**Addressing Overfitting 2023 Guide — 13 Methods**

**Your one-stop place to learn 13 effective methods to prevent overfitting in machine learning and deep learning models**

***Who doesn’t like to find the solutions for the worst problem that most data scientists face? “The problem of overfitting”***

This article may be the one-stop place to learn many effective methods to prevent overfitting in machine learning and deep learning models.

**What happens in overfitting?**

Overfitting usually occurs when the model is *too complex*. When a model overfits the training data, the following things happen:

* **The model tries to memorize the training data instead of learning essential patterns from the data.** Machine learning involves *learning* patterns and rules from data, not *memorizing* data.
* **The model performs well only on the training data and poorly performs on new unseen data.** A good model should be able to perform well on the training data as well as to generalize well on new unseen data.

**How to detect overfitting**

The next question is:

How do we identify overfitting in machine learning and deep learning models?

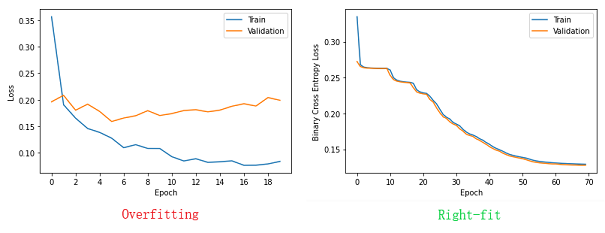
**Using the learning curve**

Overfitting is something that we can't see with our eyes! A simple, but effective [machine learning visualization](https://towardsdatascience.com/10-amazing-machine-learning-visualizations-you-should-know-in-2023-528282940582) called the *learning curve* can be used to detect overfitting in machine learning and deep learning modes.

The learning curve plots the training and validation scores against the number of epochs.

The learning curve indicates the model is overfitting:

* If there is a clear gap between the training and validation scores.
* When the validation error (loss) begins increasing at some point while the training error (loss) still decreases. In the case of accuracy, the validation accuracy begins decreasing at some point while the training accuracy still increases.



**Detecting overfitting with the learning curve** (Image by author)

**Using the validation curve**

The learning curve is very common in deep learning models. To detect overfitting in general machine learning models such as decision trees, random forests, k-nearest neighbors, etc., we can use another [machine learning visualization](https://towardsdatascience.com/10-amazing-machine-learning-visualizations-you-should-know-in-2023-528282940582) called the *validation curve*.

The validation curve plots the influence of a *single* hyperparameter on the train and validation set.

The x-axis represents the given hyperparameter's values while the y-axis represents the training and validation scores.

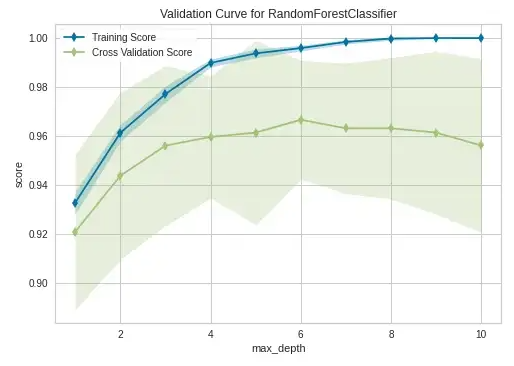
We can use the validation curve to detect overfitting in machine-learning models for the given values of a single hyperparameter.

For that, we need to identify the most important model hyperparameter and plot the influence of its values using the validation curve.

Some examples of this include:

* We can use the validation curve to plot the influence of the *max\_depth* (tree depth) hyperparameter of a decision tree or random forest model.
* We can use the validation curve to plot the influence of the *n\_neighbors* (number of neighbors) hyperparameter of a KNN model.

The following plot shows the validation curve created for a random forest classifier to measure the influence of the *max\_depth* (tree depth) hyperparameter on training and validation scores (accuracies).



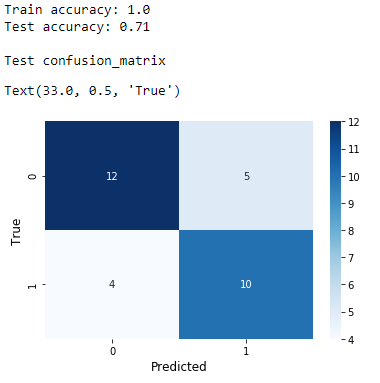
**Validation curve** (Image by author)

After the *max\_depth* value of 6, the model begins to overfit the training data. In other words, the validation accuracy begins decreasing at *max\_depth=6* while the training accuracy still increases.

**Using multiple evaluation metrics**

Based on the type of machine learning algorithm we use, Scikit-learn and TensorFlow provide different types of model evaluation metrics. We can use (even combine) those evaluation metrics to monitor the performance of the model during training and then determine whether the model is overfitting or not by analyzing the values of evaluation metrics.

The following image shows the training and test accuracies, and the confusion matrix for the test data of a fully-grown decision tree classifier.

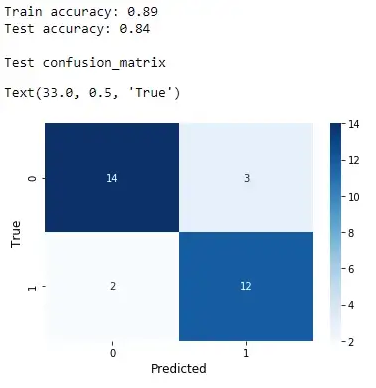


**Evaluating a decision tree classifier** (Image by author)

Overfitting is guaranteed in fully-grown decision trees! 100% train accuracy means that the decision tree classifier model performs well on the training set. 71% test accuracy clearly indicates that the model does not perform well on new unseen data. The test accuracy is much lower than the training accuracy. In other words, there is a clear gap between training and test accuracies. These things indicate that the model is clearly overfitting.

The number of false positives and false negatives is also high in this case. It is another indication that the model does not perform well on test data.

After applying proper regularization techniques (limiting the tree growth and creating ensembles), we get the following evaluation values.



(Image by author)

Both train and test accuracy scores are high and there is no clear gap between them. In addition to that, the number of false positives and false negatives has also been reduced. Now, it is clear that the regularized decision tree model is not overfitting now.

**13 Effective methods for addressing overfitting**

Here is a summary of methods used to prevent overfitting in machine learning and deep learning models. We’ll discuss each method in detail.

**Addressing Overfitting - 13 Methods**  
-----------------------------------  
01. [Dimensionality Reduction](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#76e6)  
02. [Feature Selection](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#2943)  
03. [Early Stopping](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#c513)  
04. [K-Fold Cross-Validation](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#ae2a)  
05. [Creating Ensembles](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#832f)  
06. [Pre‐Pruning](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#5b1b)  
07. [Post‐Pruning](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#235d)  
08. [Noise Regularization](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#bbc5)  
09. [Dropout Regularization](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#8cde)  
10. [L1 and L2 Regularization](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#27c3)  
11. [Data (Image) Augmentation](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#4d98)  
12. [Adding More Training Data](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#e080)  
13. [Reducing Network Width & Depth](https://towardsdatascience.com/addressing-overfitting-2023-guide-13-methods-8fd4e04fc8#3dc4)

***Note:*** If you’d like to get hands-on experience with each method discussed below, I’ve created a separate article collection for that! Visit [this link](https://rukshanpramoditha.medium.com/list/handson-practice-for-addressing-overfitting-2023-guide-de3820029e5a) to access all the articles. There, you’ll learn how to apply each method by writing code!

Let’s get started!

**1️⃣ Addressing overfitting with dimensionality reduction**

Overfitting often happens when the model is *too complex*. The main reason for model complexity is the existence of many features (variables) in the data. The number of features in the data is called its dimensionality.

The model tends to overfit the training data when its dimensionality is high.

Reducing the number of features in the data is called *dimensionality reduction*. We should keep as much of the variance in the original data as possible. Otherwise, we lose useful information in the data.

Dimensionality reduction takes care of overfitting as follows.

1. Dimensionality reduction reduces the number of features in the data. After applying dimensionality reduction, the model’s complexity will also be reduced. So, the model will not overfit the training data anymore!
2. Dimensionality reduction also removes unnecessary noise in the data. Noisy data will cause overfitting. The model will prevent overfitting the training data after removing the noise in the data.

The most common dimensionality reduction method is the *Principal Component Analysis (PCA)*. It finds a new set of uncorrelated features for the data in a lower dimensional form.

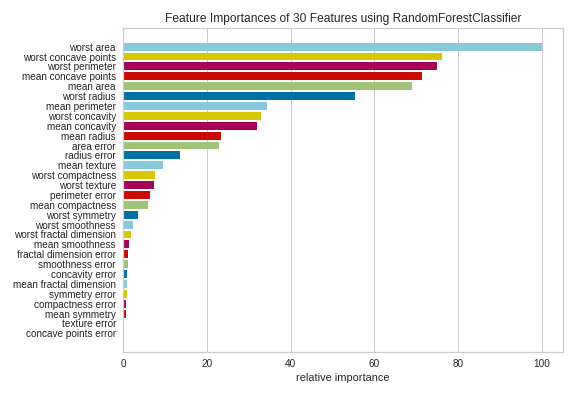
PCA can effectively eliminate the problem of overfitting.

**2️⃣ Addressing overfitting with feature selection**

Feature selection can be considered as a dimensionality reduction method as it removes redundant (unnecessary) features from the dataset. That reduces the number of features (dimensionality) in the data.

Instead of finding a new set of features, the feature section method only keeps the most important features by removing the unwanted features in the data. The original values remain unchanged. That’s how feature selection differs from PCA where we get new transformed values.

A simple [machine learning visualization](https://towardsdatascience.com/10-amazing-machine-learning-visualizations-you-should-know-in-2023-528282940582) called *feature importances plot* can be used to select the most important features. This plot is created based on the relative importance of each feature. The following image shows such a plot created using the *breast\_cancer* dataset which contains 30 features.



**Feature importance plot for the *breast\_cancer* dataset** (Image by author)

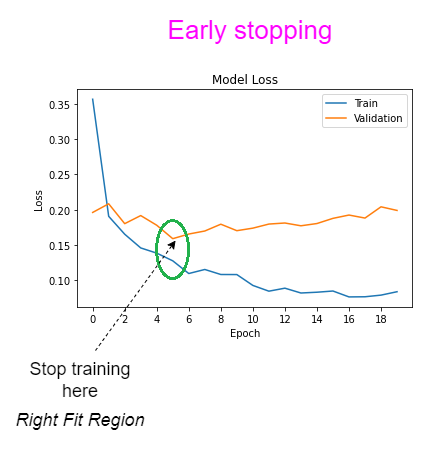
You can see that some features do not much contribute to the model. Their contribution is negligible. So, we can remove those features and build the model only with the most important features.

Removing the least important features from the data will reduce the model’s complexity and eliminate noise (if any) from the data. That is how feature selection prevents the model from overfitting.

**3️⃣ Addressing overfitting with early stopping**

Early stopping is another effective method to prevent overfitting in both machine learning and deep learning models.

In early stopping, we intentionally stop the model training process early right before the model begins to overfit by looking at the learning curve or validation curve.



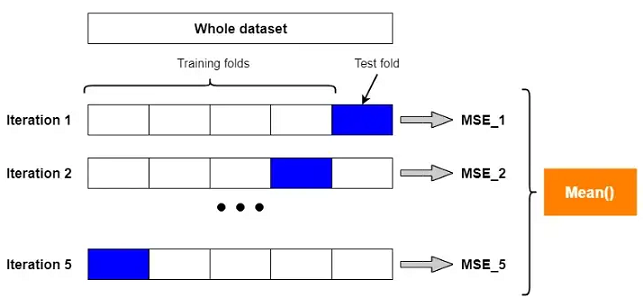
**Performing early stopping using the learning curve** (Image by author)

After the 5th epoch, the model beings to overfit the training data. If we continue training after that epoch, the validation error increases although the training error further decreases. The gap between the training and validation scores also increases. These are the signs of overfitting.

**4️⃣ Addressing overfitting with k-fold cross-validation**

K-fold cross-validation is a data-splitting strategy. When building ML and DL models, we usually split the full dataset into a train and test set. Here, the problem is that the model only sees a specific set of instances (data points) during the training process.

In k-fold cross-validation, the full dataset is split into different folds depending on the value of *k* (usually 5 or 10). Each fold contains different types of instances (data points). The model is trained on k-1 folds of data in each iteration. The evaluation is done using the remaining fold of data in each iteration. The training and evaluation folds are changed at each iteration as in the following diagram. The evaluation score is calculated in each iteration and the average is taken.



**K-fold cross-validation procedure: k=5** (Image by author)

In k-fold cross-validation, the model sees different sets of instances (data points) during training as training and evaluation folds are changed at each iteration. The model will learn all the required patterns and will generalize well on new unseen data. In other words, k-fold cross-validation prevents the model from overfitting.

**5️⃣ Addressing overfitting by creating ensembles**

This method is limited to tree-based models. Decision tree models always overfit the training data unless we limit the tree growth by setting a lower value for the *max\_depth* (tree depth) hyperparameter.

Even if we limit the tree growth during training, decision tree models may still overfit the training data. A useful approach to reduce overfitting in decision trees is creating ensembles. An ensemble (group) is a collection of multiple decision trees created from subsets of the training data and features.

For example, a random forest is an ensemble that contains a group of uncorrelated decision trees.

Compared to a decision tree model, a random forest is less likely to overfit the training data because of its extra randomness.

The extra randomness occurs in random forests due to *uncorrelated* trees. The data is well mixed up when creating a random forest. In addition to that, the final outcome is calculated by averaging the outcomes of each uncorrelated tree. So, a random forest can produce more accurate and stable results than a single decision tree.

**6️⃣ Addressing overfitting with pre‐pruning**

Pruning methods are applied to prevent overfitting in decision trees. There are two main types of pruning methods called pre‐pruning and post-pruning.

By default, a decision tree is grown into its full depth. Fully grown trees always overfit the training data.

In decision trees, pruning is the process of controlling the growth of the tree.

Pre-pruning applies an early stopping rule which stops the growth of a decision tree too early. After pre-pruning, the decision tree has fewer branches. We can apply pre-pruning in decision trees by limiting the values of the following hyperparameters.

* **max\_depth:** The maximum depth of the tree. Decreasing this value prevents overfitting.
* **min\_samples\_leaf:** The minimum number of samples required to be at a leaf node. Increasing this value prevents overfitting.
* **min\_samples\_split:** The minimum number of samples required to split an internal node. Increasing this value prevents overfitting.

There are two common methods to tune these hyperparameters.

* Measure the influence of a single hyperparameter at a time while keeping other hyperparameters on their default values. For this, we can use the validation curve to detect overfitting.
* Tuning multiple hyperparameters at once. The validation curve cannot be used here to measure the influence of multiple hyperparameters. Here, [grid search or random search](https://towardsdatascience.com/python-implementation-of-grid-search-and-random-search-for-hyperparameter-optimization-2d6a82ebf75c) should be used to tune multiple hyperparameters at once.

**7️⃣ Addressing overfitting with post‐pruning**

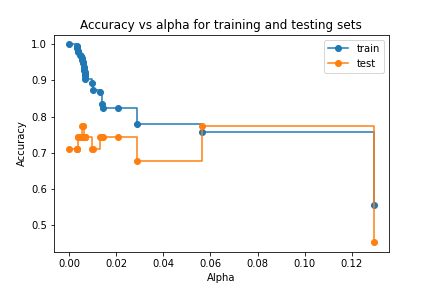
The post-pruning is the process of removing parts of the tree after the tree has been fully grown.

Cost complexity pruning (ccp) is a post-pruning method. It involves finding the right value for the ***ccp\_alpha*** hyperparameter in Scikit-learn decision tree classes.

The default value of *ccp\_alpha* is zero which means no pruning will be performed by default. Larger values increase the number of nodes to be pruned and reduce the tree depth. So, larger values of *ccp\_alpha* prevent overfitting.

To find the optimal value for the ***ccp\_alpha*** hyperparameter:

* We can try different values such as 0.01, 0.02, 0.05, 0.1, etc., and monitor training and validation scores.
* We can pass all effective alpha values to the **ccp\_alpha** hyperparameter one at a time and then calculate the training and validation scores. All effective values of alpha can be accessed by the **ccp\_alphas** attribute.



**Tuning the ccp\_alpha hyperparameter** (Image by author)

At *alpha=0.06*, the validation accuracy begins increasing while the training accuracy almost remains the same. The next available value for alpha is 0.12 which has very poor performance scores. According to this plot, the optimal value for the alpha is 0.06.

**8️⃣ Addressing overfitting with noise regularization**

Adding noise to the existing data is an effective regularization method to prevent overfitting in neural networks. We usually add noise to the input layer of a neural network as the input layer holds the training data although it is possible to add noise to the hidden layers and output layers as well.

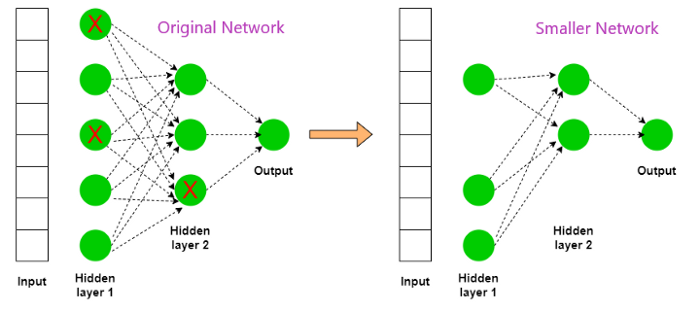
When adding noise to the training data, a small amount of noise will be added to each training instance and generate different versions of the same instance. That will expand the original dataset! This will implicitly add more training data to the original dataset. Adding more training data will help to prevent overfitting.

In addition to that, after adding noise to the data, the model will ***not*** solely capture noise in the training data. That will also help to reduce overfitting.

**9️⃣ Addressing overfitting with dropout regularization**

Dropout regularization is a neural network-specific regularization method to prevent overfitting in neural networks.

In dropout regularization, the algorithm randomly removes some nodes from the network during training based on the probability value that we define in each layer. The removed nodes do not participate in the parameter updating process. Dropout regularization is applied per-layer basis. It means that we can set different dropout probabilities in each layer separately.



**Dropout regularization process** (Image by author)

The original network becomes smaller after applying dropout regularization. Smaller networks are less flexible so overfitting will not happen there.

In dropout regularization, some weight values are zero because their nodes are inactive. So, all other weights need to participate in the weight-updating process. The network’s output does not depend on certain large weights. This can reduce overfitting in neural networks.

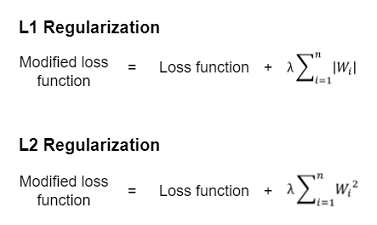
Dropout regularization is the most effective method to prevent overfitting in neural networks.

**1️⃣0️⃣ Addressing overfitting with L1 and L2 regularization**

L1 and L2 regularization is commonly applied to neural network models to prevent overfitting. The selection of L1 and L2 regularization depends on the regularization term that we add to the loss function during training.

When the regularization term is ***L1 norm*** [*λ \* (Sum of the absolute values of the weights)*], it is called *L1 regularization*. When the regularization term is ***L2 norm*** [*λ \* (Sum of the squared values of the weights)*], it is called *L2 regularization*.

L1 and L2 regularization for neural networks are defined as follows.



**L1 and L2 regularization** (Image by author)

λ controls the level of regularization. Therefore, it is called the regularization parameter (factor).

* lambda=0: Min value. No regularization is applied.
* lambda=1: Max value. Full regularization is applied.

Larger values for λ imply stronger regularization which reduces overfitting in neural networks.

Generally, L1 and L2 regularization keeps the weights of a neural network small. There will not be any large weights that put too much emphasis on some inputs. Small weight values are less sensitive to the noise present in the input data. Therefore, overfitting will not happen in L1 and L2 regularized neural networks.

L1 and L2 regularization methods can also be applied to general machine learning algorithms such as logistic regression, linear regression, etc.

In logistic regression, there is a hyperparameter called ***penalty*** (values: *‘l1’*, *‘l2’* and *‘elasticnet’*)to choose the type of regularization. Note that ‘*elasticnet’* applies both L1 and L2 regularization to the model at the same time.

For linear regression, Scikit-learn provides three separate classes called Ridge()(applies L2 regularization), Lasso()(applies L1 regularization) and ElasticNet()(applies both L1 and L2 regularization) for each type of regularization.

**1️⃣1️⃣ Addressing overfitting with data (image) augmentation**

Data augmentation is usually performed on image data. Therefore, data augmentation is also referred to as *image augmentation* in some contexts.

Image augmentation is the process of increasing the number of images by generating new variants of the same images with some transformation which includes *zooming*, *flipping*, *rotating*, *shifting*, *scaling*, *lighting*, *compressing*, *cropping*, etc.

The most important thing in image augmentation is to preserve the context of images. In other words, the image context should not be changed when augmenting images.

Image augmentation is much more suitable for deep learning models which often require more training data. It is an inexpensive way to give the model more training data!

Image augmentation reduces overfitting in neural networks as follows.

1. Image augmentation extends the dataset by adding more training instances. Adding more training data will prevent overfitting.
2. Image augmentation allows the neural network to see many variants of the same images during training. This reduces the dependency on the original form of images when learning important features. The network will become more robust and stable when tested on new unseen data.

**1️⃣2️⃣ Addressing overfitting by adding more training data**

Adding more training data to the model will prevent overfitting. There are many ways for adding more training data to the model.

* Collect new relevant data (expensive)
* Expand the original dataset by adding noise to the data (inexpensive)
* Data augmentation (inexpensive)

**1️⃣3️⃣ Addressing overfitting by reducing network width and depth**

The structure of a neural network is defined by its width and depth. The *depth* defines the number of hidden layers in a neural network. The *width* defines the number of nodes (neurons/units) in each layer of a neural network.

Decreasing the number of hidden layers and the number of hidden units reduces the flexibility of the network. Less flexible networks do not capture the noise in the data and will not overfit the training data.

This is the end of today’s post.

**Please let me know if you’ve any questions or feedback.**

**Read next (highly recommended)**

* **Get Hands-on Practice for “Addressing Overfitting 2023 Guide”**