Cross Validation

k-Fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows:

1. Shuffle the dataset randomly.

2. Split the dataset into k groups

3. For each unique group:

1. Take the group as a hold out or test data set

2. Take the remaining groups as a training data set

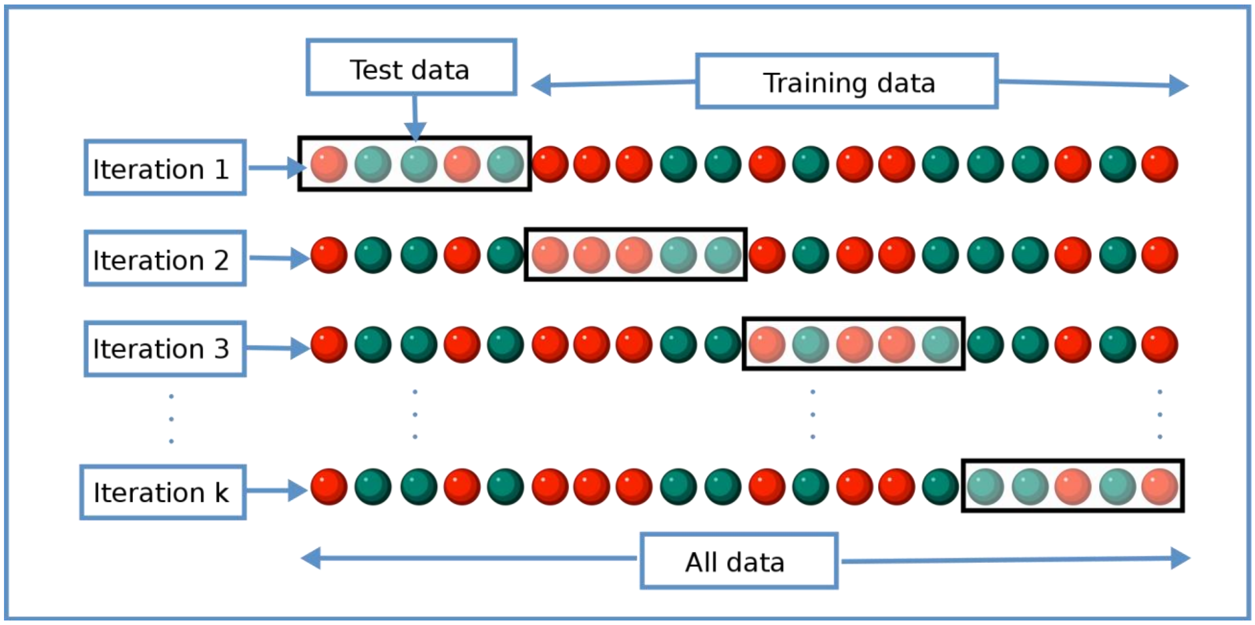
3. Fit a model on the training set and evaluate it on the test set

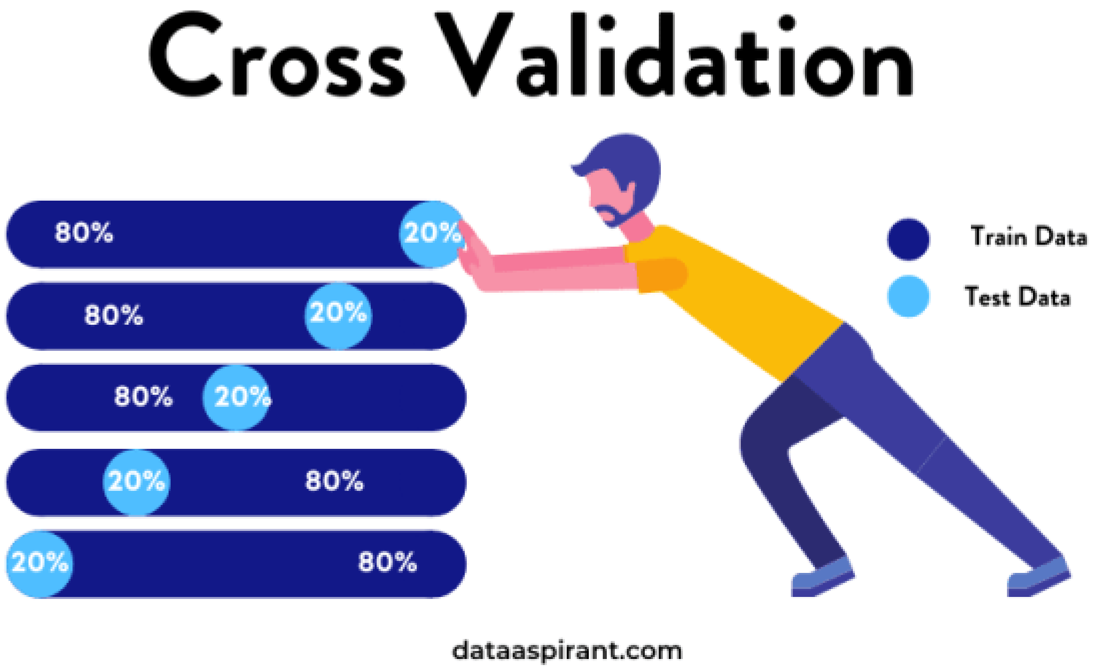
4. Retain the evaluation score and discard the model

4. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

The results of a k-fold cross-validation run are often summarized with the mean of the model skill scores. It is also good practice to include a measure of the variance of the skill scores, such as the standard deviation or standard error





Configuration of k

The k value must be chosen carefully for your data sample.

A poorly chosen value for k may result in a mis-representative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model).

Three common tactics for choosing a value for k are as follows:

- **Representative**: The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset.
- **k=10**: The value for k is fixed to 10, a value that has been found through experimentation to generally result in a model skill estimate with low bias a modest variance.
- **k=n**: The value for k is fixed to n, where n is the size of the dataset to give each test sample an opportunity to be used in the hold out dataset. This approach is called leave-one-out cross-validation.

Variations on Cross-Validation

There are a number of variations on the k-fold cross validation procedure.

Three commonly used variations are as follows:

- **Train/Test Split**: Taken to one extreme, k may be set to 2 (not 1) such that a single train/test split is created to evaluate the model.
- **LOOCV**: Taken to another extreme, k may be set to the total number of observations in the dataset such that each observation is given a chance to be the held out of the dataset. This is called leave-one-out cross-validation, or LOOCV for short.
- **Stratified**: The splitting of data into folds may be governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value, such as the class outcome value. This is called stratified cross-validation.
- **Repeated**: This is where the k-fold cross-validation procedure is repeated n times, where importantly, the data sample is shuffled prior to each repetition, which results in a different split of the sample.
- **Nested**: This is where k-fold cross-validation is performed within each fold of cross-validation, often to perform hyperparameter tuning during model evaluation. This is called nested cross-validation or double cross-validation.

Model overfitting

A model is said to be a good machine learning model if it generalizes any new input data from the problem domain in a proper way. This helps us to make predictions in the future data that the data model has never seen.

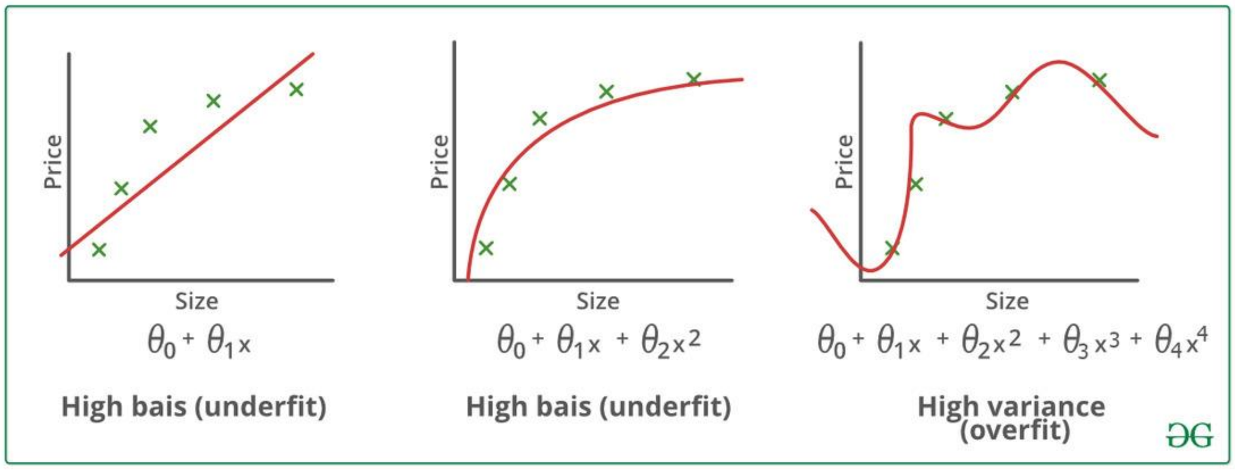
Now, suppose we want to check how well our machine learning model learns and generalizes to the new data. For that, we have overfitting and underfitting, which are majorly responsible for the poor performances of the machine learning algorithms.

Bias: Assumptions made by a model to make a function easier to learn.

Variance: If you train your data on training data and obtain a very low error, upon changing the data and then training the same previous model you experience a high error, this is variance.

Underfitting:

A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data. (It's just like trying to fit undersized pants!) Underfitting destroys the accuracy of our machine learning model.



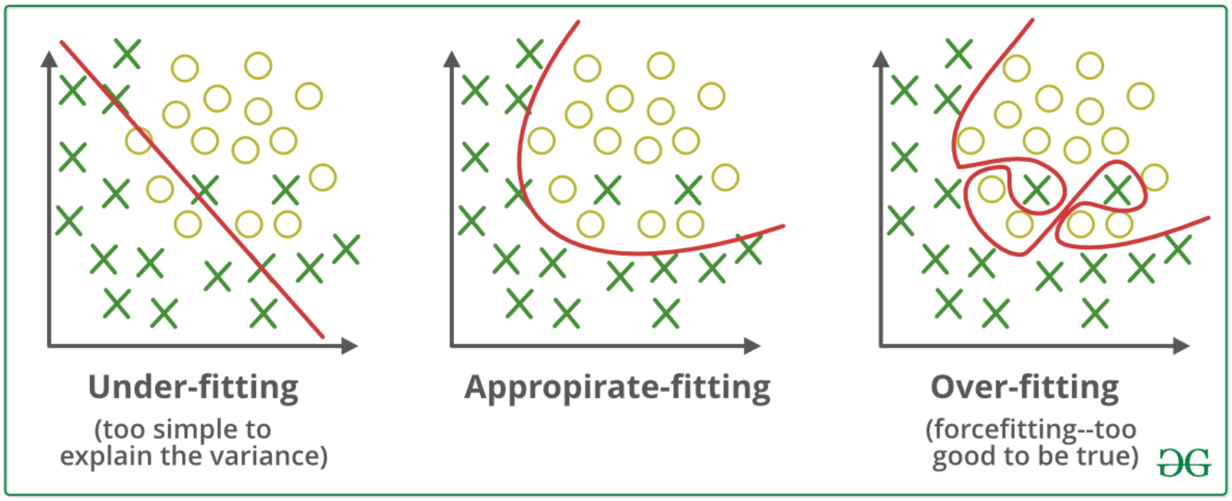
Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have fewer data to build an accurate model and also when we try to build a linear model with fewer non-linear data. In such cases, the rules of the machine learning model are too easy and flexible to be applied on such minimal data and therefore the model will probably make a lot of wrong predictions. Underfitting can be avoided by using more data and also reducing the features by feature selection. In a nutshell, Underfitting – High bias and low variance

Overfitting:

A statistical model is said to be overfitted when we train it with a lot of data (just like fitting ourselves in oversized pants!). When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in

our data set. Then the model does not categorize the data correctly, because of too many details and noise.

The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models.


Under-fitting
(too simple to explain the variance)
Appropirate-fitting
Over-fitting
(forcefitting--too good to be true)

A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees. In a nutshell, Overfitting – High variance and low bias

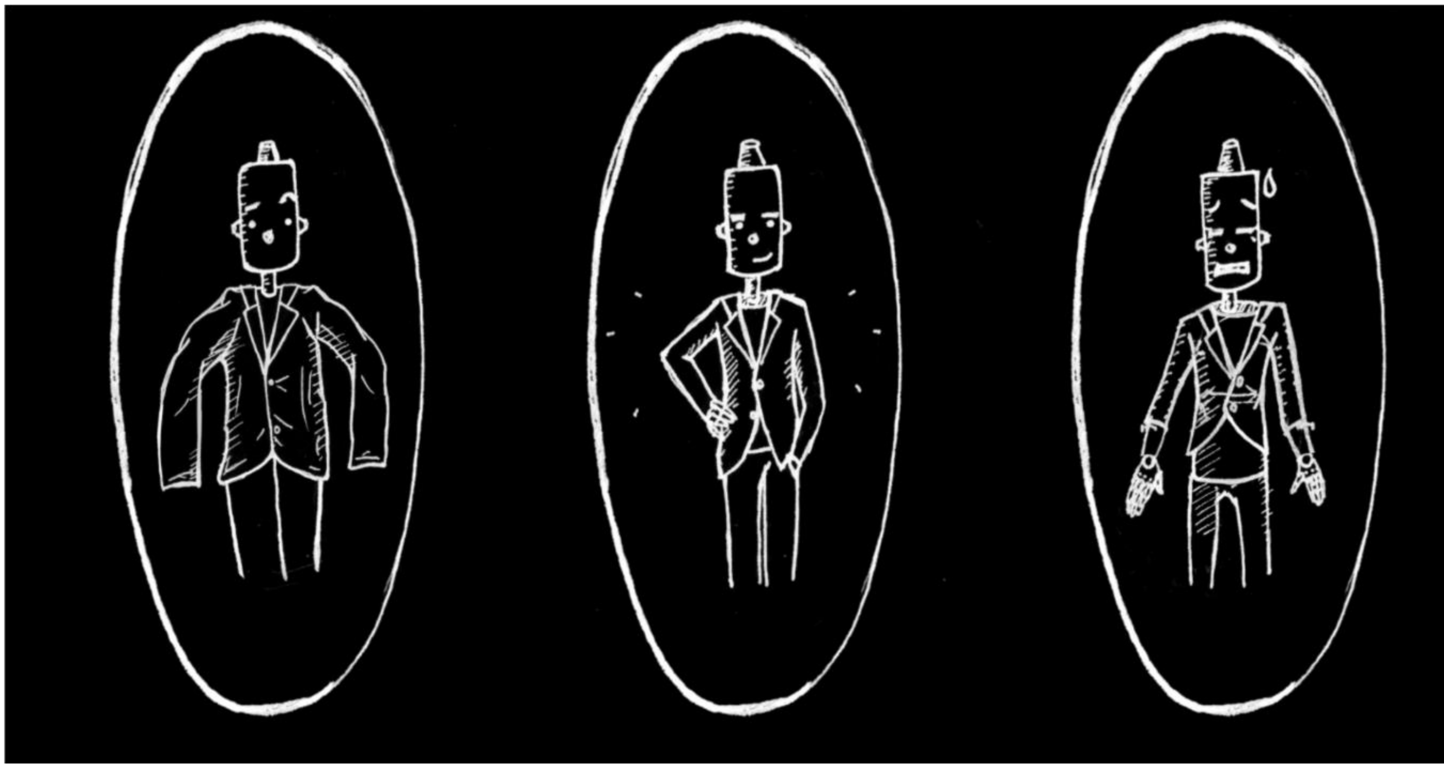
Ideally, the case when the model makes the predictions with 0 error, is said to have a good fit on the data. This situation is achievable at a spot between overfitting and underfitting. In order to understand it, we will have to look at

the performance of our model with the passage of time, while it is learning from training dataset.

With the passage of time, our model will keep on learning and thus the error for the model on the training and testing data will keep on decreasing. If it will learn for too long, the model will become more prone to overfitting due to the presence of noise and less useful details.

Hence the performance of our model will decrease. In order to get a good fit, we will stop at a point just before where the error starts increasing. At this point, the model is said to have good skills on training datasets as well as our unseen testing dataset

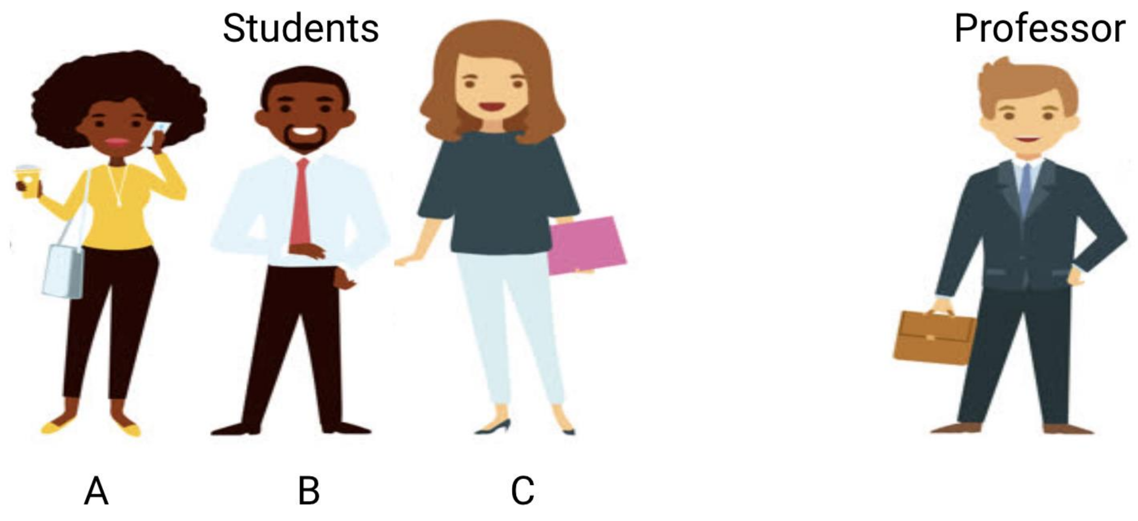
Underfitting vs. Overfitting (vs. Best Fitting) An Example



The Challenge of Underfitting and Overfitting in Machine Learning

Over fitting or under fitting happens very frequently whenever we are working with tree-based predictive models. Because of the way the algorithms work, we can imagine how tricky it is to avoid falling into the overfitting trap! Moreover, it can be quite daunting when we are unable to find the underlying reason why our predictive model is exhibiting this anomalous behavior.

Let’s Take an Example to Understand Underfitting vs. Overfitting



Consider a math class consisting of 3 students and a professor.

Now, in any classroom, we can broadly divide the students into 3 categories. We’ll talk about them one-by-one.

- Hobby = chating
- Not interested in class
- Doesn’t pay much attention to professor

Let’s say that student A resembles a student who does not like math. She is not interested in what is being taught in the class and therefore does not pay much attention to the professor and the content he is teaching.

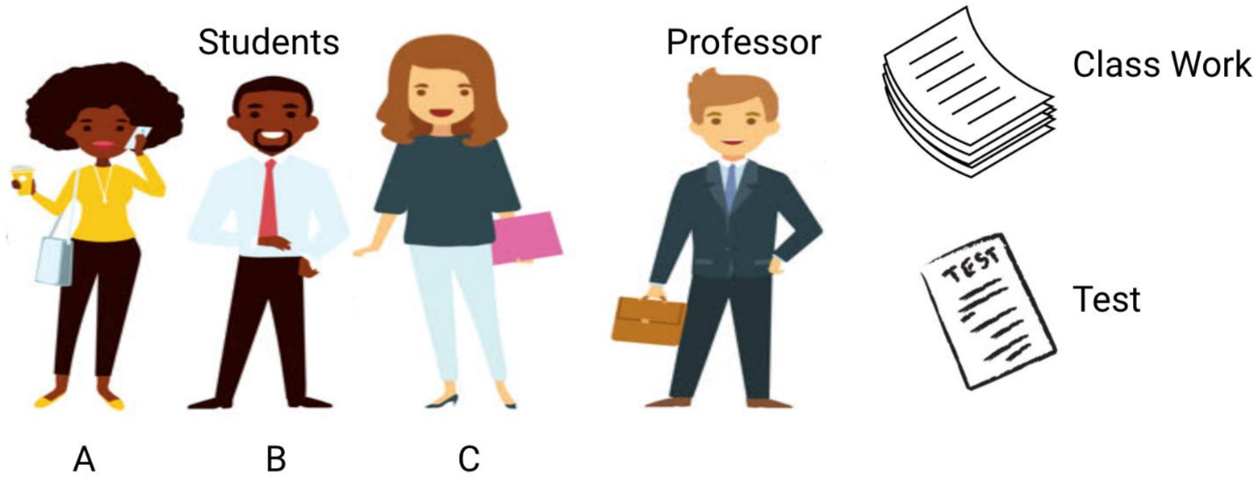
B

- Hobby = to be best in class.
- Mugs up everything professor says.
- Too much attention to the class work.

Let’s consider student B. He is the most competitive student who focuses on memorizing each and every question being taught in class instead of focusing on the key concepts. Basically, he isn’t interested in learning the problem-solving approach.

- Hobby = learning new things
- Eager to learn concepts.
- Pays attention to class and learns the idea behind solving a problem.

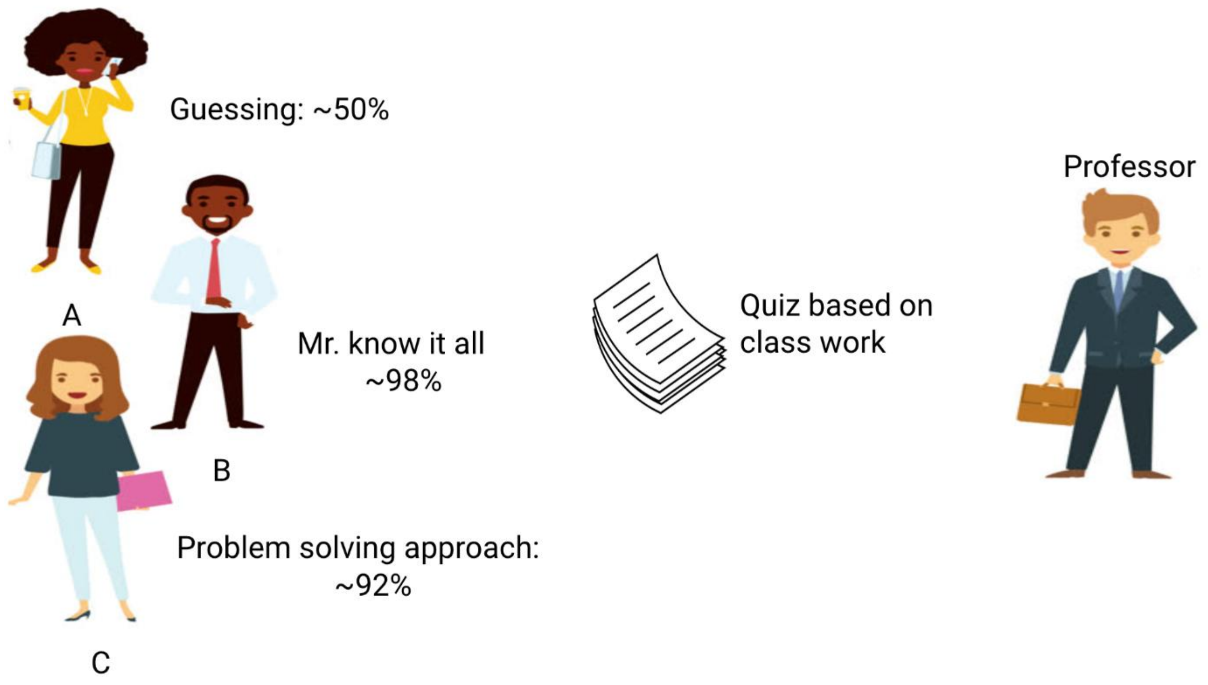
Finally, we have the ideal student C. She is purely interested in learning the key concepts and the problem-solving approach in the math class rather than just memorizing the solutions presented.



We all know from experience what happens in a classroom. The professor first delivers lectures and teaches the students about the problems and how to solve them. At the end of the day, the professor simply takes a quiz based on what he taught in the class.

The obstacle comes in the semester3 tests that the school lays down. This is where new questions (unseen data) comes up. The students haven’t seen these questions before and certainly haven’t solved them in the classroom. Sounds familiar?

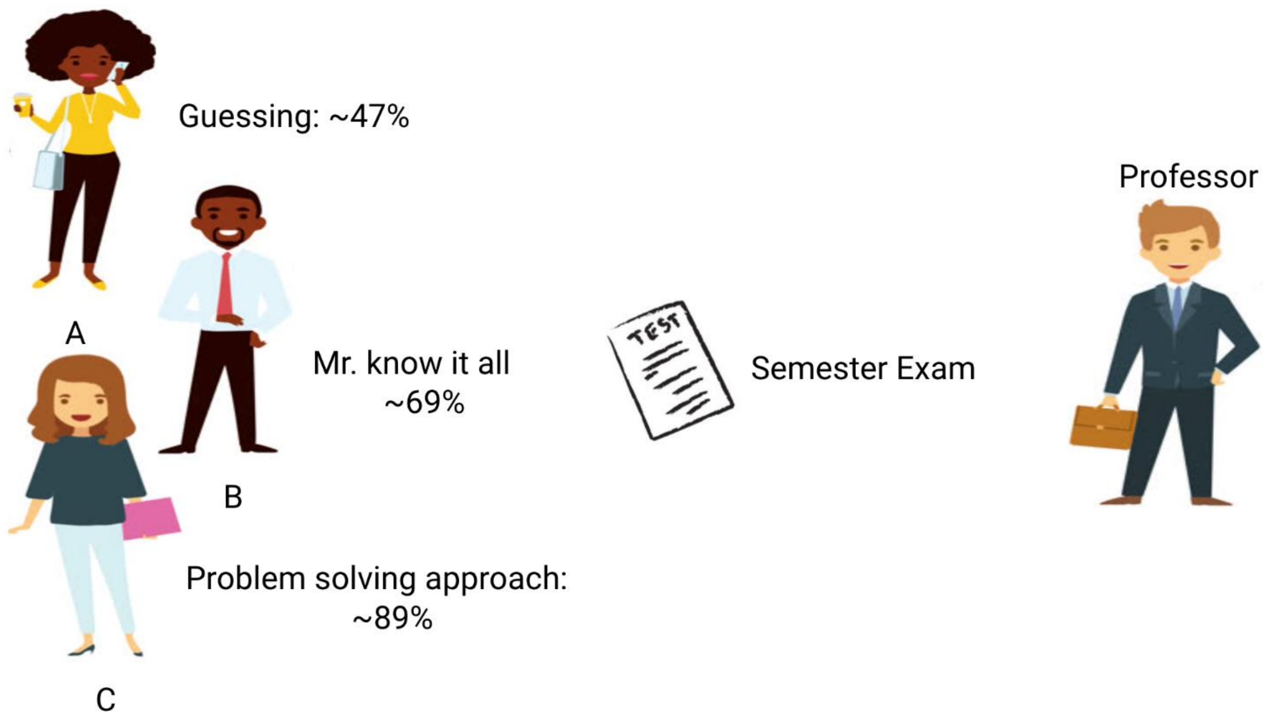
So, let’s discuss what happens when the teacher takes a classroom test at the end of the day:



- Student A, who was distracted in his own world, simply guessed the answers and got approximately 50% marks in the test
- On the other hand, the student who memorized each and every question taught in the classroom was able to answer almost every question by memory and therefore obtained 98% marks in the class test
- For student C, she actually solved all the questions using the problem-solving approach she learned in the classroom and scored 92%

We can clearly infer that the student who simply memorizes everything is scoring better without much difficulty.

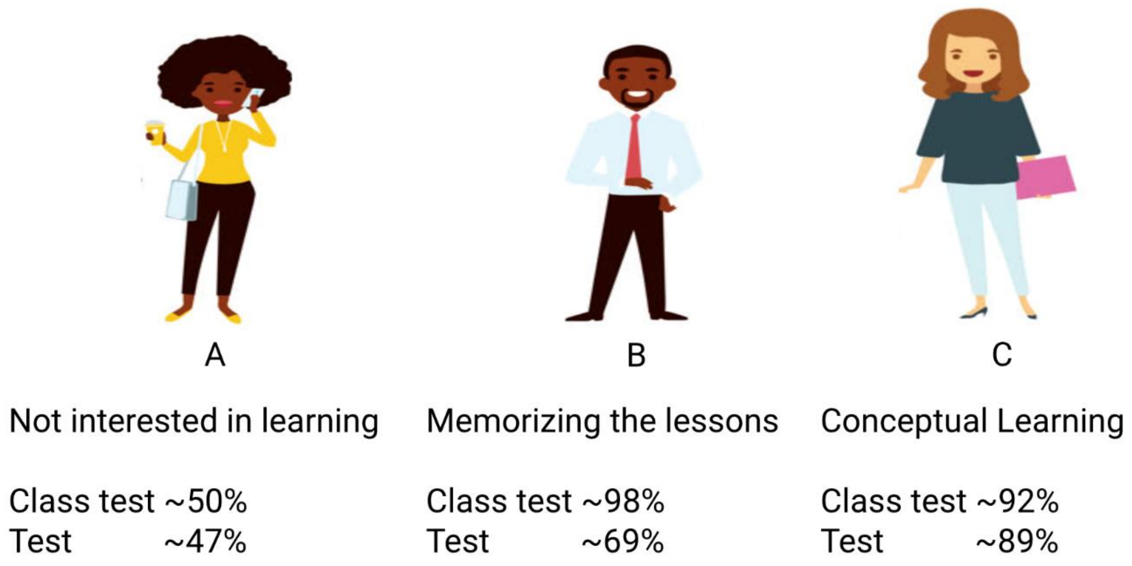
Now here’s the twist. Let’s also look at what happens during the monthly test, when students have to face new unknown questions which are not taught in the class by the teacher.



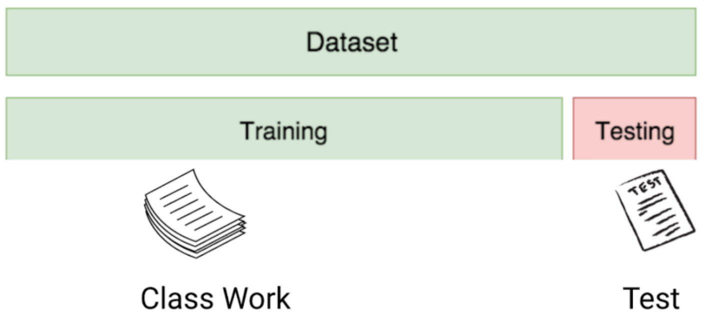
- In the case of student A, things did not change much and he still randomly answers questions correctly ~50% of the time.
- In the case of Student B, his score dropped significantly. Can you guess why? This is because he always memorized the problems that were taught in the class but this monthly test contained questions which he has never seen before. Therefore, his performance went down significantly
- In the case of Student C, the score remained more or less the same. This is because she focused on learning the problem-solving approach and therefore was able to apply the concepts she learned to solve the unknown questions

How Does this Relate to Underfitting and Overfitting in Machine Learning?

How this example relates to the problem which we encountered during the train and test scores of the decision tree classifier? Good question!



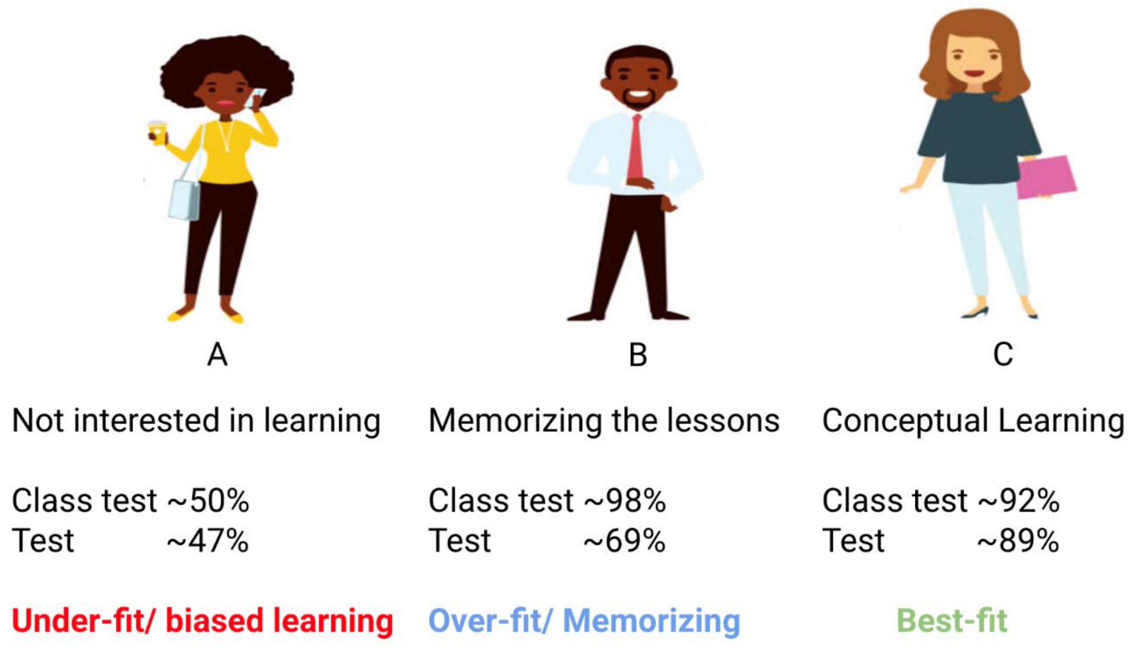
So, let's work on connecting this example with the results of the decision tree classifier that I showed you earlier.



First, the classwork and class test resemble the training data and the prediction over the training data itself respectively. On the other hand, the semester test represents the test set from our data which we keep aside before we train our model (or unseen data in a real-world machine learning project).

Now, recall our decision tree classifier mentioned earlier. It gave a perfect score over the training set but struggled with the test set. Comparing that to the student examples we just discussed, the classifier establishes an analogy with student B who tried to memorize each and every question in the training set.

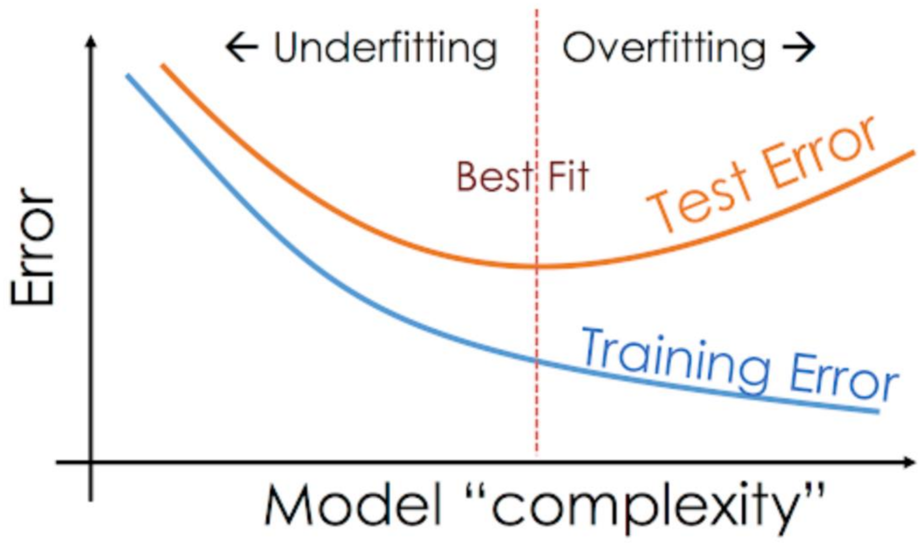
Similarly, our decision tree classifier tries to learn each and every point from the training data but suffers radically when it encounters a new data point in the test set. It is not able to generalize it well.



This situation where any given model is performing too well on the training data but the performance drops significantly over the test set is called an overfitting model.

For example, non-parametric models like decision trees, KNN, and other tree-based algorithms are very prone to overfitting. These models can learn very complex relations which can result in overfitting.

The graph below summarises this concept:



On the other hand, if the model is performing poorly over the test and the train set, then we call that an underfitting model. An example of this situation would be building a linear regression model over non-linear data.

Techniques to reduce underfitting:

- Increase model complexity
- Increase the number of features, performing feature engineering
- Remove noise from the data.
- Increase the number of epochs or increase the duration of training to get better results.

Techniques to reduce overfitting:

- Increase training data.
- Reduce model complexity.
- Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
- Ridge Regularization and Lasso Regularization
- Use dropout for neural networks to tackle overfitting.
- Good Fit in a Statistical Model: