, we describe them with x_1, x_2, x_3, x_4, x_5, but every element of the dataset has its own features so then we need the $^{(i)}$ that I mentioned.

So, for one particular element of our dataset we have

1. The ground truth, the correct answer, which is typically labelled as $y^{(i)}$ for one particular element.
2. All the independent features, which is typically labelled as $\mathbf{x}^{(i)}$ for one particular element or $\left(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)}, x_5^{(i)}, \dots\right)^T$. This is a lot of subscripts and superscripts I know!

And we put put them altogether where

1. y has all the *correct* answers for our dataset. This will be a vector, a 1d array.
2. X which will have all the independent features for our dataset. This will be a matrix, a 2d array, where each row describes one particular element.

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \dots \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \dots \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(i)} & x_2^{(i)} & x_3^{(i)} & \dots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

   You might notice that each column describes one particular feature. Row is one element of the dataset, column is one feature - this shape will persist throughout all the models and understanding this will hopefully see why some bugs happen.

3. We pass all the data and features in as one matrix as this is much more efficient (the entire sample of data) than putting in one example at a time.

X and y should both be numpy arrays.

3 How to build a model

A lot of what we do today will translate to other types of models as sklearn (and tensorflow) use the same methods for building and evaluating models.

https://scikit-learn.org/stable/supervised_learning.html has a list of the supervised models that are available in sklearn.

- First thing to do is to import the model you want to use. Typically done like

```python
from sklearn.model_location import model_name
```

- Then we initialise the model, sort of like calling a constructor

```python
myModel = model_name(constructor_parameters)
```

- If we then have a matrix (2d array) X, as described above, and a vector (1d array) y , again as described above we can train our model with

```python
model_name.fit(X,y)
```

- And like magic a model will be built. The parameters will all be learnt at this phase and depending on the complexity of the model, and size of the dataset, this is the stage where it may take a long time.

3.1 Algorithm vs Model

I just want to write something here to make sure it is clear

- Typically we say the algorithm is the type of model that we are building e.g. Linear Regression
- A model is anything where we have learnt the parameters.
- Every time you call `model.fit(X,y)` you are building a *new* model.
- You can build multiple Linear Regression models by changing the constructor_parameters and calling `.fit`, each of these is their own model.

3.2 Linear Regression

Now more specifically linear regression

```python
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X, y)
```

For now at least, we don't need to set any `constructor_parameters` for the LinearRegression model.

Remember if you want more details in Jupyter notebook, write the method with ? after it e.g.

```python
LinearRegression?
```

4 Inferences/Making Predictions

We can see what else is in the *lr* object by tab completion

```python
lr.
```

and then pressing tab which gives us a list of things.

For a LinearRegression() object, the first interesting one to look at is `.coef_` (the _ is to indicate that this is an estimation, it is supposed to be like a $\hat{y}$ that I had on notes)

```python
lr.coef_
```

This will be all the weights in the order $(w_1, w_2, w_3, \ldots)$ (confusingly `lr.coef_[0]` is w_1 etc.)

But Donny, what about w_0? Well w_0 does not relate to a specific feature in our dataset, it is the weight for the 1, also known as the *intercept*. If you did `LinearRegression?` above, you will have seen

`fit_intercept=True`

as part of the LinearRegression initialiser. Anyway, w_0 is in its own field

```
lr.intercept_
```

So now we can write our model down fully since we know all the w_0 and the $w_1, w_2, \ldots$ so if we want to make a prediction (inference) we have the formula to do it

4.1 Use Model method to make inference

Other models are a lot more complicated even when we can read off the parameters easily and construct our own function. But, this requires more work and surely sklearn developers thought of this

`newX` needs to be an array of the same format as the one we used in the `.fit` stage. This will return an array of predictions. If we only want to make one prediction, then `newX` should only have one row. It is much faster to ask for multiple answers at the same time (due to numpy) rather than doing one inference at a time in say a loop.

5 Evaluation

We will do much more understanding of this in a later lecture and much better ways of doing it but I want you to see some of the methods now so let's go ahead with some naivety.
https://scikit-learn.org/stable/modules/model_evaluation.html has a list of a lot of metrics we could use for evaluation, now let's look at a couple for regression
The talky bit of the lab had me mention the Loss function as the mean-squared error. Well we can get this value quite easily, and maybe this could be some form of evaluation. (Closer to 0 the better?)
Another commonly used for regression is called R^2 (which despite it's name can be negative but I'll talk about that later), closer to 1 the better.
Root mean squared error is another good one - this is one with closer to 0 the better.

```
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score as r2
```

all of these take the same format, say `newY` is the ground truth for `newX`:

```
prednew = lr.predict(newX)
mse(newY, prednew)
```

will give us the mse score.

Unfortunately we don't have another set to try and evaluate, so let's just look at the set we used to build the model:

```
pred = lr.predict(X)
r2(y, pred)
```

and this will give us a score.

5.1 .score in model

Maybe you notice when pressing tab earlier that there is a method in the model called .score

```
lr.score()
```

This method is different for each model type. For LinearRegression it is the same as the r2 score and you can get it by

```
lr.score(newX, newY)
```

It does not save the in between step of what the inferences made actually are, so it's not suitable if you need these inferences later, but if you want a quick score, it's quite good. Again we can only use the sets we used to build

```
lr.score(X,y)
```

and we will get something. In the case of a LinearRegression type of model, `.score` is an R2 score.

6 Generalisation and Model Evaluation

The previous sections evaluated its performance using the same dataset. While this approach is useful for learning how models are fitted, it leads to overly optimistic performance estimates and does not reflect how models are used in practice.

In real-world applications, we are interested in how well a model *generalises* to unseen data. To estimate this, we must evaluate the model on data that was *not* used during training.

To achieve this, the available data is split into two parts:

- A **training set**, used to fit the model parameters.
- A **validation set**, used to estimate how well the model performs on unseen data.

The validation set is *not* used to fit the model and therefore provides a more realistic estimate of model performance.

6.1 A Larger Synthetic Dataset

Before working with real data, we will first explore a larger synthetic dataset designed to illustrate the effects of model complexity and feature selection on generalisation performance.

This dataset contains three input features:

- One feature that is strongly predictive of the response variable.
- One feature that is weakly predictive of the response variable.
- One feature that contains only noise and has no predictive value.

Using this dataset, we will:

- Split the data into training and validation sets.
- Fit linear regression models using different subsets of the available features.
- Compare models using both training performance and validation performance.

By comparing these results, we can observe that:

- Models with higher training performance do not necessarily generalise better.
- Adding additional features can lead to worse validation performance when those features do not contain useful information.

This process of comparing models using validation performance is known as *model selection*.