# 1 Regularisation

When using too many features, it is often the case that overfitting happens where the model fits well to the training data, but does not generalise to unseen data. A typical way to deal with overfitting is to reduce the number of features, either manually or automatically. This is done because some input features may not be very relevant to the general problem, but still affect the prediction.

**Regularisation** is another method to reduce the effect of some input features without removing them completely. It works by keeping all input features, but using multipliers to reduce the scale of certain $\theta$ parameters. This allows more relevant features to have more impact while still keeping a lower effect of less relevant features. The parameters are penalised using a large multiplier to cause them to be minimised because of the increased constant. For example $... + 1000\theta_3 + 1000\theta_4$. This changes the cost function by adding a regularisation parameter $\lambda$.

$$\text{Cost} = \frac{1}{2m}\left[\sum_{i=0}^{m}(h(x^i) - y^i)^2 + \lambda\sum_{j=1}^{n}\theta_j\right] \tag{1}$$

The regularisation parameter $\lambda$ determines how much to suppress the $\theta$ parameters. If $\lambda$ is too large, then we may underfit as we have minimised the $\theta$ parameters too much. If $\lambda$ is too small, then we may continue to overfit as it does not do enough to reduce the effect of $\theta$ parameters.

The regularisation parameter can also be added to the normal equation.

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\right)^{-1} X^T y \tag{2}$$

In general, regularisation can be thought of as adding an extra component to the equation of minimising the cost function.

$$J(\theta) = L(\theta) + R(\theta) \tag{3}$$

where $J(\theta$ is the cost, $L(\theta)$ is the loss and $R(\theta)$ is the regularisation. Further, regularisation has a computational advantage over feature selection, as additional training and searching of models is not needed.

## 1.1 Ridge regression

The $\lambda\sum_{j=1}^{n}\theta_j$ term for regularisation can be calculated using the $l_2$ norm, usually denoted as $||\theta||_2^2$. The $l_2$ norm is the euclidean norm, which is a sum of squares of all elements, defined as:

$$||\theta||_2 = \sqrt{\sum_{j=1}^{n}\theta_j^2} \tag{4}$$

It measures the distance of $\theta$ from zero. As $\lambda$ increases, the $l_2$ norm will *always* decrease. The reason regularisation has an advantage of a simple least squares regression is based on the **bias-variance trade-off**. As $\lambda$ increases, the flexibility of the model decreases, leading to decreased variance, but increased bias. For some values of $\lambda$ up to a limit, the variance will decrease rapidly with only a small increase in bias, which reduces the overall mean squared error of the model.

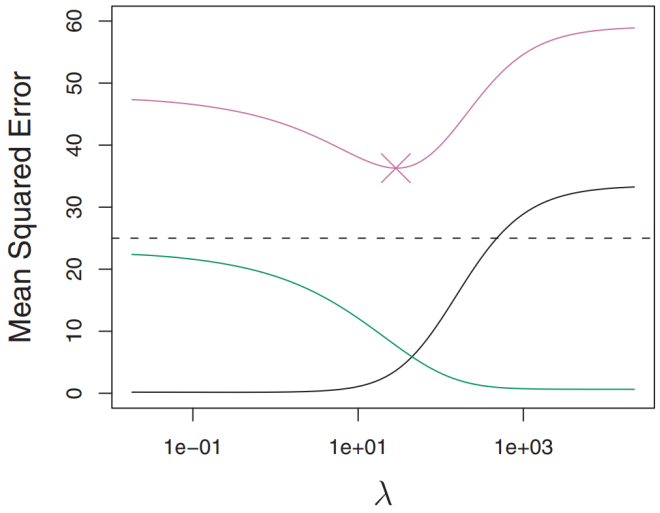



Figure 1: Plot showing why ridge regression leads to lower MSE for some values of $\lambda$. Squared bias (black), variance (green) and test MSE (purple) for the ridge regression predictions on a simulated dataset as a function of $\lambda$. Taken from *An Introduction to Statistical Learning*.

### 1.2 Lasso regression

The disadvantage of ridge regression is that while it will shrink all coefficients towards zero, it will not set them to be exactly zero unless $\lambda = \infty$. In other words, for datasets with a large number of features, increasing *lambda* will reduce their effect but not exclude any of them completely.

Lasso regression works the same way as ridge regression, except instead of using the $l_2$ norm, the $l_1$ norm is used. Rather than being a sum of squares, the $l_1$ norm is simply a sum of all elements.

$$||\theta||_1 = \sum_{j=1}^{n} |\theta_j| \tag{5}$$

Lasso regression also shrinks the coefficient estimates towards zero. However, the $l_1$ penalty is able to force some coefficient estimates to be equal to zero when the *lambda* value is large enough. In this sense, lasso regression performs variable selection like selecting the best features. It can be said that lasso regression yields *sparse* models because the models may only involve a smaller subset of features.

## 2 Evaluation

The goal of supervised machine learning is to train the models on known data in order to predict new data. To be able to evaluate how well the model works on new data, any given data has to be split into separate training and test data sets. The testing data is used as a final step after training models to evaluate how well they perform. Testing data should *never* be used to train, as otherwise this would bias the model heavily towards the testing data, rather than towards general unseen data. Further, testing data should not be used for any step of the machine learning process except for checking the performance of trained models at the very end.

### 2.1 Source of errors

- Not enough data which leads to overfitting to available data and inability to generalise

- Fine tuning parameters on test data which still causes overfitting because the model is designed to work well only on available data
- Training or making decisions based on testing data could lead to very low error rates on available data, but the heavy bias is undesirable

Most often, the source of errors in machine learning comes from overfitting in some way, either from looking and training towards the test data, or from not enough data or too many input features.

## 2.2 Classification metrics

Classification differs from regression in that all errors are treated the same. It is not as easy as using functions such as the root mean squared error or $R^2$ coefficient of determination. Classification are either correct or incorrect, which makes errors simpler as there is no need to calculate how far the predicted value is.

- Correct classification = true positives + true negatives
- Incorrect classification = false positives + false negatives

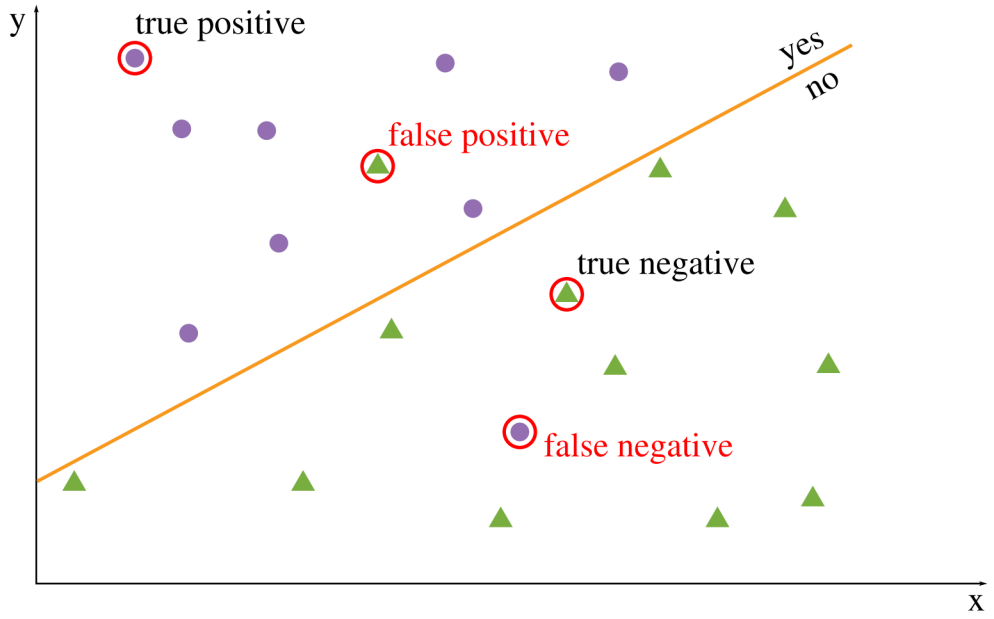



Figure 2: Examples of the different types of correct/incorrect classification

With different combinations of true/false positives/negatives, different metrics can be calculated for classification problems. For example, the accuracy (or classification rate) is defined as:

$$\text{accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{total positives} + \text{total negatives}} \tag{6}$$

There are many further metrics that can be used for classification, most notably are **precision** and **recall**.

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \tag{7}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \tag{8}$$

The precision is the fraction of correct classifications among all classified data points and the recall is the fraction of correct classifications among all relevant data points. In other words, the precision tells us how much of what was classified is correctly classified, while the recall tells us how much of the positive classifications were found out of the total positive data points. The two scores represent a trade-off as to how correct the classifications are against how many relevant data points were found. This trade-off depends on the actual classification problem. For example, in the case of cancer identification, it may be better to bias towards false positives (higher recall, lower precision) to be on the safe side.

The trade-off between precision and recall can often be visualised by varying a threshold on the classification and plotting a precision-recall curve.

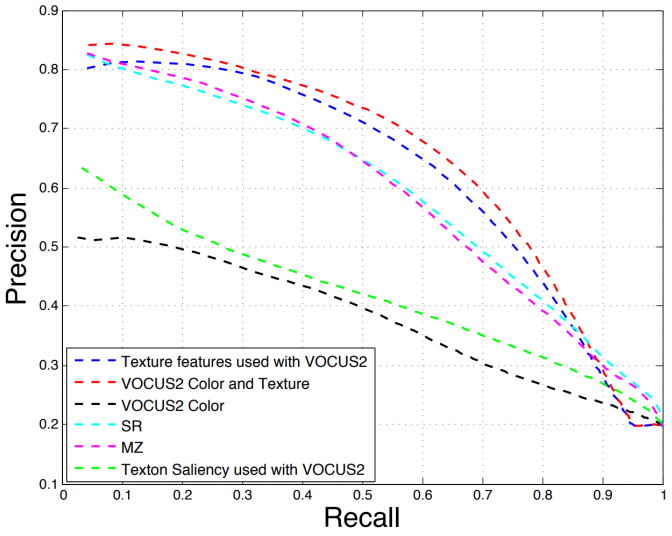



Figure 3: Precision-recall curves.

It is often difficult to compare precision-recall curves as different curves could represent different trade-offs. It is more useful to have a single number to compare, so the area under the curve is used as an alternative. The larger the area, the better the performance. This is because a good model that achieves both high precision and recall will have a larger area under the curve. This area is the **average precision** for one binary classification task averaged over all possible parameter values. With multiple classes, a precision-recall curve would be needed for each class in the one-vs-all method. To better visualise this for an average over all classes, a **confusion matrix** can be used.

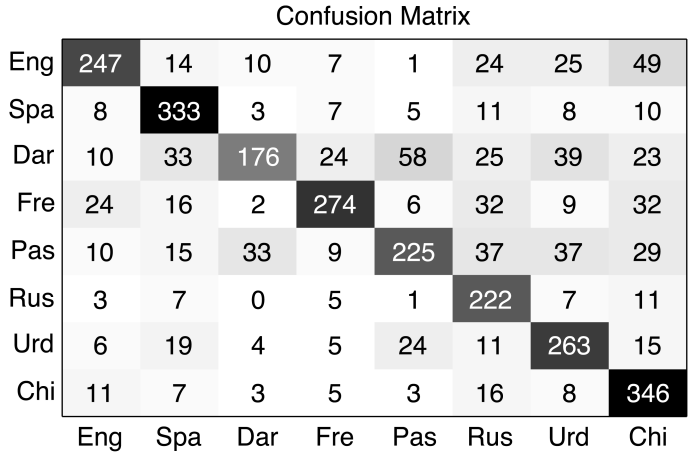
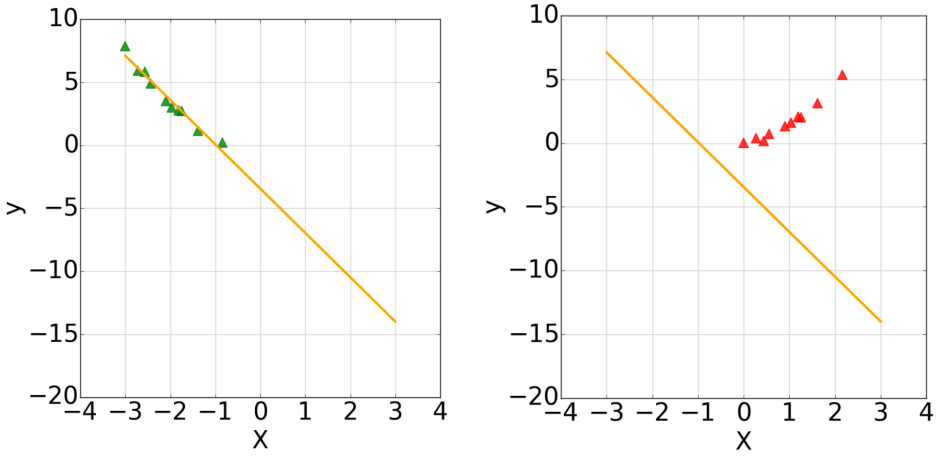



Figure 4: Example of a confusion matrix.

The diagonal represents correct detections for that class. The rows and columns represent false positives and false negatives, as they are predicted the wrong class. Confusion matrices are useful to see which classes are better predicted compared to other classes

## 2.3 Validation

Because testing data is sometimes completely hidden, the given training data may need to be split into a validation set, which is used to evaluate the quality of the model. There is an issue during the splitting phase of training and validation data where the two sets are not good representation samples from the dataset. For example, if the dataset is split in half exactly, many important features may be missed.




(a) Prediction on training data (b) Prediction on validation data

Figure 5: Extreme example of an issue that can occur in splitting training/validation data.

The split training/validation datasets have to be representative samples from the domain of the entire dataset. However, this can be difficult to do without looking at the dataset, which compromises the experiment. A method to deal with this is called **stratified sampling**. In classification, this would ensure a representative number of classes are in each dataset. For regression problems, certain important features need to be used as the feature for stratification, for example gender proportions in data about people.

Another issue with splitting the data into a validation set is that it reduces the amount of data available for training. Further, randomly splitting the dataset, even with stratified sampling could lead to getting "easier" or "harder" datasets to work with. Another method to evaluate the performance of trained models is **k-fold cross validation**.

### 2.3.1 k-fold cross validation

The idea behind k-fold cross validation is to split the dataset into $k$ equally sized subsets and run training $k$ times, leaving out one of the subsets for testing. Each $k$ samples is used for validation exactly once. Larger values of $k$ means more splitting up of the data, which gives more reliable results, but requires more computation.

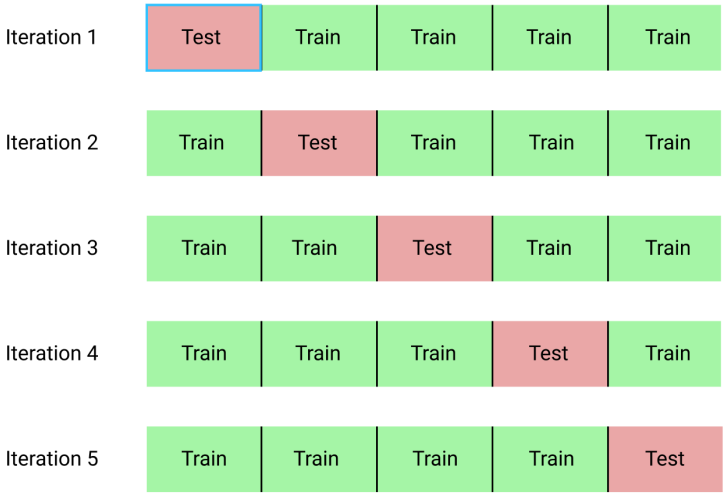



Figure 6: Example of 5-fold cross validation.

# 3 Data preparation

Often times the raw data received at the start of the machine learning process requires cleaning or preprocessing. This can be due to errors during data gathering, for example negative readings that aren't possible, or the way the data is presented, such as categorical values like colour (red, blue, green) or time. Data such as this has to be processed to be able to be passed to any machine learning algorithm, as we need numerical inputs in some cases.

Example data preparation process:

- Clean data
- Convert data types (categorical with dummy variables)
- Find mean and standard deviation of features
- Normalise data
- Select a subset of features
- Split into training and test/validation set
- Train algorithm
- Evaluate on test set

## 3.1 Categorical inputs

| Make | Horsepower | Color | Accidents |
|---|---|---|---|
| Aubi TT | 220 | Red | 13 |
| Citreom C4 | 160 | Blue | 1 |
| Fauxhall Corso | 98 | Red | 2 |
| Citreom C4 | 124 | Black | 0 |
| Aubi TT | 196 | Red | 7 |

Table 1: Example data with categorical inputs (colour)

Some algorithms such as decision trees and random forests can deal with categorical data, but for models such as linear regression, numerical data is required. There are two main methods of transforming categorical inputs into numeric inputs: **enumeration** and **dummy variables**.

### 3.1.1 Enumeration

To encode categories by enumeration, each class of the category can be assigned a number, for example assigning red = 1, blue = 2 etc. This is a very simple way to transform categories to numbers but creates a new issue. The distance between the numbers subtly implies a similarity between the values. For example if red = 1, blue = 2 and black = 3, we are implying that blue is similar to both red and black while red and black are not as similar to each other. Sometimes this can be desirable if encoded correctly to give more detailed information, such as if the colour numbers were ordered in a way that represented the difference.

| Make | Horsepower | Color | Accidents |
|---|---|---|---|
| 1 | 220 | 1 | 13 |
| 2 | 160 | 2 | 1 |
| 3 | 98 | 1 | 2 |
| 2 | 124 | 3 | 0 |
| 1 | 196 | 1 | 7 |

Table 2: Transformed categorical make and colour data using enumeration

### 3.1.2 Dummy variables

Instead of enumerating all the categorical values, a vector of ones and zeroes is used where each value corresponds to each possible category and the 1 or 0 represent true and false.

| TT | C4 | Corso | Horsepower | Red | Blue | Black | Accidents |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 220 | 1 | 0 | 0 | 13 |
| 0 | 1 | 0 | 160 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 98 | 1 | 0 | 0 | 2 |
| 0 | 1 | 0 | 124 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 196 | 1 | 0 | 0 | 7 |

Table 3: Transformed categorical make and colour data using enumeration

## 3.2 Input cleaning

When data is collected, there are often cases where mistakes or other errors appear and have to be cleaned beforehand. For example, all columns having the same data type and all rows having

the same number of columns. Further errors can be dealt with in two ways, either try to fix them, for example inserting an average value to replace a NaN value, or remove the row entirely. The latter case may not be desirable in cases with little data. Other examples include spelling errors, negative values, faulty sensors and infinity values.

## 3.3 Feature selection

It was mentioned before that a reduced feature set can help reduce overfitting in the case of regularisation. In some cases, some features should be discarded entirely, especially if they are not relevant. There are a few good reasons for selecting only relevant features to use:

- It can improve accuracy with some models and reduce overfitting
- It can improve interpret-ability, as it is easier to understand models based on fewer parameters
- It can make infeasible problems feasible by significantly reducing the computational requirements

Feature selection can be done via **univariate tests** where the correlation between each features and the output is calculated and the good well correlated features are selected. Further, correlation between feature pairs can be calculated, as features that are well correlated to each other may be redundant.

There are also automatic algorithms for feature selection to select the $k$ best features which give the best performance. Most notable, forward and backward stepwise selection.

### 3.3.1 Stepwise selection

In stepwise selection, features are either incrementally added or removed until the best number of features is found. In forward stepwise selection, each feature is added incrementally, trained on a new model and evaluated on validation or testing data. The best model is kept and another feature is added. This is repeated until the models no longer improve in performance and the best feature subset is selection.

Backwards stepwise selection works the other way by starting with all input features and removing features which affect the model the least.

## 3.4 Data augmentation

Data augmentation is a good technique both to increase size of the dataset and help models better generalise to new data. This is especially good for complex and modern classifiers, which may only work in very specific cases without data augmentation and need very large amounts of data. Augmentation can be done by transforming existing data in some way. For example shift, rotate, scale, distort, flip, change contrast in images. Domain knowledge is useful here to know how to augment the data.

Figure 7: Example of data augmentation on images of a stop sign.