The Machine Learning Landscape

What is Machine Learning?

- Machine learning is the science of programming computers so they can learn from data.
- A field of study that gives computers the ability to learn without being explicitly programmed.
- A computer program is said to learn from experience **E** with respect to some task **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improves with experience **E**.
- The examples that the system uses to learn is called the **training set**, Each training examples is called **training instance** or **sample**.

Why use Machine Learning?

- Problems for which existing solutions require a lot of fine-tuning or have a long list of complex rules that are pretty hard to maintain if programmed using traditional programming techniques ex: spam filter.
- Problems that either are too complex for traditional approaches or have no known algorithm ex: speech recognition.
- Fluctuating environments: a machine learning system can adapt to new data.
- Machine learning can help humans learn; ML algorithms can be inspected to see what they have learned (for some algorithms might be tricky).
- Applying ML techniques to dig into large amounts of data can help discover patterns that were not immediately apparent, This is called **Data Mining**

Types of Machine Learning Systems

Supervised/ Unsupervised Learning/ Semi Supervised/ Self Supervised/ Reinforcement Learning

Supervised

- In supervised learning, the training set you feed to the algorithm includes the desired solutions, called **labels.**
- A typical supervised learning task is **classification** Ex: spam filter.
- Another typical task is to predict a target numeric value called **Regression**
- some regression models can be used for classification as well, and vice versa. For example, **logistic regression** is commonly used for classification, as it can output a value that corresponds to the probability of belonging to a given class
- The words **target** and **label** are generally treated as synonyms in supervised learning, but **target** is more common in regression tasks and **label** is more common in classification tasks.
- **features** are sometimes called **predictors** or **attributes**. These terms may refer to individual sample or to all samples.
- Some supervised learning algorithms: KNN, Linear Regression, Logistic Regression, SVM, Decision Trees, Random Forests and Neural Networks.

Unsupervised

- In unsupervised learning, the training data is unlabeled, The system tries to learn without a teacher.
- **clustering** algorithm tries to detect groups of similar visitors, At no point do you tell the algorithm which group a visitor belongs to: it finds those connections without your help.
- **Visualization** algorithms are also good examples of unsupervised learning: you feed them a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted.
- **Dimensionality reduction** task, in which the goal is to simplify the data without losing too much information.
- **Anomaly detection** task, detecting outliers in a dataset.

- **Novelty Detection** task, it aims to detect new instances that look different from all instances in the training set.
- **Association Rule Learning** task, digging into large amounts of data and discover interesting relations between attributes.

Semi-supervised

- Labeling data is usually time-consuming and costly, you will often have plenty of unlabeled instances, and few labeled instances. Some algorithms can deal with data that's partially labeled. This is called **semi-supervised learning.**
- Most semi-supervised learning algorithms are combinations of unsupervised and supervised algorithms.

Self-supervised

- Another approach to machine learning involves actually generating a fully labeled dataset from a fully unlabeled one. Again, once the whole dataset is labeled, any supervised learning algorithm can be used. This approach is called **self-supervised learning**.
- Transferring knowledge from one task to another is called **transfer learning**, and it's one of the most important techniques in machine learning today, especially when using deep neural networks

Reinforcement Learning

- Reinforcement learning is a very different beast. The learning system, called an **agent** in this context, can observe the environment, select and perform actions, and get **rewards** in return (or **penalties** in the form of negative rewards). 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

- In **batch learning**, the system is incapable of learning incrementally: it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline.
- First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called **offline learning**.

Online Learning

- In **online learning**, you train the system incrementally by feeding it data instances sequentially, either individually or in small groups called minibatches. Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives.
- **Online learning** is useful for systems that need to adapt to change extremely Rapidly. It is also a good option if you have limited computing resources
- **Online learning** algorithms can be used to train models on huge datasets that cannot fit in one machine's main memory (this is called **out-of-core learning**). The algorithm loads part of the data, runs a training step on that data, and repeats the process until it has run on all of the data.
- One important parameter of **online learning** systems is how fast they adapt to changing data: this is called the **learning rate**.
- If you set a high **learning rate**, then your system will rapidly adapt to new data, but it will also tend to quickly forget the old data. Conversely, if you set a low **learning rate**, the system will have more inertia; it will learn more slowly, but it will also be less sensitive to noise in the new data or to sequences of nonrepresentative data points (outliers).

Instance Based Versus Model-Based Learning

- One more way to categorize machine learning systems is by how they **generalize**.
- There are two main approaches to generalization: **instance-based learning** and **model-based learning**.

Instance-Based Learning

- The most trivial form of learning is simply to learn by heart, then generalizes to new cases by using a similarity measure to compare them to the learned examples.

Model-Based Learning

- Another way to generalize from a set of examples is to build a model of these examples and then use that model to make **predictions**.
- You can either define a **utility function** (or **fitness function**) that measures how good your model is, or you can define a **cost function** that measures how bad it is.
- In summary:
  - You studied the data.
  - You selected a model.
  - You trained it on the training data (i.e., the learning algorithm searched for the model parameter values that minimize a **cost function**).
  - Finally, you applied the model to make predictions on new cases (this is called **inference**), hoping that this model will **generalize** well.

Main Challenges if Machine Learning

- Since your main task is to select a model and train it on some data, the two things that can go wrong are **"bad model"** and **"bad data".**

Insufficient Quantity of Training Data

- Machine learning takes a lot of data for most machine learning algorithms to work properly.
- Even for very simple problems you typically need thousands of examples, and for complex problems such as image or speech recognition you may need millions of examples (unless you can reuse parts of an existing model).

The Unreasonable Effectiveness of Data

- In a famous paper published in 2001 showed that very different machine learning algorithms, including fairly simple ones, performed almost identically well on a complex problem of natural language disambiguation☐ once they were given enough data

Nonrepresentative Training Data

- In order to **generalize** well, it is crucial that your training data be **representative** of the new cases you want to **generalize** to. This is true whether you use **instance-based learning** or **model-based learning**.
- if the sample is too small, you will have sampling noise (i.e., **nonrepresentative data** as a result of chance), but even very large samples can be nonrepresentative if the sampling method is flawed. This is called **sampling bias**.

Poor-Quality Data

- If your training data is full of errors, outliers, and noise (e.g., due to poor-quality measurements), it will make it harder for the system to detect the underlying patterns, so your system is less likely to perform well.

- It is often well worth the effort to spend time cleaning up your training data.

Irrelevant Features

- Your system will only be capable of learning if the training data contains enough relevant features and not too many irrelevant ones.
- A critical part of the success of a machine learning project is coming up with a good set of features to train on.
- This process, called **feature engineering**, involves the following steps:
  - Feature selection (selecting the most useful features to train on among existing features)
  - Feature extraction (combining existing features to produce a more useful one)
  - Creating new features by gathering new data

Overfitting the Training Data

- In machine learning this **overfitting** means that the model performs well on the training data, but it does not generalize well.
- **Complex models** such as deep neural networks can detect subtle patterns in the data, but if the training set is noisy, or if it is too small, which introduces sampling noise, then the model is likely to detect patterns in the noise itself.
- **Overfitting** happens when the model is too complex relative to the amount and noisiness of the training data. Here are possible solutions:
  - **Simplify** the model by selecting one with fewer parameters (e.g., a linear model rather than a high-degree polynomial model)
  - By **reducing** the number of **attributes** in the training data
  - By **constraining** the model.
  - **Gather** more training data.
  - **Reduce** the noise in the training data (e.g., fix data errors and remove outliers).
- **Constraining** a model to make it simpler and reduce the risk of **overfitting** is called **Regularization**.
- You want to find the right balance between fitting the training data perfectly and keeping the model simple enough to ensure that it will **generalize** well.
- The amount of regularization to apply during learning can be controlled by a **Hyperparameter**.
- A **hyperparameter** is a parameter of a learning algorithm (not of the model). As such, it is not affected by the learning algorithm itself; it must be set prior to training and remains constant during training.

Underfitting the Training Data

- **Underfitting** is the opposite of **overfitting**: it occurs when your model is too simple to learn the underlying structure of the data.
- Here are the main options for fixing this problem:
  - Select a more **powerful** model, with more parameters.
  - Feed better **features** to the learning algorithm (**feature engineering**).
  - **Reduce** the **constraints** on the model (for example by reducing the **regularization** hyperparameter).

Testing and Validating

- The only way to know how well a model will **generalize** to new cases is to actually try it out on new cases.
- A better option is to **split** your data into **two sets**: the **training set** and the **test set**.
- You train your model using the **training set**, and you test it using the **test set**.
- 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.

- If the **training error** is **low** (i.e., your model makes few mistakes on the training set) but the **generalization error** is **high**, it means that your model is **overfitting** the training data.

Hyperparameter Tuning and Model Selection

- If you are hesitant between different types of models, one option is to train both and compare how well they generalize using the test set.
- If you are choosing the value of the **regularization hyperparameter** and you find the best **hyperparameter** value that produces a model with the lowest **generalization error**, but it doesn't perform as well as expected and produces more errors.
- The problem is that 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. This means that the model is unlikely to perform as well on new data.
- A common solution to this problem is called **Holdout validation**, you simply hold out part of the training set to evaluate several candidate models and select the best model.
- The new **held-out set** is called **validation set** or **development set**.
- You train multiple models with various **hyperparameters** on the reduced **training set,** and you select the model that performs best on the **validation set.**
- After this you train the best model on the full **training set** including the **validation set** and this gives the final model.
- Lastly, you evaluate the model on the **test set** to get an estimate of the **generalization error.**
- If the **validation set** is too small, then model evaluations will be imprecise: you may end up selecting a suboptimal model by mistake.
- Conversely, if the **validation set** is too large, then the remaining **training set** will be much smaller than the full training set. This is bad because it is not ideal to compare candidate models trained on a much smaller **training set**.
- One way to solve this problem is to perform repeated **cross-validation**, using many small **validation sets**. Each model is evaluated once per validation set after it is trained on the rest of the data.
- There is a drawback, however: the training time is multiplied by the number of **validation sets**.

Data Mismatch

- The most important rule to remember is that both the **validation set,** and the **test set** must be as **representative** as possible of the data you expect to use in production, so they should be composed exclusively of representative samples. you can shuffle them and put half in the validation set and half in the test set (making sure that no duplicates or near-duplicates end up in both sets).
- What if after training the model, you observe that the performance of the model on the **validation set** is disappointing, you don't whether this is because your model has overfitted the training set, or whether this is just due to the mismatch between the web pictures and the mobile app pictures.
- One solution is to hold out some of the training pictures(from the web) in another set called **train-dev set**. After the model is trained on the **training set**, evaluate it on the **train-dev set**. If it performs well, then the model is not overfitting the **training set**. If it performs poorly on the **validation set**, the problem must be coming from the **data mismatch**.
- Conversely, if the model performs poorly on the **train-dev set**, then it must have overfit the training set.