# The Machine Learning Landscape [1]

## Types of Machine Learning Systems [7]

### Supervised vs Unsupervised

* **Supervised Learning**  
  The training set you feed to the algorithm includes the desired solutions, called *labels*.
* **Unsupervised Learning**  
  The training set is unlabeled. The system tries to learn without a teacher
* **Semi-supervised Learning**  
  The algorithms can deal with data that’s partially labeled. Most of these types of algorithms are combinations of unsupervised and supervised algorithms
* **Reinforcement Learning**The learning system (*agent*) can observe the environment, select and perform actions, and get *rewards* or *penalties* in return. It must then learn by itself what is the best strategy, called a *policy*, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.

### Batch and Online Learning

* **Batch Learning**  
  The system is incapable of learning incrementally: it must be trained using all the available data, usually done offline, and then once the model is trained, it is then launched into production and runs without learning anymore (*offline learning*)
* **Online Learning**  
  You train the system incrementally by feeding it data instances sequentially, either individually or in small groups called *mini batches*. Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives.
  + Great for systems that receive data as a continuous flow (e.g., stock prices) and need to adapt to change rapidly or autonomously.
  + Also good with limited computing resources: once an online learning system has learned about new data instances, it does not need them anymore and you can discard them.
  + Online learning algorithms can be used to train systems on huge datasets that cannot fit in one machine’s main memory (*out-of-core learning*).
  + One important parameter of online learning systems is how fast they should adapt to changing data (*learning rate*)
  + If bad data is fed to the system, the system’s performance will gradually decline.

### Instance-Based vs. Model-Based learning

How do machine learning systems *generalize*?

* **Instance-based learning**  
  The system learns the examples by heart, then generalizes to new cases by using a similarity measure to compare them to the learned examples (or a subset).
* **Model-based learning**Build a model of these examples and then use that model to make *predictions*.
  + The model takes parameters.
  + To determine how you can know which values make your model perform best, you need to specify a performance measure.
    - *Utility/fitness function* measures how good your model is.
    - *Cost function* that measures how bad it is.

## Main Challenges of Machine Learning [23]

* Insufficient Quantity of Training Data
* Nonrepresentative Training Data
* Poor Quality Data
* Irrelevant Features
* Overfitting
* Underfitting

## Testing and Validation [30]

Split data into two sets: *training* and *test* sets. The error rate on new cases is called the *generalization error* (or *out-of-sample error*), and by evaluating your model on the test set, you get an estimate of this error. This value tells you how well your model will perform on instances it has never seen before.

### Hyperparameter Tuning and Model Selection

**Situation**  
You want to compare models, so you train both and compare how well they generalize using the test set. Then you apply some regularization to avoid overfitting. To pick the regularization hyperparameter, you train 100 different models with different values. You pick the model with the lowest generalization error, and deploy to production, but the error in production is much higher.

**Problem**  
You measured the generalization error multiple times on the test set, and you adapted the model and hyperparameters to produce the best model for that particular set, which means the model is unlikely to perform well on new data.

Solutions

* Holdout validation – You simply hold out part of the training set to evaluate several candidate models and select the best one. The new held-out set is called the *validation set* (or sometimes the *development set*). You train multiple models on the reduced training set and select the model that performs best on the validation set. After this holdout validation process, you train the best model on the full training set (including the validation set), and this gives you the final model. Lastly, you evaluate this final model on the test set to get an estimate of the generalization error.
  + *Cross validation* – Use many small validation sets. Each model is evaluated once per validation set after it is trained on the rest of the data. By averaging out all the evaluations of a model, you get a much more accurate measure of its performance.

### Data Mismatch

Training data might not be representative of the data that will be used in production.

## Exercises [33 – Sol 719]

* *What is a test set, and why would you want to use it?*  
  A test set is used to estimate the generalization error that a model will make on new instances, before the model is launched in production.
* *What is the purpose of a validation set?*  
  A validation set is used to compare models. It makes it possible to select the best model and tune the hyperparameters.
* *What is the train-dev set, when do you need it, and how do you use it?*The train-dev set is used when there is a risk of mismatch between the training data and the data used in the validation and test datasets (which should always be as close as possible to the data used once the model is in production). The train-dev set is a part of the training set that’s held out (the model is not trained on it). The model is trained on the rest of the training set, and evaluated on both the train-dev set and the validation set. If the model performs well on the training set but not on the train-dev set, then the model is likely overfitting the training set. If it performs well on both the training set and the train-dev set, but not on the validation set, then there is probably a significant data mismatch between the training data and the validation + test data, and you should try to improve the training data to make it look more like the validation + test data.
* *What can go wrong if you tune hyperparameters using the test set?*  
  If you tune hyperparameters using the test set, you risk overfitting the test set, and the generalization error you measure will be optimistic (you may launch a model that performs worse than you expect).

# End-to-End Machine Learning Project [35]

## Look at the Big Picture [37]

* Frame the Problem – The first question to ask your boss is what exactly the busines object is.
  + Data pipeline – A sequence of data processing components.
  + Should also ask what the current solution is.
* Select a Performance Measure
  + Root Mean Square Error (RMSE)
    - Corresponds to the Euclidean norm – the notion of distance you are familiar with.
    - Also called the *l2 norm*.
  + Mean Absolute Error (MAE)
    - Sometimes called the Manhattan norm because it measures the distance between two points in a city if you can only travel along orthogonal city blocks.
    - Also called the *l1 norm*.
  + The higher the norm index, the more it focuses on large values and neglects small ones.
* Check the assumptions
  + List and verify the assumptions.

## Get the Data [42]

* Create the Workspace
  + Create an Isolated Environment
* Download the data
* Take a quick look at the data structures
  + .info()
  + .value\_counts()
  + .describe()
    - *Standard deviation* – the square root of the *variance*, which is the average of the squared deviation from the mean. When a feature has a bell-shaped *normal distribution* (*Gaussian distribution*), which is very common, the “68-95-99.7” rule applies: about 68% of the values fall within 1std of the mean, 95% within 2std, and 99.7% within 3std.
    - Percentiles
    - Quartiles
  + .hist()
* Create a Test Set
  + Protect from *data snooping bias*
  + .train\_test\_split()
  + np.random.seed(42)
  + .cut()
    - Bin numerical attribute into categories
  + StratifiedShuffleSplit

## Discover and visualize the data to gain insights [56]

* Looking for correlations
  + .corr()
    - Computes *standard correlation coefficient* (*Pearson’s r*)
    - Only measures linear correlations.
  + .scatter\_matrix()
    - Plots every numerical attribute against every other numerical attribute.
* Experimenting with Attribute Combinations
  + Computing logs, exponents, combining attributes to make new ones

## Prepare the data for Machine Learning algorithms [62]

* Data Cleaning
  + Handle missing values
    - SimpleImputer
* Handling Text and Categorical Attributes
  + OneHotEncoder()
* Feature Scaling
  + Min-max scaling (normalization)
    - Values are shifted and rescaled so that they end up ranging from 0 to 1.
    - *MinMaxScaler*
  + Standardization
    - Subtract the mean, and then divide by the standard deviation
    - *StandardScaler*
* Transformation Pipelines
  + *Pipeline*
  + *ColumnTranformer*

## Scikit-Learn Design [64]

### Consistency

All objects share a consistent and simple interface

#### Estimators

Any object that can estimate some parameters based on a dataset is called an *estimator* (e.g., an *imputer* is an estimator). The estimation itself is performed by the *.fit()* method, and it takes only a dataset as a parameter (or two for supervised learning algorithms; the second dataset contains the labels). Any other parameters needed to guide the estimation process is considered a hyperparameter (such as an *imputer’s strategy*), and it must be set as an instance variable (generally via a constructor parameter).

#### Transformers

Some estimators (such as an *imputer*) can also transform a dataset; these are called *transformers*. Once again, the API is simple: the transformation is performed by the *.transform()* method with the dataset to transform as a parameter. It returns the transformed dataset. This transformation generally relies on the learned parameters, as is the case for an *imputer*. All transformers also have a convenience method called *.fit\_tranform()* that is equivalent to calling *.fit()* and then *.transform()* (but sometimes *.fit\_transform()* is optimized and runs much faster).

#### Predictors

Finally, some estimators, given a dataset, are capable of making predictions; they are called predictors. For example, the *LinearRegression* model is a predictor. A predictor has a *.predict()* method that takes a dataset of new instances and returns a dataset of corresponding predictions. It also has a *.score()* method that measures the quality of the predictions, given a test set (and the corresponding labels, for supervised learning algorithms). Some predictors also provide methods to measure the confidence of their predictions.

### Inspection

All the estimator’s hyperparameters are accessible directly via public instance variables (e.g., *imputer.strategy*), and all the estimator’s learned parameters are accessible via public instance variables with an underscore suffix (e.g., *imputer.statistics\_*).

### Non-proliferation of classes

Datasets are represented as NumPy arrays or SciPy sparse matrices, instead of homemade classes. Hyperparameters are just regular Python strings or numbers.

### Composition

Existing building blocks are reused as much as possible. For example, it is easy to create a Pipeline estimator from an arbitrary sequence of transformers followed by a final estimator.

### Sensible defaults

Scikit-Learn provides reasonable default values for most parameters, making it easy to quickly create a baseline working system.

## Select and Train a Model [72]

* Training and Evaluating the Training Set
  + *sklearn.metrics.mean\_squared\_error*
* Better Evaluation Using Cross-Validation
  + *sklearn.model\_selection.cross\_val\_score*
    - Expects a utility function (greater is better) rather than a cost function (lower is better).
* You should save every model you experiment with.
  + *pickle*
  + *joblib*

## Fine-tune your model [75]

* Grid Search
  + *sklearn.model\_selection.GridSearchCV*
  + Tell which hyperparameters to experiment with and what values to try out, and it will use cross-validation to evaluate all the possible combinations.
  + When you have no idea what value a hyperparameter should have, a simple approach is to try out consecutive powers of 10.
  + *.best\_params\_* attribute
  + *.best\_estimator\_* attribute
  + *.cv\_results\_* attribute for all evaluation scores
* Randomized search
  + *sklearn.model\_selection.RandomizedSearchCV*
* Ensemble Methods
* Analyze the Best Models and Their Errors
  + E.g. *RandomForestRegressor* has *feature\_importances\_*
* Evaluate your system on the test set

## Launch, monitor, and maintain your system [80]

## Exercises [84]

In Notebook