Google Machine Learning Course

Amaan Ahmad

July 2021

Abstract

Machine Learning Crash Course with TensorFlow APIs Google's fast-paced, practical introduction to machine learning. Learn and apply fundamental machine learning concepts with the Crash Course, get real-world experience with the companion Kaggle competition, or visit Learn with Google AI to explore the full library of training resources.
Course found here

Contents

1 Framing

1.1 Labels

A label is the thing we're predicting — the y variable in simple linear regression. The label could be the future price of wheat, the kind of animal shown in a picture, the meaning of an audio clip, or just about anything.

1.2 Features

A feature is an input variable — the x variable in simple linear regression. A simple machine learning project might use a single feature, while a more sophisticated machine learning project could use millions of features, specified as: $x_1, x_2, ..., x_n$
In a spam detector example, the features could include the following:

- email address
- words or phrases in the content of the email
- time of day that email was sent

1.3 Models

A model is the relationship between the features and the label. For example, the spam detection model the features may associate some features stronger with "spam" than others. There are two phases in a models life:

- **Training** - or the **Learning** phase. In this phase, you show the model labelled examples to enable the model to gradually learn the relationship between the features and label.
- **Inference** phase. This phase involves using the model to unlabelled examples in order to make useful predictions.

1.4 Regression or Classification

- **Regression** models predicts continuous values i.e. value of a house or probability.
- **Classification** models predict discrete values i.e. if an email is spam or not spam, or is an image a dog, a cat or horse.

2 Descending into ML

2.1 Linear Regression

For a linear relationship, we can model it using the equation for a line, which is:

$$y = mx + c$$

However, in machine learning we use:

$$y' = w_1 x_1 + b$$

Where:

- y' is the predicted label - the output.
- w_1 is the weight of feature 1.
- x_1 is the feature.
- b is the bias - also known as w_0.

This model only uses one feature however if we wanted to use multiple features, each feature would have a different weight and hence the formula would look like this:

$$y' = b + w_1 x_1 + w_2 x_2 + w_3 x_3 + ... + w_n x_n$$

2.2 Training and Loss

Training a model means determining good values for all the weights and the bias from labelled examples. A machine learning algorithm determines a suitable model by attempting to find a model that minimizes loss - this process is called **Empirical Risk Minimization**.
Loss is the penalty of making a bad predictions. The models aim is to find a set of weights and biases that have a *low loss* on average. Figure 1 demonstrates how a model attempted to find an appropriate model. Here the Arrowed lines represent the loss and the blue line represent the predictions.

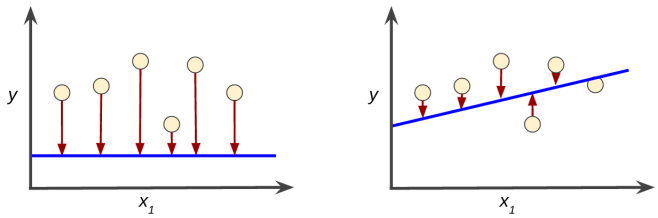


Figure 1: Shows a high loss model (left) and a low loss model (right)

2.2.1 Squared Loss

We use a loss function on linear regression called **Squared Loss** or L_2 loss. It is the square of the difference of the label and the prediction. Or:

$$(observation - prediction(x))^2 \text{ or } (y - y')^2$$

2.2.2 Mean Square Error

MSE is the average squared loss pre example over the entire dataset. MSE is calculated but summing all the square losses and then divide by the number of examples:

$$MSE = \frac{1}{N} \sum_{(x,y)\in D} (y - prediction(x))^2$$

Where:

- (x, y) is an example in which
  - x is the set of features that the model uses to make predictions.
  - y is the example's label.
- $prediction(x)$ is a function of the weights and bias with the set of features x.
- D is the Data set which contain numerous labelled examples.
- N is the number of examples in D.

3 Reducing Loss

3.1 Iterative Approach

Similar to the *Hot or Cold game* where we start with a guess then see what the loss is, then try another guess of model parameters, (w_1 and b_1), and see if you are closer to the smallest loss. Figure 2 shows use this process.
For example for a linear regression:

$$y' = w_1 x_1 + b$$

We choose random numbers for example; $b = 0$ and $w_1 = 0$

The initial values we choose does not matter. We then get the predict value y'. Using this predicted value we use the loss function to determine the loss. The program then iterates over to find the corresponding pair of values to find the lowest loss. Eventually, the loss stops changing and this means the model has **converged**.

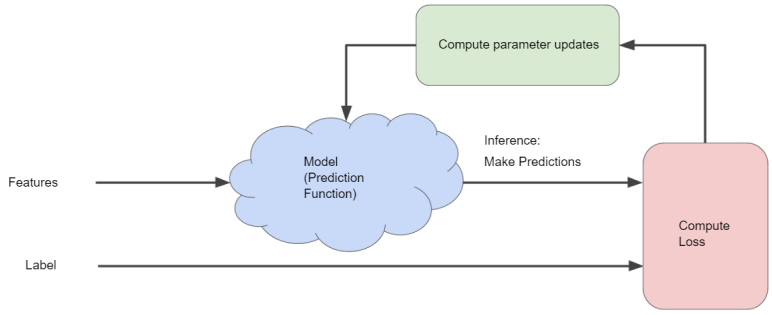


Figure 2: Shows the trial and error approach which machine learning algorithms us to train the model

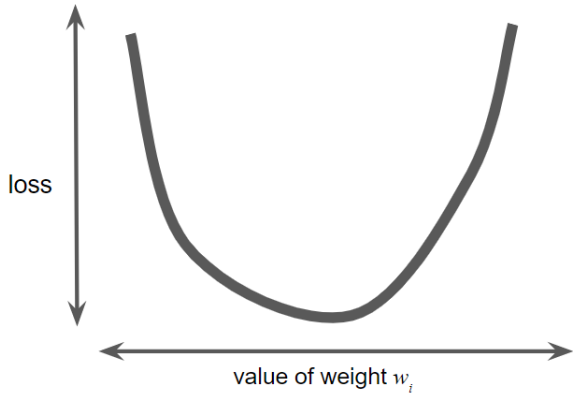


Figure 3: Shows the convex shaped graph

3.2 Gradient Decent

For linear regression problems, the resulting graph of loss vs w_1 will always be convex. which looks like this:

As shown in figure 3, we can see that the convex problem only has one minimum and hence it is where the loss function converges. To find the convergence point we use a mechanism called **gradient descent**.

The algorithm picks a starting point, finds the gradient of the curve, then the algorithm takes a step in the direction of negative gradient to reduce loss. The amount that the step is is very important. As shown below in figure 4.

The step size is also referred as the **learning rate**. They are used to determine the next point. **Hyperparameters** are "knobs" that programmers tweak in machine learning to tune the learning rate. As you can see from figure 4, if the learning rate is too small the learning would take too long. If the learning rate is to big, there is a possibility that you can overshoot the optimal point. To select the best learning rate, we use the Goldilocks learning rate. It works like this:

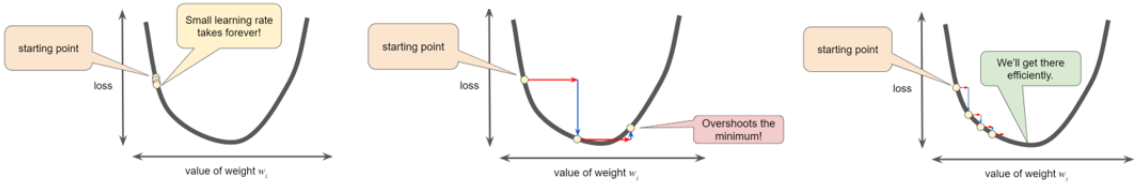


Figure 4: Left graph shows the small steps, Middle graph shows the effect of large steps and the Right graph shows the Goldilocks approach

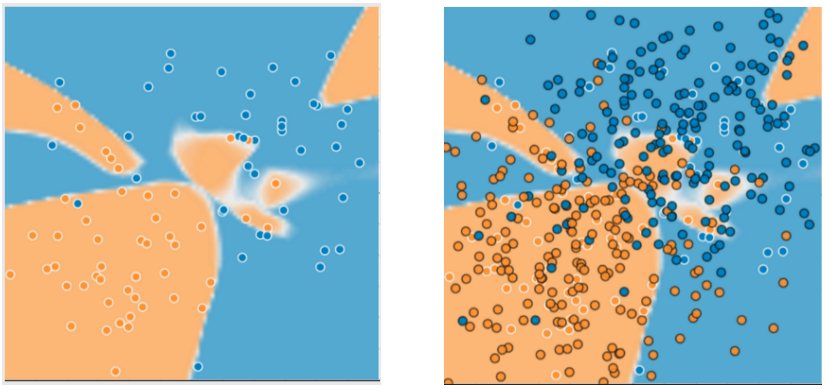
- If the gradient of the loss function at the point is small then you can try a larger learning rate.
- And vice-versa.

4 First Steps with TF

This section was using code. To see the code please refer to `Code/FirstStepswithTF`.

5 Generalization

The figure below shows the two graphs **overfits** the traits of the data that the model was trained on. The model gets a low loss on the training data set however doesn't as much of a good job of the new dataset.



(a) A complex model for distinguishing sick from healthy trees.

(b) The model did a bad job predicting new data.

Figure 5

This problem is due to the fact that the model is core complicated than necessary.

5.1 William of Ockham

William of Ockham was a 14th century philosopher and loved simplicity. He believed that simpler formulae or theories are better than complex ones. Thus in machine learning terms:

The less complex an ML model, the more likely that a good empirical result is not just due to the peculiarities of the sample.

A machine learning model aims to make good predictions on new, previously unseen data. But if you are building a model from your data set, how would you get the previously unseen data? Well, one way is to divide your data set into two subsets; training set and a test set.

5.2 Machine Learning Fine Print

There are three basic assumptions that aid generalization:

- Drawing examples **independantly and identically (i.i.d)** at random. THis ensures the examples do not influence each other.
- The distribution is **stationary** and doesn't change within the data.
- Drawing examples from partitions from the same distribution.

6 Training and Test Sets

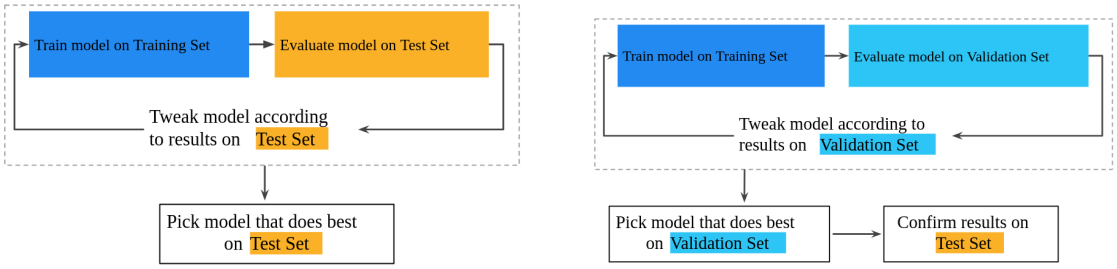
If you have a single data set, it is important to split the data into two subsets; Training set and Test set.
It is important that the test set is:

- Large enough to get meaningful results
- Is representative of the data in the entire set.

Another important thing is that you must make sure that you never train the model using the test set.

7 Validation

In the model mentioned previously, the dataset is split into 2 subsets. The workflow looks like **Figure 6(b)**. However, this poseses a problem for us as the "Tweak Model", i.e. adjusting anything about the model to design a new model, means that we have a greater risk of getting an overfit. Due to this, we divide the set into 3 subsets.
Figure(b) shows how the third partition's workflow and how it reduces the chances of overfitting. This helps as the model is chosen based on the validation set and also that the model is double checked by the test set.



(a) Workflow with two partitions of the dataset

(b) Workflow with three partitions of the dataset

Figure 6

8 Representaion

8.1 Feature Engineering

The left side of **Figure 7** demonstrates the raw data from the source and the right shows the **Feature Vector**, a set of float values.
Feature Engineering is the process in which raw data is transformed into a feature vector.

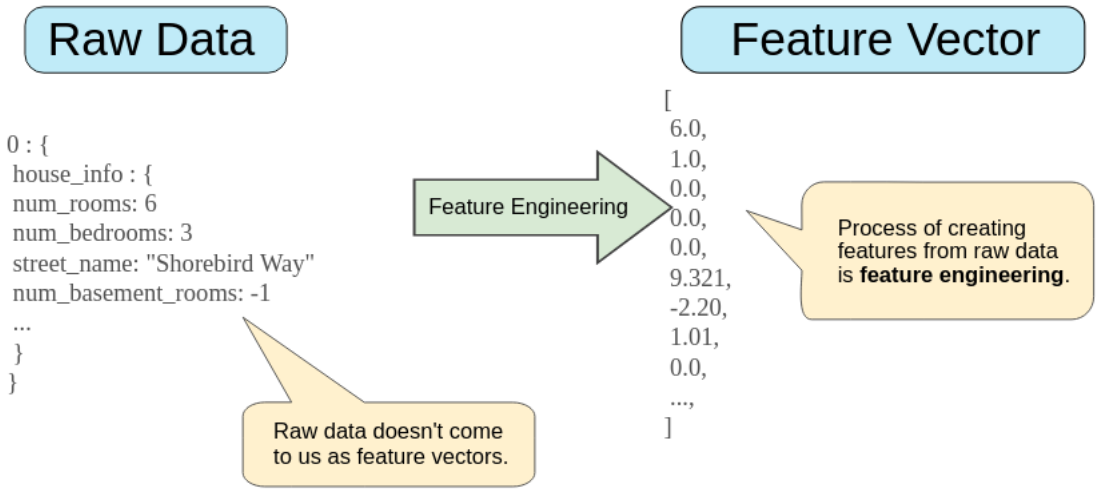


Figure 7: Feature Engineering example

8.1.1 Mapping Numerical Values

Mapping numerical values to floating-point is trivial as they can be multiplied by a numeric weight. The figure below gives an example.

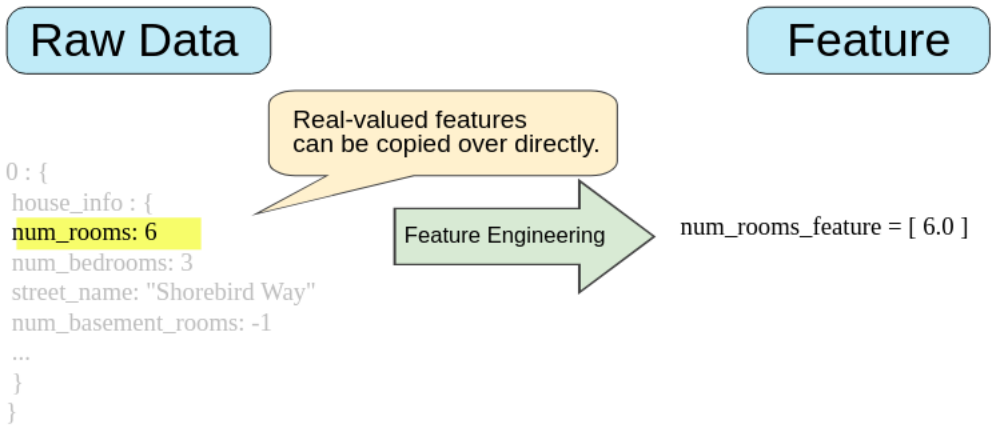


Figure 8: Mapping numerical values to floating point values

8.1.2 Mapping Categorical Values

Mapping categorical is much less than mapping num'Makefile' has erical values. It is best described using an example:
`{'Charleston Road', 'North Shoreline Boulevard', 'Shorebird Way', 'Rengstorff Avenue'}`

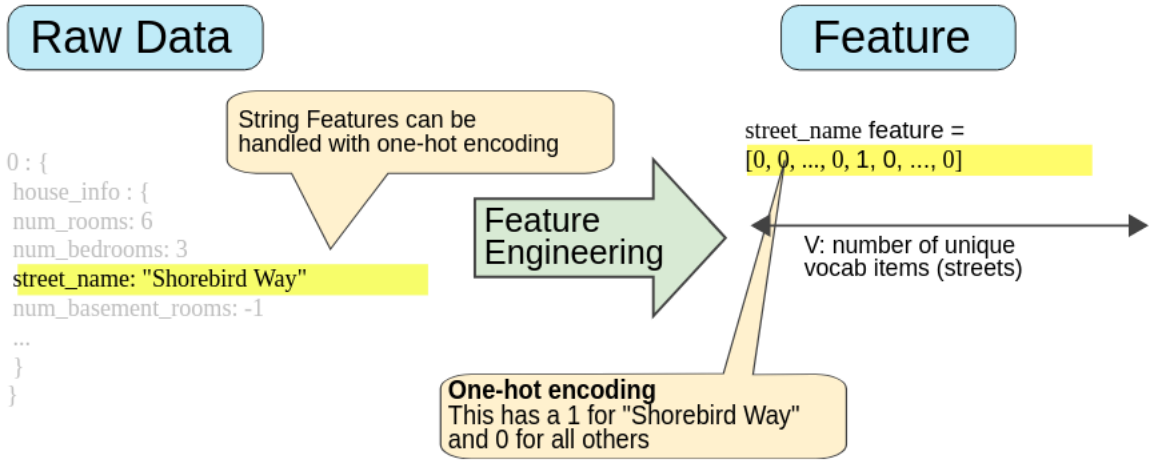


Figure 9

We can accomplish this by defining a mapping from the feature values, which we'll refer to as the vocabulary of possible values, to integers. Since not every street in the world will appear in our dataset, we can group all other streets into a catch-all "other" category, known as an **OOV (out-of-vocabulary)** bucket.
Using this, we get:

- Charleston Road = 0

- North Shoreline Boulevard = 1
- Shorebird Way = 2
- Rengstorff Avenue = 3
- Everything else (OOV) = 4

We then crete a binary vector for each categorical feature. So we set the corresponding vector elements to 1. And all other values to 0. However, multiple values could be set if necessary, this is called **multi-hot encoding**. If only a single value is set, it is called **one-hot encoding**. The figure above gives us an visual example.

8.2 Ensuring a Good Representation of Features

8.2.1 Use Discrete Feature Values

Having many discrete values would help the model see the feature in different situations, and thus would be able to determine when it is a good predictor. For example, having `house_type: victorian` instead of `unique_house_id: 8SK982ZZ1242Z`. Having the latter, would mean the model would not be able to learn anything from it as it would appear very rarly.

8.2.2 Provide Obvious Meanings

Each feature should give a clear meaning for example, having `house_age_years: 27` instead of `house_age: 851472000`. This helps us find noisy data for example `user_age_years: 277`, which is easier to spot that it this cannot be valid.

8.2.3 Don't Mix Actual Data with "Magic Numbers"

Floating point values should be in between 0 and 1, and if there is no value associated, e.g. if the user did not enter a value, then have a Boolean feature to specify if that feature is defines.

8.2.4 Account for Potential Changes with the Meanings of Feature Values

The meaning of the feature value should not change over time. For example, having `city_lol: "london"` instead of `inferred_city_cluster: "23"`. As "london" is less likely to change in comparison to the meaning of "23".

8.3 Cleaning Data

8.3.1 Scaling Feature Values

Scaling means converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range (for example, 0 to 1 or -1 to +1). In this way we avoid the model from inappropriately weighting features.

8.3.2 Handling Extreme Outliers

For outliers, there are numberious ways to avoid this:

- Logging the values - e.g. roomsPerPerson = log((totalRooms population) + 1)
- Clipping, capping the values at a certain point - e.g. roomsPerPerson = min(totalRooms population, 4)

8.3.3 Binning

Binning is grouping values together as shown below.

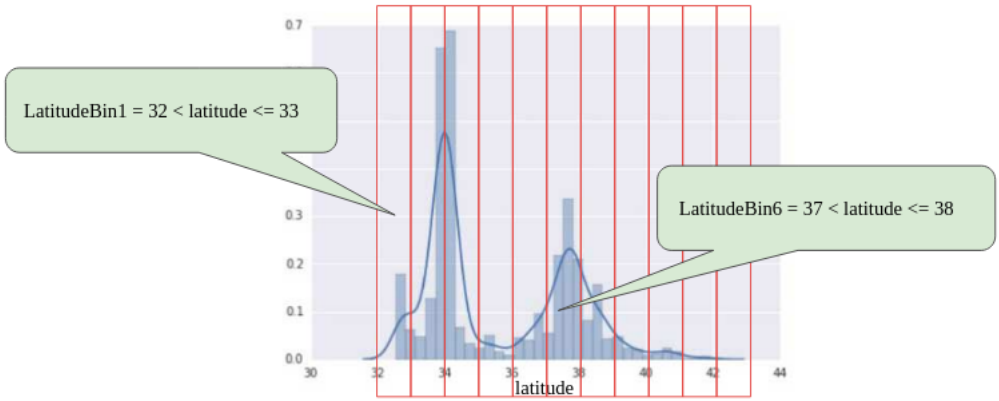


Figure 10: Binning Latitude

Instead of having one floating-point feature, we now have 11 distinct boolean features. Having 11 separate features is somewhat inelegant, so let's unite them into a single 11-element vector.

8.3.4 Scrubbing

Many examples in data sets are unreliable, and hence these values need to be removed from the set.

- **Omitted Values** - If the user forgets to enter a value.

- **Duplicates**
- **Bad Labels** - Labelling errors.
- **Bad Feature Values** - Value errors e.g. adding an extra digit or a wrong one.

To find bad examples, a Histogram would be a good way to represent the data. Other statistics that would help find errors would be; Maximum, Minimum, Mean, Median and Standard Deviation.

9 Feature Crosses

If it is difficult to separate the categories in the model as shown in **Figure 11**, we create a feature cross. A **feature cross** is a synthetic feature that encodes nonlinearity in the feature space by multiplying two or more input features together. For example, if we had features x_1 and x_2, the feature cross, x_3, would be $x_3 = x_1 * x_2$.
This would mean the linear formula becomes: $y = b + w_1x_1 + w_2x_2 + w_3x_3$. Where the weights w_1, w_2 and w_3 are all learnt by the linear model.

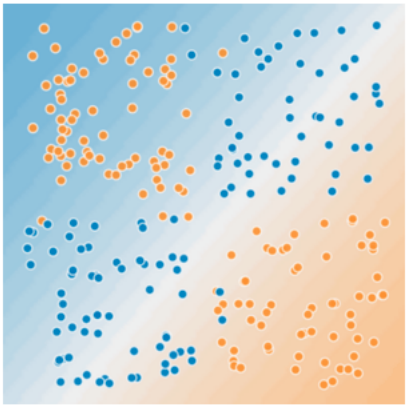


Figure 11: Figure showing a non linear model

10 Regularisation for Simplicity

As you can see in **Figure 12** as you increase the iterations, the training loss decreases however the validation loss eventually begins to increase. This is because the model overfits to the given data. Up till now we have only been trying to decrease the training loss, now we will explore how we can avoid overfitting using a principle called regularisation.

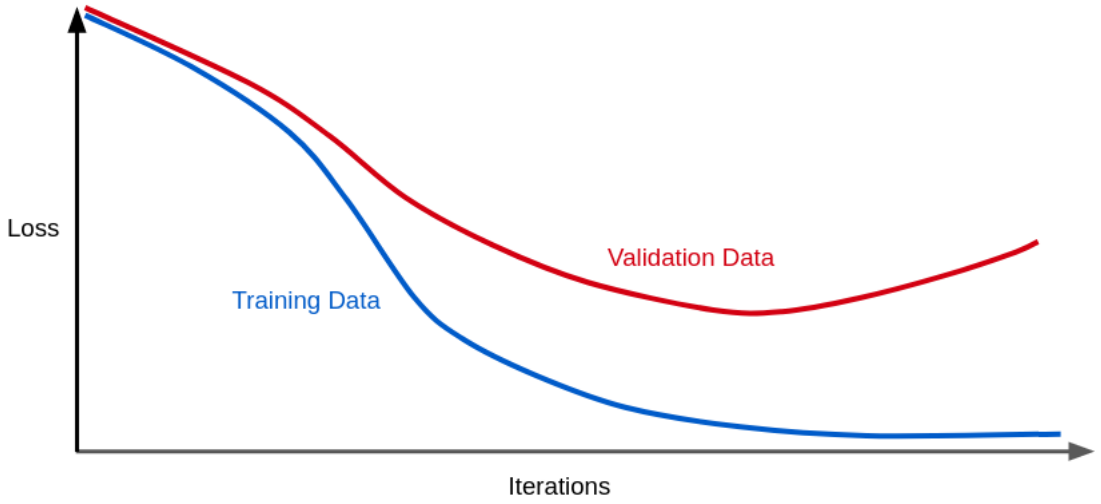


Figure 12: Figure showing the generalization curve

We will now minimize the loss + complextity, this is called structural risk minimization:

$$minimize(Loss(Data|Model))$$
$$\rightarrow$$
$$minimize(Loss(Data|Model)) + complexity(Model)$$

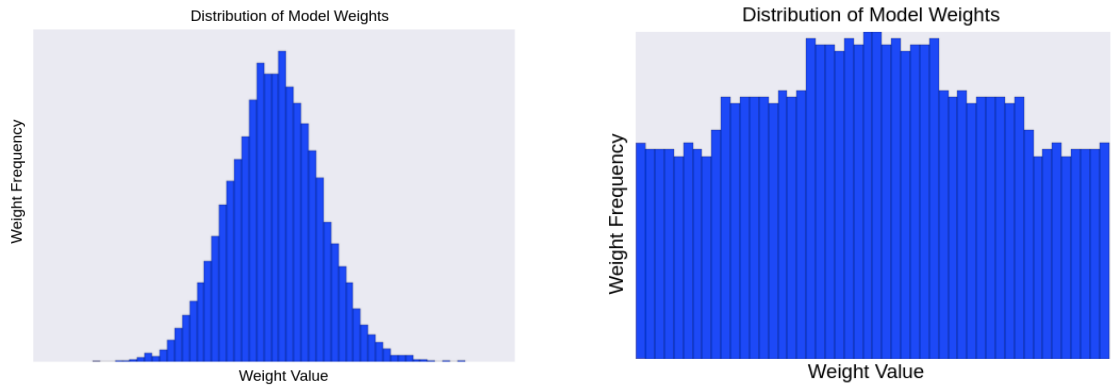
In this section, we will use the L_2 regularization, which quantifies the regularization term as the sum of squares of the feature wieghts.

$$L_2 \text{ regularization term} = w_1^2 + w_2^2 + .. + w_n^2$$

To further tune the impact of the regularization term. We multiply the L_2 value by a scalar, λ. Thus we now aim to:

$$minimize(Loss(Data|Model)) + \lambda complexity(Model)$$

In this way, the regularization ecourages weight values and the mean of the weights towards 0, with a Gaussian Distribution. Increasing the lambda value strengthens the regularization effect.



(a) Histogram of weights with a high lamda value.

(b) Histogram of weights with a low lamda va'Makefile' has lue.

Figure 13: Lowering the value of lambda tends to yield a flatter histogram

If the lambda value is:

- too high, the model would be simple however there is a greater risk of underfitting the data. Hence the model would not be able to learn much from the training data.
- too low, the model would be more complex however there is a greater risk of overfitting the data. This means it will learn too much about the particularities of the training data set.

11 Logistic Regression

There a two ways of calculating probability, "As is" and Converted to a binary category. The "As is" approach is self-explanatory. However, in classification cases we use a **sigmoid function**, which produces an output which always falls between 0 to 1. The sigmoid function is as follows:

$$y' = \frac{1}{1+e^{-z}}$$

Where

- y' is the output of the logistic regression model for a particular example
- $z = b + w_1x_1 + w_2x_2 + .. + w_Nx_N$

11.1 Loss function for Logistic Regression

The loss function for linear regression is squared loss. The loss function for logistic regression is Log Loss:

$$LogLoss = \Sigma_{(x,y)\in D} -ylog(y') - (1-y)log(1-y')$$

Where

- $(x, y) \in D$ is the dataset containing many labelled examples.
- y is the label in the label example. y is 0 or 1 as this is a logistic regression.
- y' is the predicted value, which is between 0 and 1.

12 Classification

12.1 Thresholding

Logistic regression returns a probability. We use the probability to predict how likely something is e.g. if an email was spam. We create a classification threshold to do the obvious, if its above a value then it is spam and if below it is not.

12.2 True vs. False, Positve vs. Negative

When predicting scenarios, there are four situations:

- True positive - when the model correctly predicts the positive class.
- True negative - when the model correctly predicts the negative class.
- False positive - when the model incorrectly predicts the positive class.
- False negative - when the model incorrectly predicts the negative class.

12.3 Accuracy, Precision and Recall

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predicitons}$$

$$Precision = \frac{Number\ of\ True\ Positives}{Total\ number\ of\ Positive\ Predicitons}$$

$$Recall = \frac{Number\ of\ True\ Positives}{Number\ of\ True\ Positives\ Predicitons\ and\ Number\ of\ False\ Negatives}$$

12.4 ROC and AUC

ROC curve (Receiver Operating Characteristic Curve) is a graph that shows the performance of a classifcation model, it has two parameters: True Positive Rate and False Positive Rate.

$$TPR = \frac{TP}{TP+FN}$$
$$FPR = \frac{FP}{FP+TN}$$

The ROC pots TPR vs FPR at various classification decision thresholds.
AUC is Area Under the ROC Curve. It provides a total performance measure over all classificatoin thresholds.

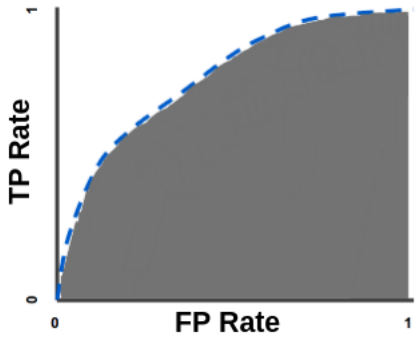


Figure 14: Area Under the ROC Curve

AUC can be interpreted as the probability that the model ranks a random positive example more highly than a random negative example. Therefore, the closer the AUC to 1.0, the better as an AUC value of 1.0 means the predictions are 100% correct.

12.5 Prediction Bias

Logistic regression predictions should be unbiased and this means:

$$average\ of\ predictions \approx average\ of\ observations$$

The prediction bias is a the measure of how far apart the two are:

$$prediction\ bias\ =\ average\ of\ predictions - average\ of\ observations$$

There are various causes of a prediction bias; Incomplete data set, Noisy data set, Buggy pipeline, Biased training sample or Over strong regularization. In order to correct the prediction bias by adding calibration layer. However, this could be problematic as you could be fixing symptoms rather than the cause or the system is brittle and you would need to constantly keep the model up-to-date.

As the logistic regression model predicts a value between 0 and 1 however all labelled exapmles are either 0 or 1 therefore we can use "bucketing" to mitigate this issue. The are two ways to form buckets, linearly breaking up the target predictions or forming quantiles.
Consider the following calibration plot from a particular model. Each dot represents a bucket of 1,000 values.

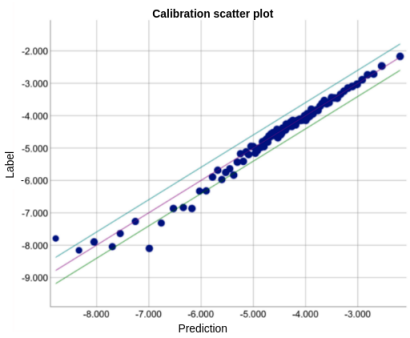


Figure 15: Area Under the ROC Curve

13 Regularization: Sparsity

Sparse vectors often contains many dimensions and creating a feature cross, creates even more dimensions. However, having numerous dimension features means when training the model, it becomes large and requires a lot of RAM.
In order to optimize the model we use regularization, however L_2 would not fix the issue as it does not force the weights to 0.0.
An apporoach we could take is L_0 regularization which is when chosen weights are reduced to 0 in order to be able to get the model to fit the data. However, this would turn our convex optimization problem into a non-convex optimization problem.
An alternate is called L_1 regularization which means to make the noisy uninformative coefficients to 0 in order to create an approxi mate to L_0 which saves a lot of RAM.

14 Neural Networks

A "non-linear" classification problem means that we can't accurately predict a label which has a model in the form of $b + w_1x_1 + w_2x_2$. For example:

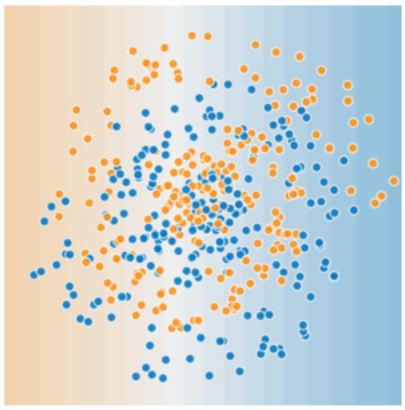


Figure 16: A Complex Nonlinear model

As you can see, we can't predict the model shown in **Figure 16** with a linear model. To begin understanding neural networks, let's visualize a linear model:

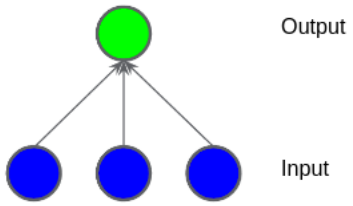


Figure 17: A Linear model as a graph

Each blue node represents an input feature and the green node which represents the weighted sum of the inputs.

14.1 Hidden Layers

We have now added a second layer to the model. The yellow nodes are the hidden layer which compute the weighted sum of the blue nodes and the green node computes the weighted sum of the yellow nodes. This model is still linear as it produces one output.

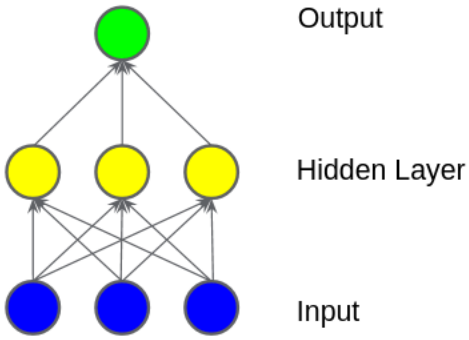


Figure 18: Graph if a two-layer model

14.2 Activation Functions

To model a nonlinear problem, we can introduce nonlinearity. In the model below we see that Hidden Layer 1 is transformed by a nonlinear function before computing the weighted sum for the next layer. This nonlinear function is called the activation function.

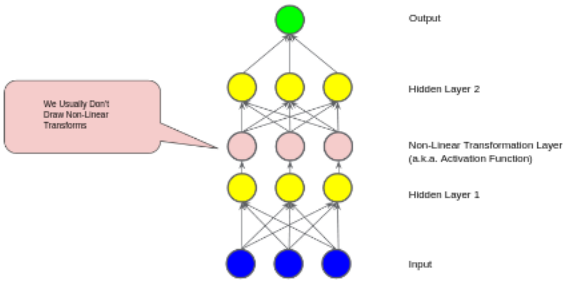


Figure 19: Three-layer model with activation function

In this way we add more impact as stacking nonlinearities lets us model more complicated relationships.

14.3 Common Activation Functions

The sigmoid function converts the weighted sum to a value between 0 and 1.

$$F(x) = \frac{1}{1+e^{-x}}$$

The **rectified linear unit** activation function (ReLU) works slightly better than the sigmoid "smooth" function.

$$F(x) = max(0, x)$$

However, any mathematical function can be an activation function.

$$\text{value of a node in the network} = \sigma(w \cdot x + b)$$

Where σ represents our activation function e.g. sigmoid or ReLU.

15 Best Practices when training neural networks

15.1 Failure Cases

There are numerous ways backpropagation can go wrong.

15.1.1 Vanishing Gradients

Gradients for the layers closer to the input can be very small and when working with deep networks, the gradients could vanish towards 0.
The ReLU function can help mitigate this potential issue.

15.1.2 Exploding Gradients

If the weights are very large, the gradient of the lower layers involves products of many large terms. This overall means that the gradents become to large that they begin to converge.
We can use batch normalizations to prevent exploding gradients.

15.1.3 Dead ReLU Units

When the weighted sum for a ReLU falls below 0, the ReLU unit may get stuck. It then outputs 0 activation and thus the gradients become cut off. To avoid this we can lower the learning rate.

15.2 Dropout Regularization

Another type of regularization is Dropout regularization. It works by randomly dropping unit activations in a network for a single gradient step. The more you drop, the stronger the regularization. 0.0 = No dropout regularization and 1.0 = Drop out everything which means model learns nothing.