[1]

In this set of lectures, we will turn our attention to the problem of predicting continuous value targets, a process called regression. Unlike classification, which makes predictions from among a finite range of categorical values, regression models output continuous values as predictions of some target feature values such price, weight or height. We will present one of the most popular approaches to regression modelling called linear regression and show how this model generalise well to multi-feature predictions of a single target feature.

[2]

To begin to build an intuition as to how linear regression works, let’s consider the simple case of two positively or negatively correlated features. We can graph these features on a 2D scatter plot to see how closely they are correlated. The scatter plot uses the feature values as the x-coordinate and y-coordinate values respectively. Correlation here means that one feature’s value is a good predictor of the second feature and visa versa. In the first example, this is very clear with the plotted points almost showing a straight line through them, implying that they are strongly, linearly correlated. The second example is less obvious, but there is a hint of a linear correlation here. We can overlay a straight line on these data points showing this relationship more clearly. For a two-feature case when there is a linear relationship, we can fit a line that relates the features as a linear equation of a y-axis intercept b and a slope m

[3]

When we fit a line to linearly correlated data, we don’t expect this line to be a perfect fit, at least not for most real-world datasets we will encounter. The process of linear regression is all about finding a line that is as close as possible to all the data points, a compromise best-fit. The way linear regression does this is to collectively minimise the distances from all the points to the line. These distances are called residuals and are measure as the y-axis offset from the line to the points. The residuals can be positive if below the line or negative if above the line

[4]

Formally, we can define an error function (also called a loss function) which measures the total residuals from the line to the points in our dataset. Remember the points are the dimensional coordinates of our feature space. Before presenting the error function, we generalise the notation for the linear correlation function to use weights w[0] in place of the y-intercept and w[1] in place of the slope. Later, we will generalise our linear systems to N-dimensions where this new notation will be more useful. Now we can define our error function as the average of the sum of squared errors. The error is the difference between the actual target feature value and the value predicted by the linear model. We square the error to get rid of negative residuals and we multiply by the constant one half to make some of our algebra easier in a later step. The subscripts in these formulae represent the dataset instances. The number indexing represents the feature index within an instance.

[5]

Still considering a system of two weights and a single descriptive feature, we can visualise the error function as a continuous surface on a 3D graph where the x-axis represents the first model weight w[0] and the y-axis represents the second model weight. The z-axis represents the error for various choices of model parameter w[0] and w[1]. The intuition you should have is that as we chose different slopes and y-axis intercepts of an arbitrary number of possible lines, the total error of squared residuals will go up and down as we vary these parameter values. This variation can be graphed as a surface similar to the one shown. It turns out that, because this is a convex function, there is a point on this surface, known as the global minimum, which provides the optimal parameter values for w[0] and w[1] to minimise the total error.

[6]

To obtain the global minimum for this system of two weights and one feature we take the partial derivates of our loss function with respect to w[0] and with respect to w[1]. When these partial derivates are equal to zero, we have reached the global minimum for this system. Hopefully you will already have reasoned that finding these values is not feasible either manually or as a trial-and-error process and requires some significant computation. We will present the gradient descent algorithm in a separate lecture which is a way of efficiently and automatically finding the global minimum.

[7]

Now that you have an intuition how linear regression models work for one descriptive feature and one target, we can generalise our system to consider N features. These N features form an N-dimensional feature space (or hyperplane) with each feature’s value representing some offset from the origin along its dimensional axis. Therefore we introduce corresponding weights w[1] through w[n] for each of the N features. The w[0] weighting, which originally represented the y-axis intercept in our 2D model can be folded more neatly into our system if we introduce a new dummy feature f[0] which always has a value of 1. This allows our linear model to have a more compact form as shown

[8]

Now that we have a model of N features, it’s worth pausing to consider how an implementation of a linear regression model would look in code and what kind of runtime performance we might expect. Suppose that there may also be many features per instance. With the compact result for our linear model, we can that there are two ways to look at the computation. One way is as the sum of weighted features and the other is the dot product of a vector of weights and a vector of feature values. You might think that there should be no difference between these approaches given how you know dot product works. However, many libraries include specific optimisation for linear algebra operations such as dot product which take advantage of CPU featires like single instruction, multiple data or SIMD. If a GPU is available, then vector and matrix operations may be accelerated for better performance. The recommendation, if coding machine learning models involving operations like this is to use the libraries and not to try to hand code these using language level loops

[9]

One of the most attractive features of the linear regression model is that it is relatively easy to interpret the model results. If will be clear from the output models final weights vector which features play a part in the model prediction and which don’t. This knowledge may help in dimensionality reduction decisions or in informing the training of other models and what features you should include. However, you have to be careful when dealing with the scaling of different dimensions. Some feature ranges may differ significantly from others so their relative effects may not be so obvious. One way to deal with this is to normalise all of the features into a specific, comparable range before training so that post model analysis is easier to do

[10]

The best way to determine the importance of a feature is to perform a formal statistical test on the feature of interest. In this test we assume that the null hypothesis is true and that the feature is not significant in the model. We then try to refute the null hypothesis to see if the feature does play a significant role. The p-value is the probability that the our test statistic value is as big as or greater than a value computed by chance. If the p-value is less than a prior threshold value such as 5%, then we can reject the null hypothesis and conclude the that feature is significant. The test statistic we use here is based on the ratio of the associated weight and the standard error of the descriptive feature in question. We then use a standard t-statistic lookup table to find the appropriate p-value and make our determination.

[11]

Categorical descriptive features need to be transformed before including in a linear model. This is because the arbitrary numerical values in the feature range are meaningless as multipliers of weights for a model fit.

[12]

One simple and popular way to transform categorical features is called one hot encoding. In this process we extend the dataset by adding one new column for every possible value of the categorical feature. Then we put a one in that new column for the corresponding feature value in the original descriptive feature. During model training, we exclude the old column and only consider the newly created ones. For example, suppose we have a categorical feature having four values, one through four, then we would create four new columns with a binary values zero or one to represent the original values. In model training, only the ones would activate, and each categorical value would have equal weighting.

[13]

When considering machine learning model performance, we measure how well the generated model predicts targets from a test set for which we already know the results. In regression models we need to use different performance measures than those we use in classification models. By its nature, linear regression is already minimising the residual error within the training set to fit a hyperplane model for the points it trains with. We can generate some error measures for a test set using metrics like mean-squared error or mean-absolute error to verify that it performs well on data not yet seen. The results may tell us more about the training set than the model if the error is unexpectedly high. It’s possible that the training set is not representative of the population. For this reason, it is recommended, if possible, to train and test the model multiple times with different dataset subsamples to ensure consistency in the results

[14]

In summary, linear regression is an error-based learning approach which treats the feature values as coordinates in a N-dimensional hyperplane. The resulting predictor is the sum of the weighted feature values where the weights are automatically discovered by the gradient descent algorithm which minimises the residual error between the training data and the prediction function. Linear regression generalises well to multi-feature modelling, is efficient to compute and is easily interpretable. It is arguably the most important regressor algorithm in use in machine learning today.