**Linear Regression**

1. What is linear regression in simple terms?
   1. Explain it to a 5 year old
   2. Explained to an adult
   3. Explained to a data scientist
2. What assumption are made when using linear regression?
3. What uses has linear regression had in the past?
4. Where does linear regression show the best results? Where is it most useful?
5. How do you know when to use linear regression?
6. Are there different types of linear regression?
7. When should I use linear regression?
8. What can I do if my linear regression isn’t immediately showing good results?

Linear Regression is a model that fits the best possible straight line through the data as a way to generalise the relationship between the output variable and one or more input variables. The overall difference between the straight line and the data points is minimised, which is why we sometimes call it a “line of best fit”. It is useful as it is an easy way to make predictions based upon the line of best fit. For a simple linear regression, you need a prediction of y from the x value, you simply go to the x value on the x axis, read upwards to the line, and then read the corresponding y value from the y axis to get your prediction. Simple! In reality, using code, you’d use a formula to calculate the linear regression line, and thus would just need to type in your independent variable, and it’d spit out a predicted dependent variable.

You can also do something called multiple linear regression, which takes multiple input variables, such as a,b,c and then uses those to predict a single y value.

R-squared is the success metric for measuring how well our output variable is being explained by our input variable, and is measured between 0 and 1.

The mean is usually a line y = some number, so a horizontal line represents the mean, thus this is a good base model to compare how well our linear regression model fits the data vs the mean, which is done using the R2 metric

Process

1. Get data
2. Split data into inputs and outputs
3. Choose a % split and split inputs and outputs into train and test pairs
4. Instantiate your regressor
5. Fit the model to the training data
6. Use the trained model to predict output values from the input test set
7. Compare the predicted output values against the output values and assess using R2 metric

The equation for a straight line is y = mx + c, where y is how far up the y-axis we are, m denotes the slope of the line i.e. for every unit we move along the x-axis, we move this many units up the y axis, x denoting how far along the x-axis we are, and c denoting the intercept, which is the point our line would cross the y-axis when x is equal to 0

Worth looking at some y = mx + c examples to understand how it works, as this equation is the basis for linear regression. Once the code has learned the best fit values for m and c from your data, then any x value you plug in will spit out a predicted y value.

So how does the algorithm find the line of best fit from your data?

It uses a method called “Least Squares” – which is an approach used in regression problems to approximate a “line of best fit” by minimising the sum of the squared residuals. A residual is the distance from our approximated line of best fit and the actual data points that it is trying to represent. If you have 5 data points, then you will have 5 residual values, as you can measure 5 distances from the regression line. Often, you get negative residual values, depending on how you minus the data point y coordinates and the line y coordinates, therefore we square the residuals to ensure we are always working with positive numbers. REMEMBER! We only care about predicting y values for a given x, so we do not care about x distances from the line, as we only want to know the error in y for a given value of x.

To get the best line programmatically, a gradient descent algorithm is used under the hood to find out the line equation that produces the lowest sum of squared residuals.

R2 metric

R2 shows the percentage of variance in our output variable y, that is being explained by our inputs variables x

In other words, it shows how well our regression line fits the data compared to the mean line

R = correlation (sometimes referred to as the coefficient of correlation)

Correlation = 0 means a change in x has almost no effect on y

Correlation = 1 means a change in x leads to a direct change in y

Correlation = 0.75 means a change in x leads to a mild change in y

Correlation = -1 means a change in x leads to a strong negative correlation in y

R2 means the correlation score falls between 0 and 1 for easier comparison, so 0.75 and -0.75 will have the same R2 score as we are just using it to measure goodness of fit, not the direction of the relationship

Also in reality, most linear regression models involve multiple input values, so making everything positive for this makes comparison far easier

R2 = (SSR[mean line] – SSR[regression line]) / SSR[mean line]   
where SSR = sum of squared residuals

The mean value tells us how well we can predict the data just by using the y values themselves, whereas the R2 metric tells us how much better we can predict the data by introducing our linear regression (which also factors in input variables) than if we just used the mean. If we had an 87.5% R2, this means 87.5% of the variance in the y values is explained by the linear regression model, and the remaining 12.5% must be explained by some other variables.

We calculate R2 on the training, test and validation sets and hope for similar R2 values in all sets, as if the R2 is far higher on the training set than the test, then this is a sure indicator of overfitting. Solving overfitting is solved by getting more data or data that better represents the real world, or getting rid of features that have weak relationships with the output variable, or those that have high p values in the output summary.

Multiple linear regression

Here we are looking to fit a plane of best fit rather than a line of best fit, but this is hard to visualise.

This would be denoted by equation:

y = (m1 \* x1) + (m2 \* x2) … (xn + mn) + c

m values here are called coefficients rather than slope values

y and c are the same as before, y is the prediction, c is the intercept

x1 etc just represent the different features that affect our dependent variable

e.g. features that affect house price are house size, house age and intercept

so x1 would be house size, x2 would be house age, then m1 and m2 and c would be learned by the algorithm

so m1 = 240 means we’d expect the price to rise by £240 for every increase of 1 metre in floor space, and m2 = -2500 means we’d expect to see a -£2500 fall in price for every year older a house is

c = + £50,000 means we have a baseline of £50k for house price, and is the value when house size and house age = 0, but obviously house size would never be 0, as this is just the value at which the line crosses the y axis and doesn’t make sense properly when applied to the real world

Metric for multiple linear regression

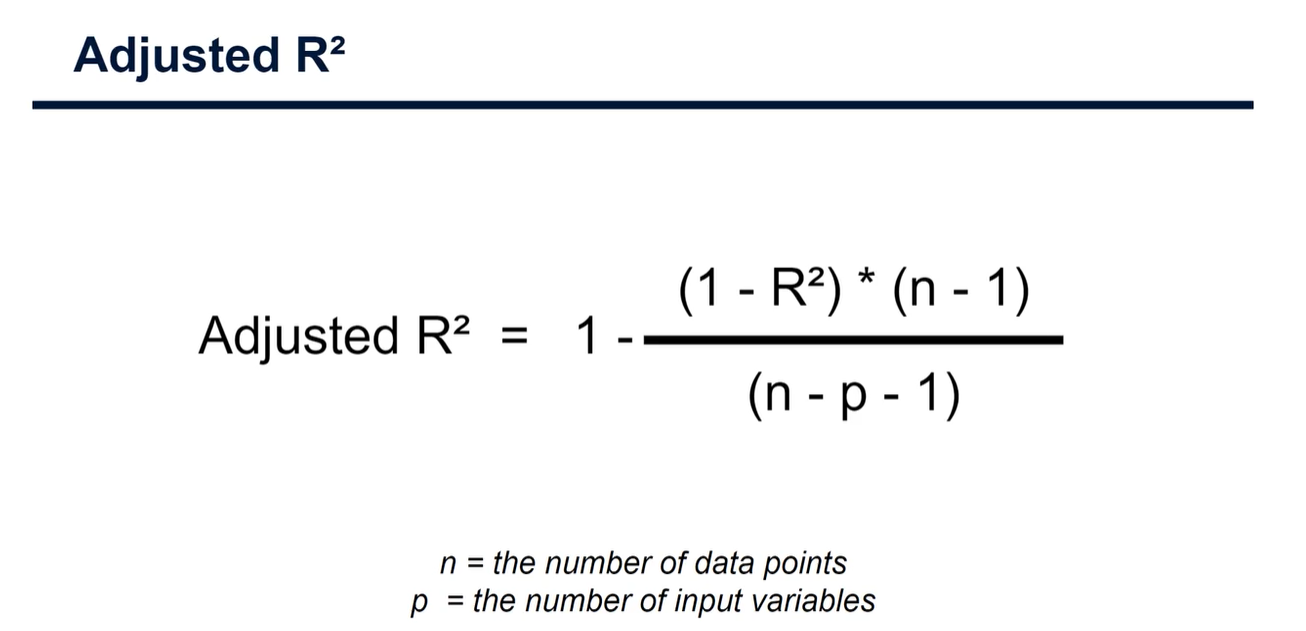
Adjusted R2

This is the same formula as for the normal R2 calculation, except we calculate it for each input variable and then add them together. Adding each together does make sense, but it can cause a problem – and only really becomes a problem if we are throwing a lot of input variables into the problem and just seeing what happens. This is because every input variable added to a model increases the R2 value and never decreases it, even if by chance. So just throwing 100 input variables into a model without any proper consideration for whether they really do have a relationship with the dependent variable is not very clever, as we’d overinflate the metric with random and coincidental relationships from noise (through a high number of small values being added together).

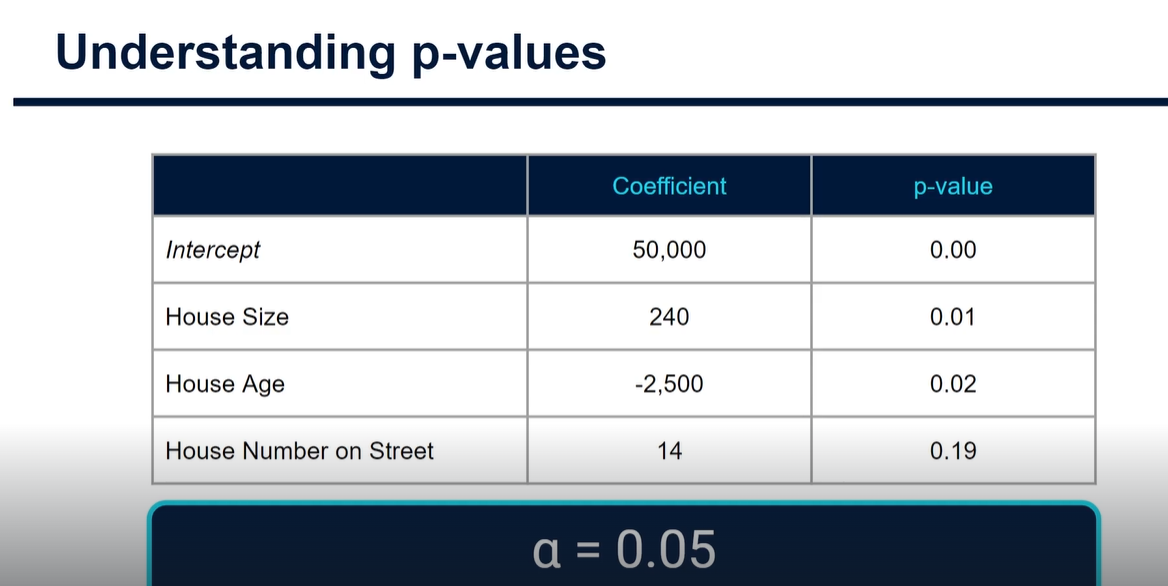


The 2nd example will never have an R2 value smaller than the first, even though the added features don’t really correspond much to the output variable, and have very low logical sense on impacting it. The 2nd model would obviously be a lot worse and very inefficient, as most of the work is being done by the first 2 features, which logically make sense in impacting house prices.

Adjusted R2 is used to compensate for this addition of input variables, and only increases if the variable improves the model above what would be obtained by probability. It scales R2 by the number of input variables that have been used.



This is not readily available in scikit learn, so we need to code it up ourselves.



In adjusted R2, we use p values to understand which features have a greater impact on our output variable than random chance. We have a null hypothesis that each feature has no impact on our output variable, and then we test this for statistical significant. If the p value is above our threshold, in this case 0.05, then we say that there is not enough evidence to suggest that our null hypothesis is not correct, and we accept it within our confidence interval. So here we can say that house size and house age seem to have a statistically significant impact on house price, whereas house number on street does not. Therefore, in the next iteration of our model, we’d likely drop the house number on the street feature and keep only house size and house age, to prevent our R2 value and our model from being unnecessarily influenced by a statistically insignificant feature.