# **Model Selection**

The central issue in all of the machine learning is “how do we extrapolate learnings from a finite amount of available data to all possible inputs ‘of the same kind’?” Training data is always finite, yet the model is supposed to learn everything about the task at hand from it and perform well on unseen data.

Take the car pricing dataset in linear regression, for example, which was trained using a few thousand observations. How do you ensure, and be confident, that the model is as good as it seems on the training data and deploy it to make predictions on real, unseen data?

 Often, it is mistaken that if a model performs well on the training data, it will produce good results on test data as well. Very often, that is not the case.

Central Issue in Machine Learning

The central issue in machine learning can be said to be the study of

How to extrapolate learnings from a finite amount of data to explain or predict all possible inputs of the same kind

**Feedback :*Machine learning*** *does not simply involve building models to fit the available data. The real challenge is to learn patterns which can be used to explain the behaviour of similar unseen data.*

Occam's Razor

Occam’s razor is a fundamental principle which suggests that:

Top of Form



A model should be simplified as much as possible



A model should be as simple as possible, but robust

**Feedback :*Occam’s razor*** *does not say that a model should be unjustly simplified till no further simplification is possible. It says that when faced with a trade-off between a complex and a simple model, with all other things being roughly equal, you are better off choosing the simpler one. The reason for this will be explained shortly.*

Bottom of Form

Regression Models

Choose the simplest regression model among the following (all lowercase alphabets are features):

Top of Form



Y = 3x + 0.005z + w



Y = x + 58z + log(w)



Y = x + 3w

Correct



Y = 2x + exp(w)

Bottom of Form

**Occam's razor** is perhaps the most important thumb rule in machine learning, and incredibly 'simple' at the same time. When in dilemma, choose the simpler model. The question then is 'how do we define simplicity?'

Let's understand 2 concepts and come back to Occam's Razor.

* Underfitting -  We need to ensure that training error as small as possible.
* Overfitting - Training scores are high but Test scores are very low. Model has memorized the training data.

Now while building model we have to ensure that we build optimal model so that model is not under or over trainings else it will not give right predictions. If there are more than 1 suitable models then we should select the simplest one as per Occam's Razor. Idea is to avoid unnecessary complexity in the model.

|  |  |
| --- | --- |
|  | This could be one linear model. Another example we can take where say a model is logical, like below    A model is something which takes some input and produces some expected output based on the underlying processing. It also expects to perform same with future unseen unknown inputs as it performed previously with training data inputs. |

|  |  |
| --- | --- |
|  | Say these two model, to represent the models we use coefficients. In first case, it’s a floating point representations where probably we need 2 float needs 32bit memory where as the other one just need 4bits. So the later one is much more simpler. |

# Model and Learning Algorithm

Before you dive into what exactly a simple model is, what all its benefits are, we will take a short detour to reiterate some terminologies and the machine learning framework. You will now understand the process of using training data, learning from it and then building a model to describe a system which performs a task at hand, like classification or regression. The key objectives here are to understand:

The meaning of model, learning algorithm, system and hypothesis class

The (often misunderstood) difference between a learning algorithm and a model

The meaning of ‘class of models’

|  |  |
| --- | --- |
|  | Here, the learning algorithm knows given a system say ‘x’, it knows ‘what’ to produce which is ‘v’. And the outcome of this learning algorithm is the model. Once the model is in place we can typically remove the system, as the model already learned from the learning algoritm and now it knows ‘how’ to produce similar output from any such ‘x’. |

a basic property of a learning algorithm - that it can only produce models of a certain kind within its boundaries. This means that an algorithm designed to produce linear class of models, like linear / logistic regression, will never be able to produce a decision tree or a neural network. The class of model becomes critical because a wrong class will yield a sub-optimal model.

**Note**: A detailed discussion on the linearity of SVMs will be done in a later module. For now, remember that all SVMs (with any kernel or type of problem) is fundamentally a linear problem.

**The relationship between a model and a learning algorithm is that:** A learning algorithm learns from training data and produces a model

**Feedback :*Learning algorithm's*** *task is to figure out what needs to be done and how. In linear regression, for example, the learning algorithm optimises the cost function and produces the models, i.e. the coefficients.*

Linear Class of Models

A learning algorithm belonging to the 'linear' class will only produce linear models. Which of the following models can NOT be produced by such a learning algorithm?

Top of Form



Linear regression



Logistic Regression

Feedback :*Logistic regression is a linear model. Recall that log(odds) is equal to a linear expression in the features. By the way, the full form of glm() is generalised linear models, remember?*

Incorrect



k-NN

Feedback :*Logistic regression is a linear model. Recall that log(odds) is equal to a linear expression in the features. The full form of glm() is generalised linear models. K-NN is non-linear (Food for thought: Is it even a model or just a learning algorithm?)*

Bottom of Form

|  |  |
| --- | --- |
|  | Every learning algorithm puts a boundary around the kinds of model it is going to ever consider and among those models it will try to find the best which fits the data it has been given for training. That model is going to come out as output from the learning algorithm.  A linear model like linear, logistics or svm will never do best fit for this example data points. This is called Hypothesis class. |

# Simplicity, Complexity and Overfitting

Various class of models includes

1. High dimensional data
2. Noisy data
3. Real time data

For example say with some data set we have build two models one with Logistic regression and another one with SVM. Now the question is which one should I choose ? How do I make that decision ?

First gospel rule in model building is

* Do not use the training data for model evaluation

Below see the data points is like a curve where we fit a liner model having error terms > 0. In Linear model error never will be zero it will always be some error, so that’s why we minimize the error terms by means of R2 or some other technique

|  |  |
| --- | --- |
|  |  |

From your school or college, you can probably recall those few fellows who seemed to study less but understood much more than others. They seem to never care about memorizing or mechanically practicing what was being taught, yet are able to explain complex problems in physics or mathematics with simplicity and elegance.

Assuming that people learn using ‘mental models’, do these students have remarkably different mental models than those who solve a bunch of books and focus on memorization? How can they learn so much from a finite amount of information and apply that to solve unseen, complex problems?

In this segment, Prof. Raghavan will explain the meaning of model simplicity, complexity as well as the pros and cons associated with them. As a by-product, you will also understand that the best way to ‘learn’ is ‘to keep your mental models simple’.

**Unique points about using a simpler model where ever possible:**

1. A simpler model is usually more generic than a complex model. This becomes important because generic models are bound to perform better on unseen datasets.
2. A simpler model requires less training data points. This becomes extremely important because in many cases one has to work with limited data points.
3. A simple model is more robust and does not change significantly if the training data points undergo small changes.
4. A simple model may make more errors in the training phase but it is bound to outperform complex models when it sees new data. This happens because of **overfitting**.

|  |  |
| --- | --- |
|  |  |

Here the straight-line model probably the best model in terms of generalizing the 21,22,23 data points well compared to polynomial or degree-15 fit complex models even though those are passing through most of the data points making error terms almost 0.

The reason simpler models are considered better than complex models are

1. Simpler models are generic i.e. they apply to a wider range of data
2. Simpler models require less training data than complex models
3. Simpler models are more robust

# Overfitting

Overfitting is a phenomenon where a model becomes too specific to the data it is trained on and fails to generalise to other unseen data points in the larger domain. A model that has become too specific to a training dataset has actually ‘learnt’ not just the hidden patterns in the data but also the noise and the inconsistencies in the data. In a typical case of overfitting, the model performs very well on the training data but fails miserably on the test data.

Models are trained on a set of training data but their efficacy is determined by the ability to perform well on unseen (test) data

**Feedback :***It is possible to memorize the training data while failing to truly learn the underlying trends and patterns. On unseen data (read tricky but unseen exam questions), memorizing is bound to fail.*

**Overfitting in Linear Regression**

In linear regression, which of the following are clear signs of overfitting?

The R-squared value on training data is 0.90 and 0.30 on train and test data respectively

**Feedback :***In overfitting, the model fits the training data very well since it has somehow memorised it.*

**Identifying Overfitting in a Neural Network**

Say you have a 100,000 Google images as training observations and you are trying to build a neural network to classify the images in 3 classes - nature, cities and others.  You use another 50,000 observations to test it and the accuracy on the test set comes out to be 10%. Which of the following can you use to check whether the model has overfitted?

Accuracy on the training set

**Feedback :***If the training accuracy is way higher than 10%, it is likely to have overfitted. Neural networks, as you’ll study later, can be made extremely complex using a number of hyperparameters like hidden layers and number of neurons. The basic idea, though, remains the same - memorisation of training and and failure on test data.*

# Bias-Variance Tradeoff

So far, we have discussed the pros and cons of simple and complex models. On one hand, simplicity is generalizable and robust and on the other hand, some problems are inherently complex in nature. There is a trade-off between the two, which is known as the bias-variance tradeoff in machine learning.

* Variance in the model is how sensitive is the model towards the input data aka training data
* Bias is how much error terms the model is likely make in test data

Bias and Variance

We said that the first person's mental model has high variance and that the second one's has high bias. Relating model complexity to bias and variance, we can say that:

* Complex models have high variance and simple ones have high bias

**Feedback :***The first person's model is complex, since he/she needs to memorise a lot, and is therefore prone to change when the exam pattern or syllabus (read training data) changes. The second one's model is simple and may be too simple to crack a competitive exam, hence high bias (read 'naive model').*

**Total Error of a Model**

Imagine a linear regression model, say car pricing dataset, where you consider transforming the features to log(x), sqrt(x) etc. Complexity here will be proportional to the number of coefficients, the magnitude of coefficients, degree of polynomial (x squared is a more complex feature than x) etc. In principle, you can take as many features as you like and transform them in complex ways to 'make a model which fits perfectly'. In short, you can increase complexity to reduce the error to zero. In the figure, however, the total error goes up (after the minima) as model compelxity increases. Why is that so?

* By increasing complexity, I can reduce the training error, whereas the figure plots the test error

**Feedback :***It is crucial to remember that we always talk about the test error (i.e. the expected error on a similar, unseen dataset) while discussing bias-variance tradeoff. If training error were concerned, we could just increase complexity to make the model pass through all the points.*

**Correct**



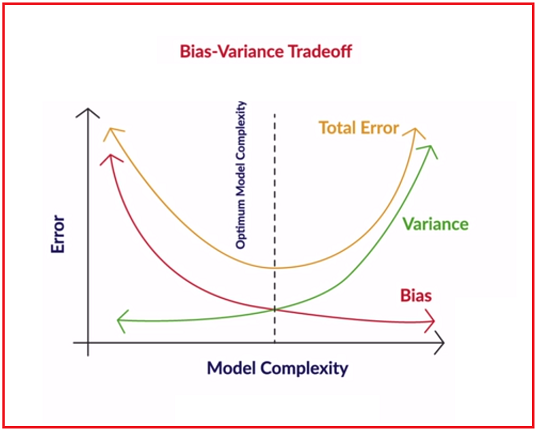
We considered the example of a model memorizing the entire training dataset. If you change the dataset a little, this model will need to change drastically. The model is, therefore, **unstable and sensitive to changes in training data**, and this is called **high variance**.

The ‘variance’ of a model is the **variance in its output** on some test data with respect to the changes in the training data. In other words, variance here refers to the **degree of changes in the model itself** with respect to changes in training data.

**Bias** quantifies how **accurate the model is likely to be**on future (test) data. Extremely simple models are likely to fail in predicting complex real world phenomena. Simplicity has its own disadvantages.

Imagine solving digital image processing problems using simple linear regression when much more complex models like neural networks are typically successful in these problems. We say that the linear model has a high bias since it is way too simple to be able to learn the complexity involved in the task.

In an ideal case, we want to reduce both the bias and the variance, because the expected total error of a model is the sum of the errors in bias and the variance, as shown in the figure below.



Although, in practice, we often cannot have a low bias and low variance model. As the model complexity goes up, the bias reduces while the variance increases, hence the trade-off.

# Regularization

Having established that we need to find the right balance between model bias and variance, or simplicity and complexity, we need tools which can reduce or increase the complexity. In this segment, we will learn regularization methods which are used to keep an eye on model complexity.

Regularization is the process of deliberately simplifying models to achieve the correct balance between keeping the model simple and yet not too naive. Recall that there are a few objective ways of measuring simplicity - choice of simpler functions, lesser number of model parameters, using lower degree polynomials, etc.

* To prevent the model from becoming complex
* It is part of the learning algorithm

# Summary

In this session, you learnt the most fundamental principles of machine learning which you should now be able to apply while building models. The most important points to re-iterate are:

**Occam's Razor**

* A model should be as simple as necessary, but no simpler
* When in doubt, choose a simpler model
* Advantages of simplicity are generalisability, robustness, making few assumptions and less data required for learning

**Bias-Variance Tradeoff**

* Bias measures how accurately a model can describe the actual task at hand
* Variance measures how flexible the model is with respect to changes in the training data
* As complexity increases, bias reduces and variance increases, and we aim to find the optimal point where the total model error is the least

**Overfitting**

* A model memorizes the data rather than intelligently learning the underlying trends in it
* It arises because it is possible to memorize data, and this is a problem, because the real test happens on unseen, real world data

Model Evaluation

# Regularization and Hyperparameters

**Regularization** discourages the model from becoming too complex even if the model explains the (training) observations better. In the last session, you were introduced to this term which is used to find the optimal point between extreme complexity and simplicity. In this context, we will now discuss the use of hyperparameters of a model.

**Hyperparameters** are parameters that we pass on to the learning algorithm to control the complexity of the final model. Hyper parameter are choices that the algorithm designer makes to ‘tune’ the behavior of the learning algorithm. The choice of hyperparameters, therefore, has a lot of bearing on the final model produced by the learning algorithm.  Hyperparameters are a part of most learning algorithms which are used for training and regularization. In linear regression, as you will now see, the hyperparameter is used to regularize the model so that it does not become more complex than it should be.

To summarize the concept of hyperparameters:

* Hyperparameters are used to 'fine-tune' or regularize the model so as to keep it optimally complex
* The learning algorithm is given the hyperparameters as an 'input' and returns the model parameters as the output
* Hyperparameters are not a part of the final model output

|  |  |
| --- | --- |
|  | The choice of lambda is very important. If I want to keep the error term low the use lower lamda value. If I want to use higher regularization then use high lamda value.  In Decision Tree algorithm hyperparameter controls how many layer depth I would go or in Nural Network algorithm it controls how many layers of Nuron’s will be there in my model  The output of the model parameters are the Hyperparameters where as the input to the model parameters are coefficients (for linear regression) like a,b in this example |

|  |  |
| --- | --- |
| Fig 1 | Fig 2 |

In Fig 1-, typically in machine learning model building we must keep some data set away from the training or input set so that ML algorithm cannot sneak the test data. Every iteration we tune the hyperparameters and re-run the ML algorithm but doing that many times the algorithm become smart enough to sneak into the test data sometimes for the model. Later when we run the algo against the test data it always tends to produce high accuracy output.

To avoid this keep aside a data set called validation set (Fig -2) from the input data. Now to tune the model we do the tuning against the hyperparameters from the validation set and re-run the ML algorithm against that to see how good/bad is the model. Later once we satisfied the model is good, we run the model against test data or give the model to clients to run against their test data set. This way ML algorithm does not know anything about test data.

# Model Evaluation and Cross Validation

The key thing to remember is that a model should never be evaluated on data it has already seen before. With that in mind, you will have either one of two cases - 1) The training data is abundant and 2) The training data is limited.

The first case is straightforward because you can use as many observations as you like to both train and test the model. In the second case, however, you will need to find some ‘hack’ so that the model can be evaluated on unseen data and at the same time doesn’t eat up the data available for training. This hack is called **cross-validation**.

|  |  |
| --- | --- |
|  | Here, we build 6 models but none of the model trained on entire data set. Initially, we had split the entire input data sat in 6 sub-sets and build the model using those sets. Then we can run the model with respective test data. Almost every ML algorithm does this in practice.  The Cross-Validation technique is extremely popular in industry. |

﻿The following figure illustrates k-fold cross-validation with k=4. There are some other schemes to divide the training set, we'll look at them briefly later.

|  |  |
| --- | --- |
| https://cdn.upgrad.com/UpGrad/temp/ec2a2d98-ac54-450b-8a62-327b863fb7e7/cv.png | In **K-fold CV**, you divide the training data into K-groups of samples. If K=4 (say), you use K-1 folds to build the model and test the model on Kth fold. |

**Python Demo – Check the notebook ML-2/ModeSelection/CrossValidationLinearRegression**

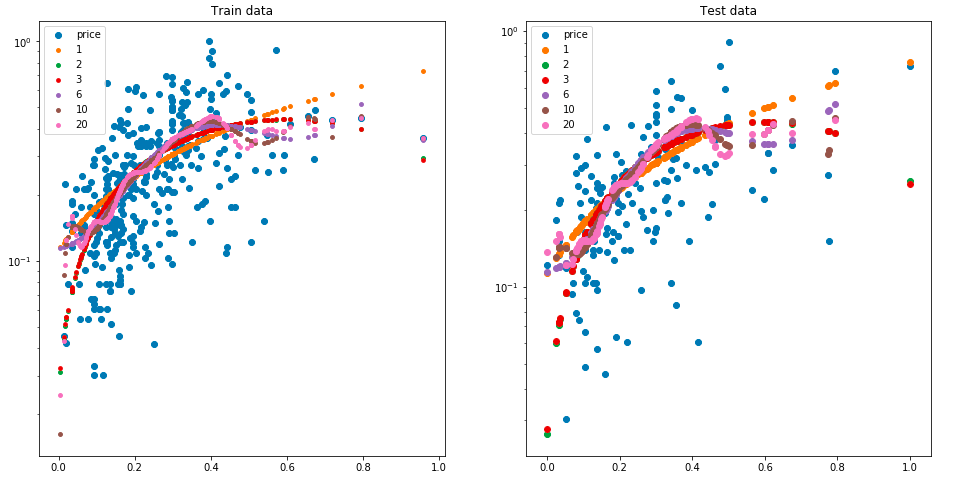
For linear regression of degree 1, you fit the curve of the form:

y=β0+β1x1

For polynomial regression of degree n, you fit the curve of the form:

y=β0+β1x1+β2x21+β3x31+....+βnxn1

In this way, instead of just fitting a straight line to a single input-feature dataset, you can fit a more complex curve to the training data. How does fitting a more complex curve effect the accuracy on the test set? As you know, this can result in overfitting which can lead to low test accuracy. Recall that you populated the following matrix with the outputs obtained by the linear regression models of various polynomical degrees.



In the graphs shown above, the blue points represent the train and test data points respectively - train on left-side and test on the right-side. The predictions for various polynomial degrees are shown in different colours. From the plots above, you can see the curves start to overfit the data as the degree of the polynomial is increased beyond a certain degree. This is confirmed by the value of R-squared for both test and train data:

|  |  |
| --- | --- |
| https://cdn.upgrad.com/UpGrad/temp/a0a576eb-3264-4bc1-81da-fb57de8583f6/rsquare.PNG | **Model Results**  As you can observe, the training score is increasing, while the test score is going down as the degree of the polynomial is increased. This is a clear sign of overfitting. |

﻿

1. **K-Fold** cross-validation: Most common
2. **Leave One Out (LOO)**: Takes each data point as the 'test sample' once, and trains the model on the rest n-1 data points. Thus, it trains n total models.
   * Advantage: Utilises the data well since each model is trained on n-1 samples
   * Disadvantage: Computationally expensive
3. **Leave P-Out (LPO)**: Creat all possible splits after leaving p samples out. For n data points, there are (nCp) possibile train-test splits.
4. (**For classification problems**) **Stratified K-Fold**: Ensures that the relative class proportion is approximately preserved in each train and validation fold. Important when ther eis huge class imbalance (e.g. 98% good customers, 2% bad).

Additional Reading: The sklearn documentation enlists all CV schemes [here.](http://scikit-learn.org/stable/modules/cross_validation.html)

# Questions:

**Training**

Why should we have disjoint training and test datasets? i.e. model should not be tested with data it is trained on.

**Suggested Answer**

Any model has to be tested on how well it would work in the proverbial ‘real’ world. Because once a model has seen the data, it can pretty much attempt to ‘memorise’ it and once that is done, testing it on the same dataset will not help in determining its performance on unseen data. In an ideal scenario, when we have plenty of data, we should divide the data into three sets. First would be the training data. On which we shall train the model. . Second would be the validation data. On which we shall test the model and tune the hyperparameters. Third would be the test data, which we shall use for assessing our model.

**Variance**

Which of the following is more likely to have low variance?

**Weak Learner**

**Feedback :**

*Weak learners create simpler models which have lower variance. They are not able to model complex relationships and hence create a more generic model.*

**Correct**



Strong Learner

**Bias-Variance**

Consider that data is generated via a polynomial equation of degree 4 (i.e. the said polynomial equation will perfectly fit the given data). Which of the following statements are correct in this case?

Linear regression will have high bias and low variance

**Feedback :**

Linear regression would create a degree 1 polynomial which would be less complex than the degree four polynomial and hence would have a higher bias. Since the model is less complex and won’t overfit, it would have a low variance.

Polynomial equation of degree 4 will have low bias and Low variance

**Feedback :**

Since the equation fits the data perfectly, bias and variance will both be low here.

**Regularization**

Regularization is a:

Technique which is used to strike a balance between model complexity and model accuracy on training data.

**Feedback :**

*Regularization does not improve accuracy; it improves the balance between accuracy and complexity.*

**Simplicity of a model**

How would you quantify the simplicity of a model?

Top of Form



**Number of features used in the model**

**Feedback :**

If you use more features in your model, it would become more complex and might even overfit with given data. An overfitted model is not simple enough for prediction purposes.

**Correct**



**Number of nodes and depth of tree in case of a tree model**

**Feedback :**

The number of nodes and tree depth determine how complex the decision tree is. If a decision tree has more number of features, it will have more nodes and hence will be more complicated.

**Model Variance**

How do you measure the variance of a model?

By measuring how much does the estimates of the model change on the test data, on changing the training data.

**Feedback :**

*Variance measures how much the model changes with respect to the training data.*

**k-fold Cross Validation**

Which of the following statements is correct with respect to k-fold cross validation?

**Feedback :**

*Training happens k times and higher k would imply higher run time for training with k-fold cross validation. Also, a higher k implies that the training set every time is bigger and is a better representation of the actual data.*

Bottom of Form