

Machine learning workflow

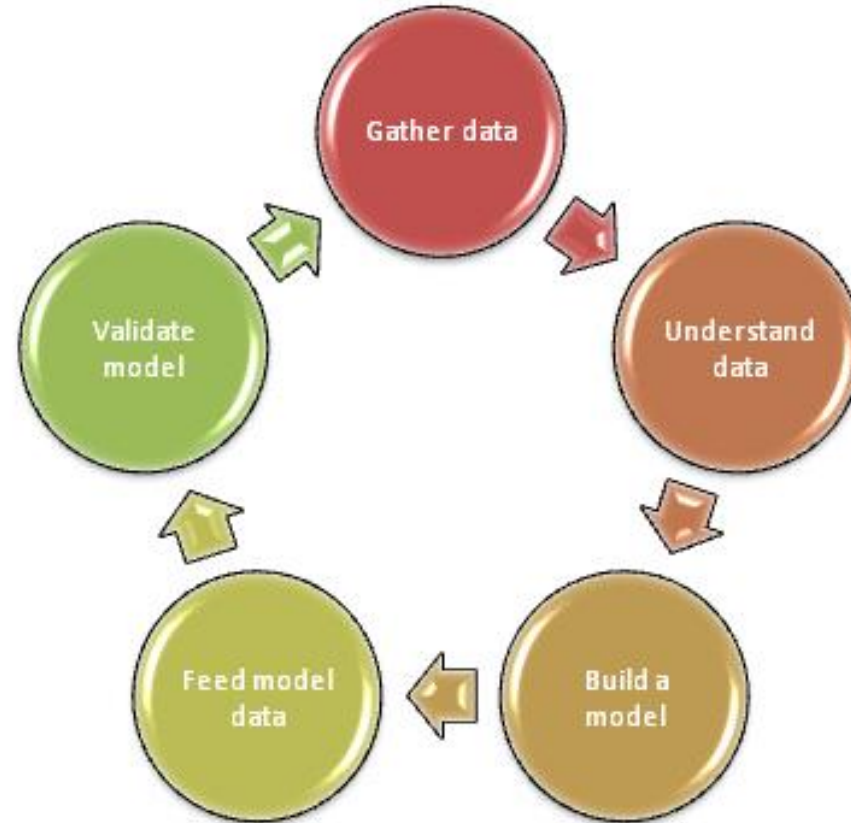
Francesco Pugliese, PhD

neural1977@gmail.com

Table of contents

- ✓ Feature Engineering
- ✓ Learning algorithm selection
- ✓ Three sets
- ✓ Underfitting and overfitting
- ✓ Regularization
- ✓ Model performance assessment
- ✓ Hyper-parameter tuning

Machine Learning Workflow



Feature Engineering

- ✓ The problem of **transforming** raw data into a dataset is called feature engineering.
- ✓ For most practical problems, feature engineering is a labor-intensive process that demands from the data analyst a lot of **creativity** and, preferably, **domain knowledge**.
- ✓ The role of the data analyst is to create informative features: those would allow the learning algorithm to build a model that predicts well labels of the data used for training.
- ✓ Highly informative features are also called features with high predictive power.
- ✓ For example, the average duration of a user's session has high predictive power for the problem of predicting whether the user will keep using the application in the future.
- ✓ We say that a model has a low bias when it predicts the training data well. That is, the model makes few mistakes when we use it to predict labels of the examples used to build the model.

Feature Engineering

- ✓ Feature engineering involves following,
 - ✓ One hot encoding
 - ✓ Binning
 - ✓ Normalization
 - ✓ Standardization
 - ✓ Dealing with missing features
 - ✓ Data imputation techniques

One hot encoding

- ✓ Some learning algorithms only work with numerical feature vectors.
- ✓ When some feature in your dataset is categorical, like "colors" or "days of the week", you can transform such a categorical feature into several binary ones.
- ✓ If your example has a categorical feature "colors" and this feature has three possible values: "red", "yellow", "green", you can transform this feature into a vector of three numerical values:

red = [1, 0, 0]
yellow = [0, 1, 0]
green = [0, 0, 1]

- ✓ By doing so, you increase the dimensionality of your feature vectors.

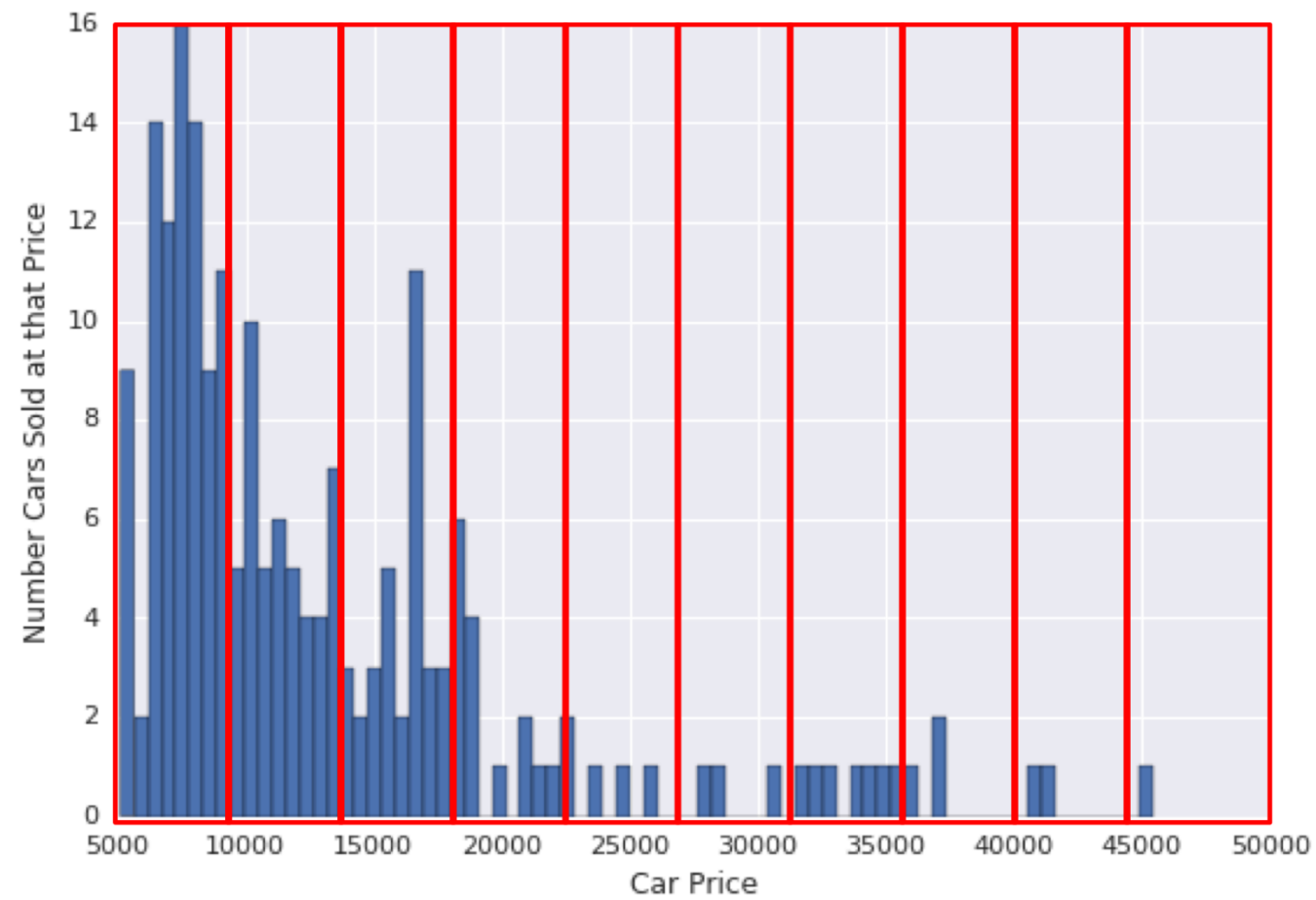
One hot encoding

- ✓ You should not transform red into 1, yellow into 2, and green into 3 to avoid increasing the dimensionality because that would imply that there's an order among the values in this category and this specific order is important for the decision making.
- ✓ If the order of a feature's values is not important, using ordered numbers as values is likely to confuse the learning algorithm,¹ because the algorithm will try to find a regularity where there's no one, which may potentially lead to overfitting.

Binning

- ✓ When you have a **numerical** feature but you want to **convert** it into a **categorical** one.
- ✓ Binning (also called **bucketing**) is the process of converting a continuous feature into multiple binary features called bins or buckets, typically based on value range.
- ✓ For example, instead of representing **age** as a single real-valued feature, the analyst could chop ranges of age into discrete bins: all ages between 0 and 5 years-old could be put into one bin, 6 to 10 years-old could be in the second bin, 11 to 15 years-old could be in the third bin, and so on.
- ✓ For example, let feature $j = 4$ represent age. By applying binning, we replace this feature with the corresponding bins. Let the three new bins, "age_bin1", "age_bin2" and "age_bin3" be added with indexes $j = 123$, $j = 124$ and $j = 125$ respectively. Now if $x(4) = 7$ for some example x_i , then we set feature $x(124)$ to 1; if $x(4) = 13$, then we set feature $x(125)$ to 1, and so on.
- ✓ In some cases, a carefully designed binning can help **the learning algorithm to learn using fewer examples**. It happens because we give a "**hint**" to the learning algorithm that if the value of a feature falls within a specific range, the exact value of the feature doesn't matter.

Binning



Data Normalization

- ✓ Also referred to as **Column Normalization**. Data Normalization usually means to scale a variable to have a value between 0 and 1, and we can achieve that by using the following formula:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- ✓ **Example:** Let us consider a list of numeric values, $x[] = [3.5, 3.0, 3.2, 3.1, 3.6, 3.7, 3.4, 3.4, 2.9, 2.7]$ and apply data normalization using above equation as follows,

$x_{max} = \max(\text{numAry}) = 3.7$

$x_{min} = \min(\text{numAry}) = 2.7$

$x[]_{new} = [0.7, 0.2, 0.5, 0.3, 0.8, 1.0, 0.6, 0.6, 0.1, 0.0]$

Data Standardization

- ✓ Standardization transforms data to have a mean of zero and a standard deviation of 1. Data points can be standardized with the following formula:

$$Z_i = \frac{x_i - \bar{x}}{S}$$

Where S is standard deviation

Standard deviation / Formula

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$$

σ = population standard deviation
 N = the size of the population
 x_i = each value from the population
 μ = the population mean

Feedback

- ✓ **Example:** Let us consider a list of numeric values, $x[] = [3.5, 3.0, 3.2, 3.1, 3.6, 3.7, 3.4, 3.4, 2.9, 2.7]$ and apply data standardization using above equation as follows,

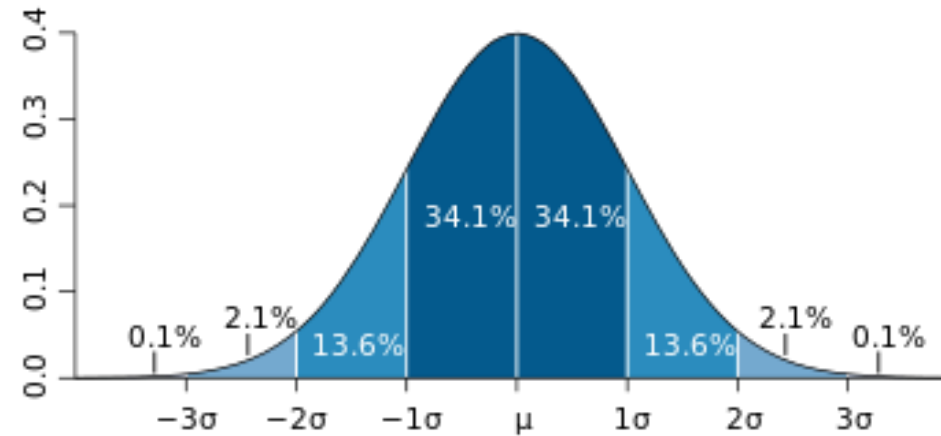
$$\bar{x} = \text{mean}(x[]) = 3.25$$

$$S = \text{standard deviation}(x[]) = 0.32403703492039304$$

$$x[] \text{ new} = [0.77, -0.77, -0.15, -0.46, 1.08, 1.38, 0.46, 0.46, -1.0, -1.6]$$

Data Standardization

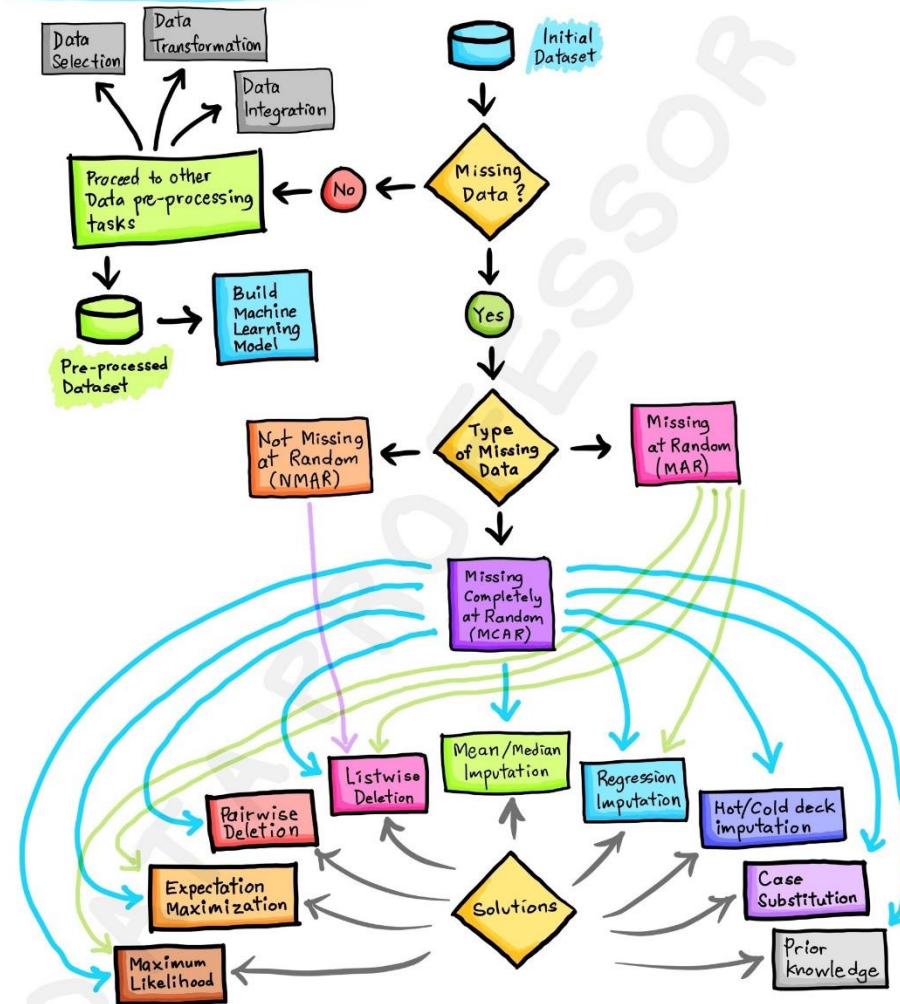
- ✓ What we meant by standard deviation by 1 is all the points will lie between -1σ and 1σ (as shown below).
- ✓ As 68% of the values lie within one standard deviation of the mean. Here we can say that one standard deviation is lying between -1.5 and 1.5.



- ✓ Column Standardization is also called as Mean Centering.
- ✓ Sometimes it's also known as z-score.

HANDLING MISSING DATA

By: Chanin Nantasenamat



Dealing with missing features

- ✓ In some cases, the data comes to the analyst in the form of a dataset with features already defined. In some examples, values of some features can be missing.
- ✓ That often happens when the dataset was handcrafted, and the person working on it forgot to fill some values or didn't get them measured at all.
- ✓ The typical approaches of dealing with missing values for a feature include:
 - ✓ **removing** the examples with missing features from the dataset (that can be done if your dataset is big enough so you can sacrifice some training examples);
 - ✓ using a learning algorithm that can deal with missing feature values (depends on the library and a specific implementation of the algorithm);
 - ✓ using a **data imputation** technique.

Data Imputation techniques

- ✓ One data imputation technique consists in replacing the missing value of a feature by an average value of this feature in the dataset
- ✓ Another technique is to replace the missing value by a value outside the normal range of values.
- ✓ For example, if the normal range is $[0, 1]$, then you can set the missing value to 2 or -1 .
- ✓ The idea is that the learning algorithm will learn what is best to do when the feature has a value significantly different from regular values.
- ✓ Alternatively, you can replace the missing value by a value in the middle of the range. For example, if the range for a feature is $[-1, 1]$, you can set the missing value to be equal to 0.
- ✓ Here, the idea is that the value in the middle of the range will not significantly affect the prediction.

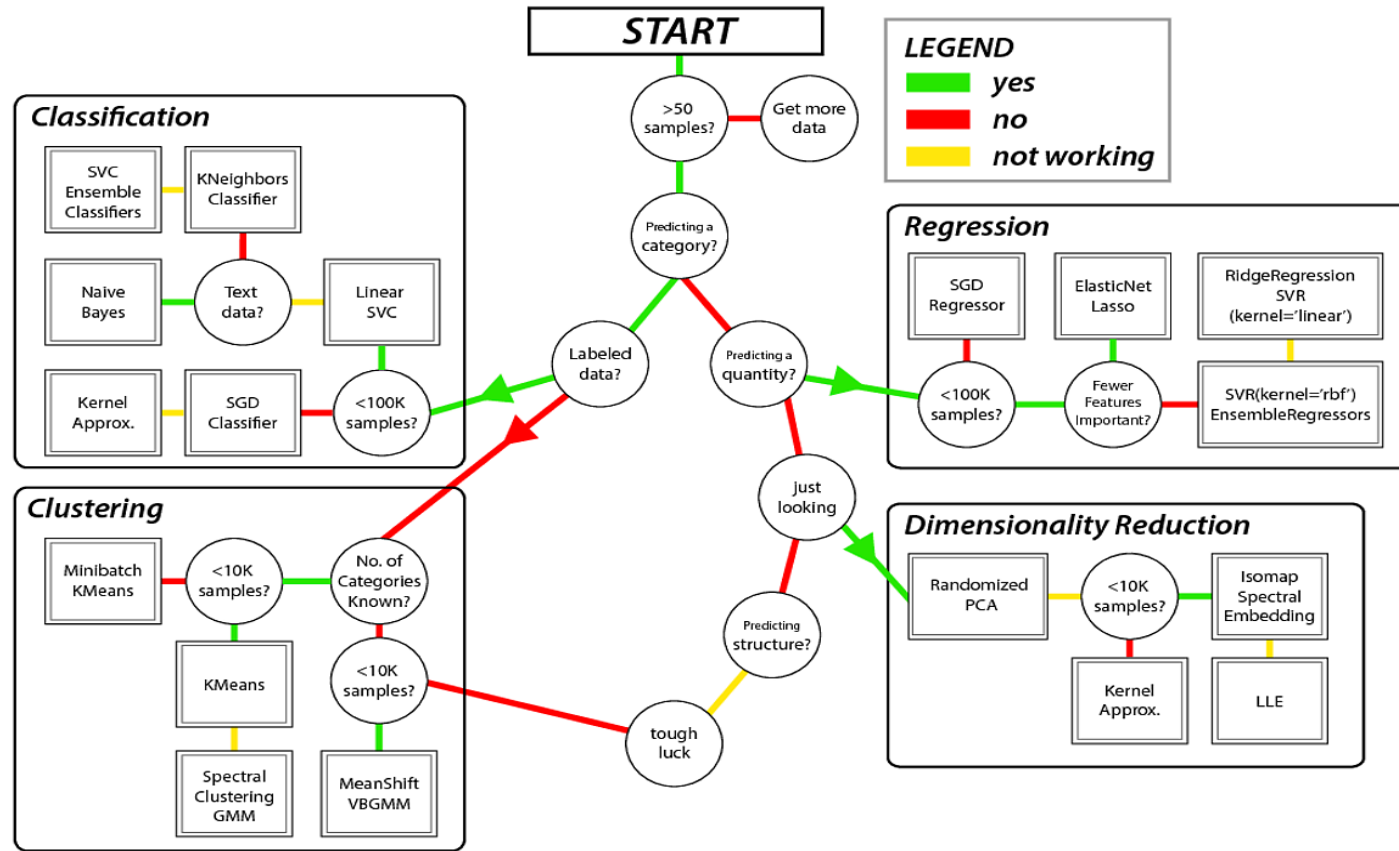
Data Imputation techniques

- ✓ A more advanced technique is to use the missing value as the target variable for a regression problem. You can use all remaining features $[x(1), x(2), \dots, x(j-1), x(j+1), \dots, x(D)]$ to form a feature vector \hat{x}^i , set $\hat{y}^i \leftarrow x(j)$, where j is the feature with a missing value. Then you build a regression model to predict \hat{y} from \hat{x} .
- ✓ Of course, to build training examples (\hat{x}, \hat{y}) , you only use those examples from the original dataset, in which the value of feature j is present.
- ✓ Finally, if you have a significantly large dataset and just a few features with missing values, you can increase the dimensionality of your feature vectors by adding a binary indicator feature for each feature with missing values.
- ✓ Let's say feature $j = 12$ in your D -dimensional dataset has missing values. For each feature vector x , you then add the feature $j = D + 1$ which is equal to 1 if the value of feature 12 is present in x and 0 otherwise. The missing feature value then can be replaced by 0 or any number of your choice.

Data Imputation techniques

- ✓ At prediction time, if your example is not complete, you should use the same data imputation technique to fill the missing features as the technique you used to complete the training data.
- ✓ Before you start working on the learning problem, you cannot tell which data imputation technique will work the best.
- ✓ Try several techniques, build several models and select the one that works the best.

Learning algorithm selection



From: [Pinterest](#)

Learning algorithm selection

- ✓ Choosing a machine learning algorithm can be a **difficult task**.
- ✓ You can ask yourself **several questions** before starting to work on the problem. Depending on your answers, you can shortlist some algorithms and try them on your data.
- ✓ Does your model have to be **explainable** to a non-technical audience?
 - ✓ Most very accurate learning algorithms are so-called "black boxes".
 - ✓ They learn models that make very few errors, but why a model made a specific prediction could be very hard to understand and even harder to explain.
 - ✓ Examples of such models are **neural networks** or **ensemble models**.
 - ✓ On the other hand, **kNN**, **linear regression**, or **decision tree** learning algorithms produce models that are **not always the most accurate**, however, the way they make their **prediction is very straightforward**.

Learning algorithm selection

- ✓ **In-memory vs. out-of-memory:**

- ✓ Can your dataset be fully loaded into the **RAM** of your server or personal computer? If yes, then you can choose from a wide variety of algorithms.
- ✓ Otherwise, you would prefer **incremental learning algorithms** that can improve the model by adding more data gradually.

- ✓ **Number of features and examples:**

- ✓ How many training examples do you have in your dataset?
- ✓ How many features does each example have?
- ✓ Some algorithms, including **neural networks and gradient boosting**, can handle a **huge** number of examples and millions of features. Others, like **SVM**, can be very **modest** in their capacity.

Learning algorithm selection

- ✓ **Categorical vs. numerical features:**

- ✓ Is your data composed of categorical only, or numerical only features, or a mix of both?
- ✓ Depending on your answer, some algorithms cannot handle your dataset directly, and you would need to **convert** your categorical features into numerical ones.

- ✓ **Training speed:**

- ✓ How much **time** is a learning algorithm allowed to use to **build a model**?
- ✓ **Neural networks** are known to be **slow** to train. **Simple** algorithms like logistic and linear regression or decision trees are much **faster**.
- ✓ **Specialized libraries** contain very **efficient** implementations of some algorithms; you may prefer to do research online to find such libraries.
- ✓ Some algorithms, such as **random forests**, benefit from the **availability of multiple CPU cores**, so their model building time can be significantly reduced on a machine with dozens of cores.

Learning algorithm selection

- ✓ **Nonlinearity of the data:**

- ✓ Is your data **linearly separable** or can it be modelled using a linear model?
- ✓ If yes, **SVM** with the linear kernel, **logistic or linear** regression can be good choices.
- ✓ Otherwise, **deep neural networks** or **ensemble** algorithms, might work better.

- ✓ **Prediction speed:**

- ✓ How **fast** does the model have to be when **generating predictions**?
- ✓ Will your model be used in production where very high throughput is required?
- ✓ Algorithms like **SVMs**, **linear** and **logistic** regression, and (some types of) neural networks, are **extremely fast at the prediction time**.
- ✓ Others, like **kNN**, **ensemble** algorithms, and very deep or **recurrent** neural networks, are **slower**.

- ✓ If you don't want to guess the best algorithm for your data, a popular way to choose one is by **testing it on the validation set**.

Three sets

- ✓ In practice data analysts work with three distinct sets of labelled examples:
 - ✓ **training** set
 - ✓ **validation** set
 - ✓ **test** set.
- ✓ Once you have got your annotated dataset, the first thing you do is you **shuffle** the examples and split the dataset into three subsets: training, validation, and test.
- ✓ The **training** set is usually the **biggest** one; you use it to **build** the model.
- ✓ The **validation** and **test** sets are roughly the **same** sizes, much smaller than the size of the training set.
- ✓ The learning algorithm **cannot** use examples from these two subsets to build the model.
- ✓ That is why those two sets are often called **holdout sets**.

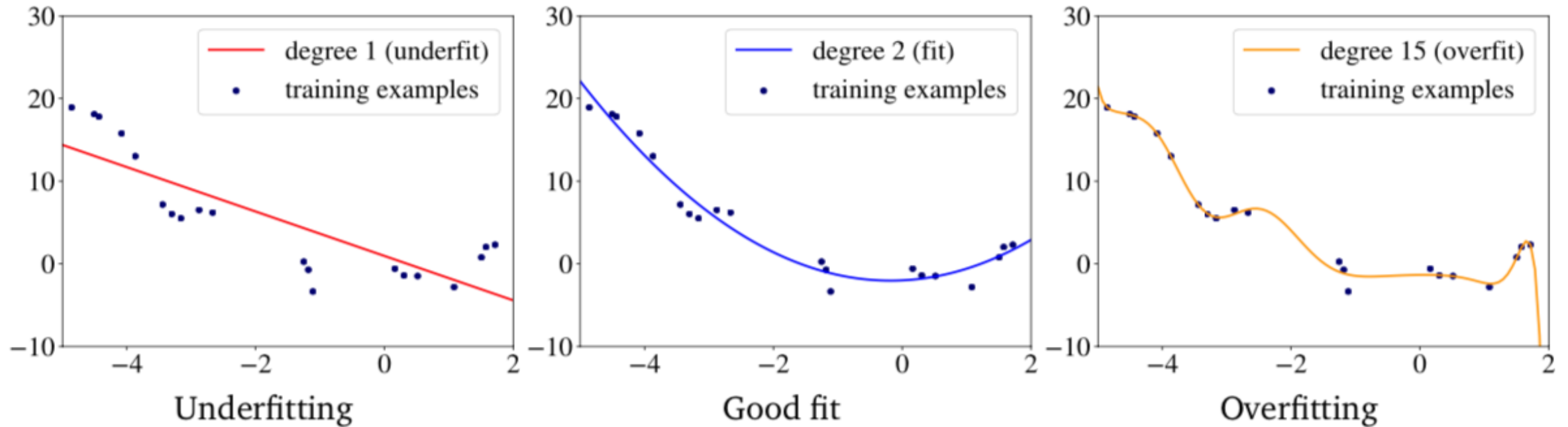
Three sets

- ✓ There's no optimal proportion to split the dataset into these three subsets.
- ✓ In the past, the rule of thumb was to use **70%** of the dataset for **training**, **15%** for **validation** and **15%** for **testing**.
- ✓ However, in the age of **big data**, datasets often have **millions** of examples.
- ✓ In such cases, it could be reasonable to keep **95%** for **training** and **2.5%/2.5%** for **validation/testing**.

Why three sets ?

- ✓ The answer is simple: when we build a model, what **we do not want** is for the model to only do well at **predicting labels** of examples the learning algorithms has **already seen**.
- ✓ A **trivial** algorithm that simply **memorizes** all training examples and then uses the memory to “predict” their labels will make no mistakes when asked to predict the labels of the training examples, but such an algorithm would be **useless** in practice.
- ✓ What we really want is a model that is **good** at **predicting** examples that the learning algorithm **didn't see**: we want **good performance on a holdout set**.
- ✓ **Why do we need two holdout sets and not one?**
 - ✓ We use the **validation** set to
 - ✓ **choose the learning algorithm** and
 - ✓ find the **best** values of **hyperparameters**.
 - ✓ We use the **test** set to **assess** the model before **delivering** it to the client or putting it in **production**.

Underfitting and Overfitting



Examples of underfitting (linear model), good fit (quadratic model), and overfitting (polynomial of degree 15).

Underfitting

- ✓ Underfitting is the **inability** of the model to **predict well the labels** of the data it was trained on.
- ✓ There could be several **reasons** for underfitting, the most important of which are:
 - ✓ your model is too **simple** for the data (for example a **linear model can often underfit**)
 - ✓ the **features** you engineered are **not informative** enough
- ✓ The first reason(**model is simple**) is easy to illustrate in the case of **one-dimensional regression**:
 - ✓ the **dataset** can resemble a **curved** line, but our **model** is a **straight** line.
- ✓ The second reason(**non informative features**) can be illustrated like this:
 - ✓ let's say you want to **predict** whether a patient has **cancer**, and the **features** you have are **height, blood pressure, and heart rate**.
 - ✓ These three features are clearly **not good predictors** for cancer so our model will not be able to learn a **meaningful relationship** between these features and the label.
- ✓ The **solution** to the problem of **underfitting** is to try a **more complex model** or to engineer **features with higher predictive power**.

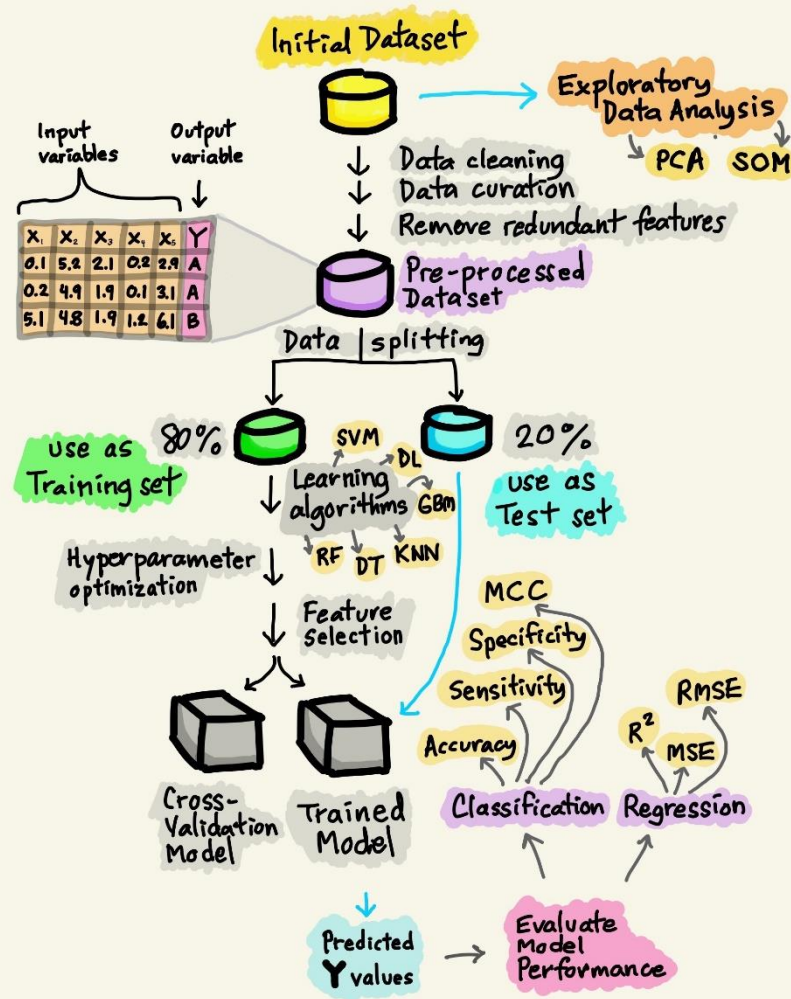
Overfitting

- ✓ The model that overfits predicts **very well** the **training** data but **poorly** the data from at least one of the two **holdout** sets.
- ✓ Several reasons can lead to overfitting, the most important of which are:
 - ✓ your model is **too complex** for the data (for example a very tall decision tree or a very deep or wide neural network often overfit);
 - ✓ you have **too many features** but a **small** number of **training** examples.
- ✓ In the literature, you can find another name for the problem of overfitting: **the problem of high variance**. This term comes from statistics. The variance is an error of the model due to its **sensitivity to small fluctuations** in the training set.
- ✓ It means that if your **training** data was **sampled differently**, the learning would result in a significantly **different** model.
- ✓ Which is why the model that overfits performs poorly on the test data: test and training data are **sampled** from the dataset **independently** of one another.

Solutions for Overfitting

- ✓ Several solutions to the problem of overfitting are possible:
 - ✓ Try a **simpler** model:
 - ✓ linear instead of polynomial regression or
 - ✓ SVM with a linear kernel instead of RBF, a neural network with fewer layers/units.
 - ✓ **Reduce** the **dimensionality** of examples in the dataset
 - ✓ For example, by using one of the dimensionality reduction techniques like **PCA**
 - ✓ Add **more training** data, if possible
 - ✓ **Regularize** the model
- ✓ **Regularization** is the **most widely** used approach to prevent overfitting

BUILDING THE MACHINE LEARNING MODEL



Regularization

- ✓ Process of introducing **additional information** in order to prevent overfitting
- ✓ In practice, that often leads to slightly **higher bias** but significantly **reduces the variance**. This problem is known in the literature as the **bias-variance trade-off**.
- ✓ The two most widely used types of regularization are called L1 and L2 regularization. A regression model that uses **L1** regularization technique is called **Lasso** Regression and model which uses **L2** is called **Ridge** Regression.
- ✓ The idea is quite simple. To create a regularized model, we **modify** the **objective function** by **adding a penalizing term** whose value is **higher** when the **model** is more **complex**.
- ✓ The key difference between these techniques is that **Lasso shrinks** the **less important feature's coefficient to zero** thus, **removing** some feature altogether. So, this works well for feature selection in case we have a huge number of features.

L1 & L2 Regularization

- ✓ Let us consider following linear regression,

$$\hat{y} = w_1x_1 + w_2x_2 + \dots + w_Nx_N + b$$

- ✓ L1 and L2 regularisation owes its name to L1 and L2 norm of a vector w respectively. Here's a primer on norms:

$$||\mathbf{w}||_1 = |w_1| + |w_2| + \dots + |w_N|$$

1-norm (also known as L1 norm)

$$||\mathbf{w}||_2 = (w_1^2 + w_2^2 + \dots + w_N^2)^{\frac{1}{2}}$$

2-norm (also known as L2 norm or Euclidean norm)

$$||\mathbf{w}||_p = (w_1^p + w_2^p + \dots + w_N^p)^{\frac{1}{p}}$$

p-norm

L1 & L2 Regularization

- ✓ It is the calculation of the loss function that includes these regularisation terms:

$$Loss = Error(y, \hat{y})$$

Loss function with no regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^N |w_i|$$

Loss function with L1 regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^N w_i^2$$

Loss function with L2 regularisation

- ✓ The regularisation terms are "constraints" by which an **optimisation algorithm** must "adhere to" when **minimising the loss function**, apart from having to minimise the error between the true y and the predicted \hat{y} .

L1 & L2 Regularization

- ✓ **L1** regularization produces a **sparse** model, a model that has most of its parameters (in case of linear models, most of $w(j)$) equal to zero. So L1 makes **feature selection** by deciding which features are essential for prediction and which are not. That can be useful in case you want to increase **model explainability**.
- ✓ However, if your only goal is to **maximize** the **performance** of the model on the holdout data, then **L2** usually gives **better results**.
- ✓ In addition to being widely used with linear models, L1 and L2 regularization are also frequently used with neural networks and many other types of models, which directly minimize an objective function.
- ✓ Neural networks also benefit from two other regularization techniques: **dropout** and **batch-normalization**. There are also non-mathematical methods that have a regularization effect: **data augmentation** and **early stopping**.

Model performance assessment

- ✓ Once you have a model which our learning algorithm has built using the training set, how can you say how good the model is? **You use the test set to assess the model.**
- ✓ The test set contains the examples that the learning algorithm has never seen before, so if our model performs well on predicting the labels of the examples from the test set, we say that our model generalizes well or, simply, that it's **good**.
- ✓ To be more rigorous, machine learning specialists use various formal metrics and tools to assess the model performance.
- ✓ For regression, we compute the mean squared error (MSE) for the training, and, separately, for the test data. If the MSE of the model on the test data is substantially higher than the MSE obtained on the training data, this is a sign of overfitting. Regularization or a better hyperparameter tuning could solve the problem.

Model performance assessment

- ✓ For classification, things are a little bit more complicated.
- ✓ The most widely used metrics and tools to assess the classification model are:
 - ✓ Confusion matrix
 - ✓ Precision & Recall
 - ✓ Accuracy
 - ✓ Cost sensitive accuracy
 - ✓ Area under the ROC curve (AUC)

Let's understand each of these 5 performance metrics

Confusion matrix for binary classification

- ✓ The confusion matrix is a table that summarizes how successful the classification model is at predicting examples belonging to various classes. One axis of the confusion matrix is the label that the model predicted, and the other axis is the actual label.
- ✓ In a binary classification problem, there are two classes. Let's say, the model predicts two classes: "spam" and "not_spam":

	spam (predicted)	not_spam (predicted)
spam (actual)	23 (TP)	1 (FN)
not_spam (actual)	12 (FP)	556 (TN)

Confusion matrix for multiclass classification

- ✓ The confusion matrix for multiclass classification has as many rows and columns as there are different classes.
- ✓ It can help you to determine mistake patterns.
- ✓ For example, a confusion matrix could reveal that a model trained to recognize different species of animals tends to mistakenly predict "cat" instead of "panther", or "mouse" instead of "rat"
- ✓ In this case, you can decide to add more labeled examples of these species to help the learning algorithm to "see" the difference between them.
- ✓ Alternatively, you might add additional features the learning algorithm can use to build a model that would better distinguish between these species.
- ✓ Confusion matrix is used to calculate two other performance metrics: **precision** and **recall**.

Precision & Recall

- ✓ Precision is the ratio of correct positive predictions to the overall number of positive predictions:

$$\text{precision} \stackrel{\text{def}}{=} \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

- ✓ Recall is the ratio of correct positive predictions to the overall number of positive examples in the dataset:

$$\text{recall} \stackrel{\text{def}}{=} \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

- ✓ **For multiple classes:**

- ✓ Even if precision and recall are defined for the binary classification case, you can always use it to assess a multiclass classification model.
- ✓ To do that, first select a class for which you want to assess these metrics.
- ✓ Then you consider all examples of the selected class as positives and all examples of the remaining classes as negatives.

Precision & Recall

- ✓ To understand the meaning and importance of precision and recall for the model assessment it is often useful to think about the prediction problem as the problem of research of documents in the database using a query.
- ✓ The precision is the proportion of relevant documents in the list of all returned documents.
- ✓ The recall is the ratio of the relevant documents returned by the search engine to the total number of the relevant documents that could have been returned.
- ✓ In the case of the spam detection problem, we want to have high precision (we want to avoid making mistakes by detecting that a legitimate message is spam) and we are ready to tolerate lower recall (we tolerate some spam messages in our inbox).
- ✓ Almost always, in practice, we have to choose between a high precision or a high recall. It's usually impossible to have both

Achieve High precision and / or High recall ?

- ✓ We can achieve either of the two by various means:
 - ✓ by assigning a higher weighting to the examples of (the SVM algorithm accepts weightings of classes as input);
 - ✓ by tuning hyperparameters to maximize precision or recall on the validation set;
 - ✓ by varying the decision threshold for algorithms that return probabilities of classes;
 - ✓ For instance, if we use logistic regression or decision tree, to increase precision (at the cost of a lower recall), we can decide that the prediction will be positive only if the probability returned by the model is higher than 0.9.

Accuracy

- ✓ Accuracy is given by the number of correctly classified examples divided by the total number of classified examples. In terms of the confusion matrix, it is given by:

$$\text{accuracy} \stackrel{\text{def}}{=} \frac{TP + TN}{TP + TN + FP + FN}.$$

- ✓ Accuracy is a useful metric when errors in predicting all classes are equally important. In case of the spam/not spam, this may not be the case.
- ✓ For example, you would tolerate false positives less than false negatives.
- ✓ A false positive in spam detection is the situation in which your friend sends you an email, but the model labels it as spam and doesn't show you.
- ✓ On the other hand, the false negative is less of a problem: if your model doesn't detect a small percentage of spam messages, it's not a big deal.

Cost sensitive accuracy

- ✓ For dealing with the situation in which different classes have different importance, a useful metric is cost-sensitive accuracy.
- ✓ To compute a cost-sensitive accuracy, you first assign a cost (a positive number) to both types of mistakes: FP and FN.
- ✓ You then compute the counts TP, TN, FP, FN as usual and multiply the counts for FP and FN by the corresponding cost before calculating the accuracy

Area under the ROC curve (AUC)

- ✓ The ROC curve (stands for "receiver operating characteristic", the term comes from radar engineering) is a commonly used method to assess the performance of classification models.
- ✓ ROC curves use a combination of the true positive rate (defined exactly as recall) and false positive rate (the proportion of negative examples predicted incorrectly) to build up a summary picture of the classification performance.
- ✓ The true positive rate (TPR) and the false positive rate (FPR) are respectively defined as,

$$\text{TPR} = \frac{\text{TP}}{(\text{TP} + \text{FN})} \text{ and } \text{FPR} = \frac{\text{FP}}{(\text{FP} + \text{TN})}.$$

Area under the ROC curve (AUC)

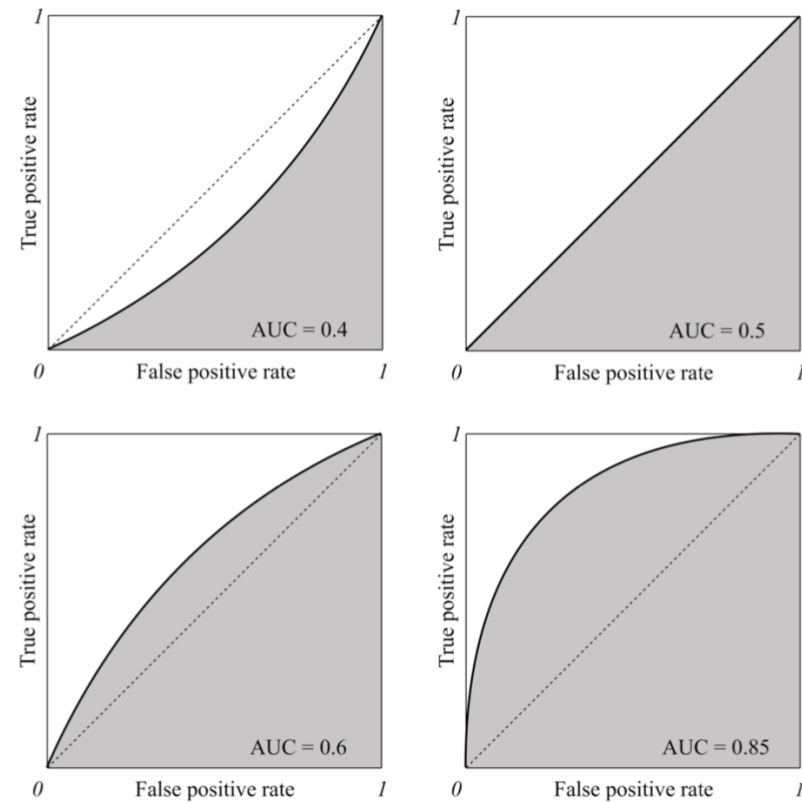


Figure 5.3: The area under the ROC curve (shown on gray).

Area under the ROC curve (AUC)

- ✓ ROC curves can only be used to assess classifiers that return some confidence score (or a probability) of prediction.
- ✓ For example, logistic regression, neural networks, and decision trees (and ensemble models based on decision trees) can be assessed using ROC curves.
- ✓ To draw a ROC curve, you first discretize the range of the confidence score.
- ✓ If this range for a model is $[0, 1]$, then you can discretize it like this: $[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$.
- ✓ Then, you use each discrete value as the prediction threshold and predict the labels of examples in your dataset using the model and this threshold.

Area under the ROC curve (AUC)

- ✓ For example, if you want to compute TPR and FPR for the threshold equal to 0.7, you apply the model to each example, get the score, and, if the score is higher than or equal to 0.7, you predict the positive class; otherwise, you predict the negative class.
- ✓ Look at the illustration in Figure 5.3. It's easy to see that if the threshold is 0, all our predictions will be positive, so both TPR and FPR will be 1 (the upper right corner). On the other hand, if the threshold is 1, then no positive prediction will be made, both TPR and FPR will be 0 which corresponds to the lower left corner.
- ✓ The higher the area under the ROC curve (AUC), the better the classifier. A classifier with an AUC higher than 0.5 is better than a random classifier.
- ✓ If AUC is lower than 0.5, then something is wrong with your model.
- ✓ A perfect classifier would have an AUC of 1.

Area under the ROC curve (AUC)

- ✓ Usually, if your model behaves well, you obtain a good classifier by selecting the value of the threshold that gives TPR close to 1 while keeping FPR near 0. ROC curves are popular because they are relatively simple to understand, they capture more than one aspect of the classification (by taking both false positives and negatives into account) and allow visually and with low effort comparing the performance of different models.

Hyper parameter tuning

- ✓ Hyperparameters aren't optimized by the learning algorithm itself. The data analyst has to "tune" hyperparameters by experimentally finding the **best combination** of values, one per hyperparameter.
- ✓ Hyper parameters can be tuned by following approaches,
 - ✓ Cross validation
 - ✓ Grid search
 - ✓ Random search
 - ✓ Bayesian techniques

Cross validation

- ✓ When you don't have a decent validation set to tune your hyperparameters on, the common technique that can help you is called cross-validation.
- ✓ When you have few training examples, it could be prohibitive to have both validation and test set. You would prefer to use more data to train the model.
- ✓ In such a case, you only split your data into a training and a test set. Then you use cross-validation on the training set to simulate a validation set.
- ✓ Cross-validation works like follows. First, you fix the values of the hyperparameters you want to evaluate. Then you split your training set into several subsets of the same size. Each subset is called a fold. Typically, five-fold cross-validation is used in practice. With five-fold cross-validation, you randomly split your training data into five folds: $\{F1, F2, \dots, F5\}$.

Cross validation

- ✓ Each F_k , $k = 1, \dots, 5$ contains 20% of your training data.
- ✓ Then you train five models as follows,
 - ✓ To train the first model, f_1 , you use all examples from folds F_2 , F_3 , F_4 , and F_5 as the training set and the examples from F_1 as the validation set.
 - ✓ To train the second model, f_2 , you use the examples from folds F_1 , F_3 , F_4 , and F_5 to train and the examples from F_2 as the validation set.
 - ✓ You continue building models iteratively like this and compute the value of the metric of interest on each validation set, from F_1 to F_5 .
 - ✓ Then you average the five values of the metric to get the final value.
- ✓ You can use grid search with cross-validation to find the best values of hyperparameters for your model. Once you have found these values, you use the entire training set to build the model with these best values of hyperparameters you have found via cross-validation.
- ✓ Finally, you assess the model using the test set.

Grid search

- ✓ Grid search is the most simple hyperparameter tuning technique.
- ✓ Let's say you train an SVM and you have two hyperparameters to tune:
 - ✓ the penalty parameter C (a positive real number) and
 - ✓ the kernel (either "linear" or "rbf").
- ✓ If it's the first time you are working with this particular dataset, you don't know what is the possible range of values for C . The most common trick is to use a logarithmic scale. For example, for C you can try the following values: [0.001, 0.01, 0.1, 1, 10, 100, 1000].
- ✓ In this case you have 14 combinations of hyperparameters to try: [(0.001, "linear"), (0.01, "linear"), (0.1, "linear"), (1, "linear"), (10, "linear"), (100, "linear"), (1000, "linear"), (0.001, "rbf"), (0.01, "rbf"), (0.1, "rbf"), (1, "rbf"), (10, "rbf"), (100, "rbf"), (1000, "rbf")].
- ✓ You use the training set and train 14 models, one for each combination of hyperparameters.

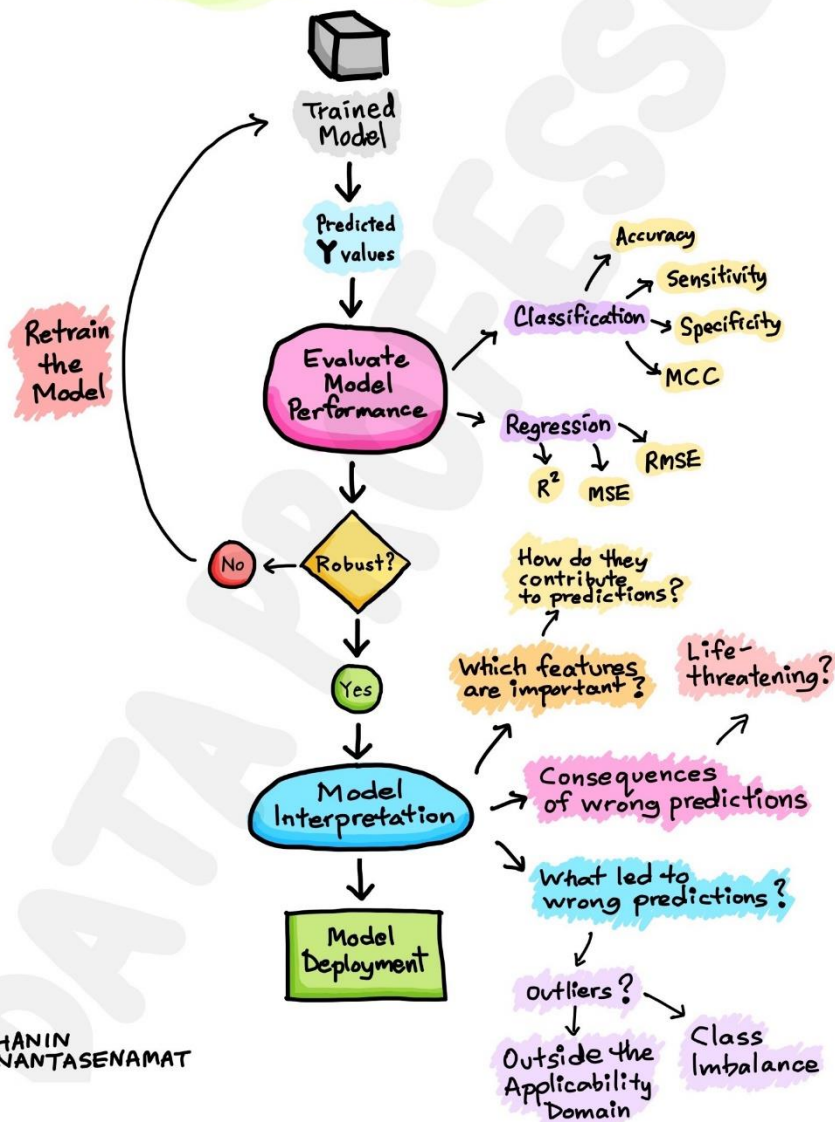
Grid search

- ✓ Then you assess the performance of each model on the validation data using one of the metrics we discussed in the previous section (or some other metric that matters to you).
- ✓ Finally, you keep the model that performs the best according to the metric.
- ✓ Once the best pair of hyperparameters is found, you can try to explore the values close to the best ones in some region around them.
- ✓ Sometimes, this can result in an even better model. Finally, you assess the selected model using the test set.
- ✓ As you could notice, trying all combinations of hyperparameters, especially if there are more than a couple of them, could be **time-consuming**, especially for **large** datasets.
- ✓ There are more efficient techniques, such as **random search** and **Bayesian hyperparameter optimization**.

Other tuning methods

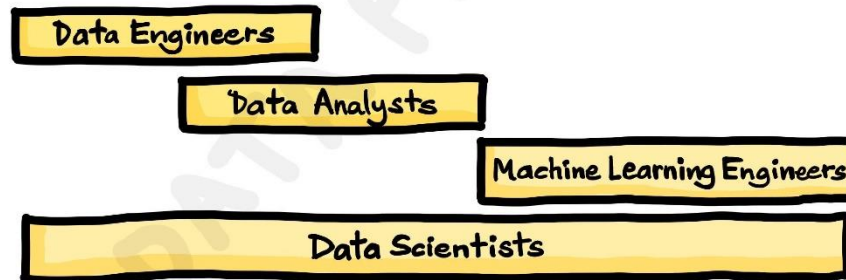
- ✓ **Random search** differs from grid search in that you no longer provide a discrete set of values to explore for each hyperparameter; instead, you provide a statistical distribution for each hyperparameter from which values are randomly sampled and set the total number of combinations you want to try.
- ✓ **Bayesian techniques** differ from random or grid search in that they use past evaluation results to choose the next values to evaluate. The idea is to limit the number of expensive optimizations of the objective function by choosing the next hyperparameter values based on those that have done well in the past.
- ✓ There are also **gradient-based** techniques, **evolutionary** optimization techniques, and other algorithmic hyperparameter tuning techniques.
- ✓ Most modern machine learning libraries implement one or more such techniques. There are also hyperparameter tuning libraries that can help you to tune hyperparameters of virtually any learning algorithm, including ones you programmed yourself.

MODEL INTERPRETABILITY



BY: CHANIN
NANTASENAMAT

DIFFERENT DATA SCIENCE ROLES EXPLAINED



References:

- ✓ <http://themlbook.com/>
- ✓ <https://www.amazon.com/Approaching-Almost-Machine-Learning-Problem-ebook>

Thanks for your attention