# Chapter 1- The Machine Learning Landscape

ML is used to solve problems:

* Existing solutions require lot of hand-tuning or long lists of rules
* Complex problems for which there is no good solution at all using a traditional approach
* Fluctuating environments: ML can adapt to new data
* Getting insights about complex problems and large amounts of data.

**Types of ML Systems: -**

Some of the most important **Supervised** machine learning algos are:

* k-Nearest Neighbours
* Linear Regression
* Logistic Regression
* Support Vector Machines (SVMs)
* Decision Trees and Random Forests
* Neural Networks

Most important **Unsupervised** ML algos are:

* Clustering (k-means, Hierarchical Cluster Analysis, Expectation Maximization)
* Visualization and dimensionality reduction (PCA, Kernel PCA, t-SNE, Locally-Linear Embedding)
* Association rule learning (Apriori, Eclat)
* Clustering algorithms are useful when we want to subdivide data into smaller groups
* Visualization and dimensionality reduction is useful when we feed a lot of complex and unlabelled data and output 2D or 3D. Another task is to simplify the data without loosing too much information. Example: A car’s mileage may be correlated with its age. Here, dimensionality reduction algo will merge them into a single feature which represents car’s wear and tear.
* Association rule learning where we dig into large amounts of data and discover interesting relations between the attributes.

**Semi supervised** learning:

Algos which can work with partially labelled data. Ex: Some photo-hosting services, as Google Photos are good examples. Once we upload a family photo it automatically recognizes the same person. Now, the system asks us to label the person. Thus, it labels everyone in the photo.

**Reinforcement** learning:

There is a learning system called *Agent* which observes the environment, selects, and performs actions called *rewards.* It learns by itself using the best strategy called *policy*. A policy defines what action the agent should choose when it is in each situation.

Example: DeepMind’s AlphaGo program is a good example of Reinforcement learning, where it learned its winning policy by analysing millions of games and then playing many games against itself

**Batch and Online learning:**

**Batch learning:**

It is a conventional way of training an ML model where we consider the entire data for training a model. Here, the model is being trained offline in system by a developer and then the trained model is fed into the PROD environment.

**Online learning:**

We train the system incrementally by feeding the data instances sequentially, either individually or by small groups called *mini-batches*. Each learning is fast and cheap, so the system can learn about new data on the fly as it arrives. It is good for systems that receive continuous flow of data like stock prices

# Chapter 2 – End-to-End Machine Learning Project

We need to follow below steps while solving any ML project.

1. Frame the problem and look at the big picture
2. Get the data
3. Discover and visualize the data to gain insights
4. Prepare the data for ML algos
5. Select a model and train it
6. Fine-tune your model
7. Present your solution
8. Launch, monitor and maintain your system

**Frame the problem and look at the big picture**

* Define the objective in business terms
* How will your solution be used?
* What are the current solutions/workarounds (if any)?
* How should you frame this problem (supervised/unsupervised, online/offline)?
* How should performance be measured?
* Is the performance measure aligned with the business objective?
* What would be the minimum performance need to reach the business objective?
* What are comparable problems? Can you reuse experience or tools?
* Is human expertise available?
* How would you solve the problem manually?
* List the assumptions you made so far, verify those assumptions

**Get the data**

* List the data you need and how much you need
* Find and document where you can get the data
* Check how much space it will take
* Check legal obligations, and get authorization if necessary.
* Get access authorizations
* Create a workspace (with enough storage space)
* Get the data
* Convert the data to format you can easily manipulate
* Ensure sensitive information is deleted or protected
* Check the size and type of data (time series, sample, geographical etc).
* Sample a test set, put it aside and never look at it

**Explore the Data**

* Create a copy of the data for exploration (sampling it down to a manageable size if necessary)
* Create a Jupyter notebook to keep a record of your data exploration
* Study the attribute and its’ characteristics: -
* Name
* Type (categorical, int/float, text, structured, bounded/unbounded, etc.)
* % of missing values
* Noisiness and type of noise (stochastic, outliers, rounding errors, etc.)
* Possibly useful for the task?
* Type of distribution (Gaussian, uniform, logarithmic etc)
* For supervised learning tasks, identify the target attribute(s).
* Visualize the data
* Study the correlations between attributes
* Study how you would solve the problem manually
* Identify the promising transformations you want to apply
* Extract the data that would be useful
* Document what you have learned

**Prepare the Data**

**Notes:**

* Create copies of the original data
* Write functions for all data transformations, for five reasons:
* We can easily prepare the data next time we get a fresh dataset
* We can apply these transformations in future projects
* To clean and prepare the test set
* To clean and prepare new data instance once the solution is live
* To make it easy to treat your preparation choices as hyperparameters

1. Data Cleaning:

* Fix or remove outliers
* Fill in missing values (e.g., with zero, mean, median) or drop their rows

1. Feature selection: Drop the attributes that provide no useful information for the task
2. Feature engineering:

* Discretize continuous features
* Decompose features (e.g., categorical, date/time)
* Add promising transformations of features (e.g., log(x), sqrt(x), x^2 etc)
* Aggregate features into promising new features

1. Feature scaling: standardize or normalize features

**Short-List Promising Models**

**Notes:**

* If the data is huge, we want sample smaller training sets so we can train many different models in a reasonable time
* Once again, we try to automate these steps as much as possible

1. Train many quick and dirty models from different categories (e.g. linear, naïve bayes, SVM, Random Forests, Neural Net, etc) using standard parameters.
2. Measure and compare their performance – For each model, use a k-fold cross-validation and compute the mean and std. deviation of performance measure on the k-folds.
3. Analyse the most significant variables for each algorithm
4. Analyse the type of errors the models make

* What data would a human have used to avoid these errors

1. Have a quick round of feature selection and engineering
2. Have one or two more quick iterations of the five previous steps
3. Short-list the top three to five most promising models, preferring models that make different types of errors

**Fine-Tune the System**

**Notes:**

* We want as much data as possible for this step, especially as we move toward the end of fine-tuning
* We can automate whatever we can

1. Fine-tune the hyperparameters using cross-validation

* Treat our data transformation choices as hyperparameters, especially when we are not sure about them (e.g., replace missing value with 0 or median value)
* Unless there are few hyperparameters to explore, prefer random search over grid search. If training is very long, we prefer a Bayesian optimization approach

1. Try Ensemble methods. Combining your best models will often perform better than running them individually
2. Once we are confident about final model, measure its performance on the test set to estimate the generalization error.

**Present your solution**

1. Document what you have done
2. Create a nice presentation – Make sure to highlight big picture first
3. Explain why our solution achieves the business objective
4. Do not forget to present interesting points we noticed along the way

* Describe what worked and what did not
* List your assumptions and your system’s limitations

1. Ensure our key findings are communicated through beautiful visualizations or easy-to-remember statements (e.g., the median income is the number-one predictor of housing prices)

**Launch!**

1. Get your solution ready for production (plug into production data inputs, write unit tests etc...)
2. Write monitoring code to check for system’s performance at regular intervals and trigger alerts when it drops

* Beware of slow degradation too: models tend to “rot” as data evolves
* Measuring performance may require a human pipeline
* Monitor our inputs quality

1. Retrain the model on fresh data, automate as much as possible

**Well-known open-source repositories:**

*Popular data repositories*

* UCI ML repository(<http://archive.ics.uci.edu/ml/>)
* Kaggle datasets(<https://www.kaggle.com/datasets>)
* Amazon’s AWS datasets(<http://aws.amazon.com/fr/datasets/>)

*Meta portals*

* <http://dataportals.org/>
* <http://opendatamonitor.eu/>
* <http://quandl.com/>

*Other open data repositories*

* Wiki’s list(<https://goo.gl/SJHN2k>)
* Quora.com qs(<http://goo.gl/zDR78y>)
* Datasets reddit(<https://www.reddit.com/r/datasets>)

**Look at the Big Picture**

As a data scientist, the task to perform is to build a model consisting of housing prices in California using California census data.

Metrics/Attributes: Population, median income, median housing price for each block group

A block group are the smallest geographical unit for which the US Census Bureau publishes sample data (population of 600 to 3,000 people)

Aim: Model should learn from this data and predict the median housing price in any district given all the metrics

**Frame the Problem**

Question to ask your boss is the business objective. End goal is not about model building.

* How does the company expect to use and benefit from this model?
* What algorithms we can select?
* What performance measure we will use to evaluate the model
* How much effort we should spend tweaking it?

Boss answers questions like, model output being fed is to predict a district’s median housing price which will be fed in other ML systems and further determining whether it is worth investing in each area or not. Stating this, it also conveys that it directly affects revenue.

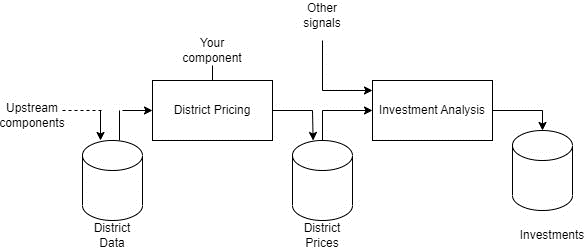


Fig. ML pipeline for real estate investments

Questions to be posed further are:

* How the current solution looks like?
* It always gives a reference performance, as well as insights on how to solve the problem.
* Boss responds by saying: Currently, team gathers up-to-date information about a district (excluding median house prices) and then use complex rules to come up with an estimate. These estimates are costly and time-consuming and their typical error rate is around 15%.

From a DS developer point of view:

**Frame the problem**

* Is it a supervised, unsupervised or Reinforcement learning?
* Is it classification or regression task?
* Do we need to use a batch learning or online learning?

Answers:

* Training data is labelled (*supervised learning*)
* Finding median housing price (*regression task*). More specifically multivariate regression task, as it involves multiple features
* There is no continuous flow of data and data being loaded is small in memory. Hence, plain *batch learning* is recommended here.

**Selecting a Performance measure**

Usually, RMSE measures the *standard deviation* of errors the system makes in its’ predictions. However, it doesn’t work in case of outliers where the MAE(*Mean Absolute Error*) is preferred.

<https://stephenallwright.com/rmse-vs-mae/#:~:text=It%20comes%20down%20to%20the,interpretable%20value%20then%20use%20MAE>.

**Check the Assumptions**

We also need to double check if the problem being solved is a regression or classification task before carrying out the activity.

**Get the Data**

In real-time we may get data from Relational DB or multiple tables/documents/files. Often, we must use credentials or authorizations to familiarize with the data schema.