Notes for Hands-on ML\_v3

# **Chap01 - the\_machine\_learning\_landscape**

## Chap01 Exercises:

### # QnA:

1. How would you define machine learning?
   1. Machine Learning is about building systems that can learn from data. Learning means getting better at some tasks, given some performance measure.
2. Can you name four types of applications where it shines?
   1. Machine Learning is great for complex problems for which we have no algorithmic solution, to replace long lists of hand-tuned rules, to build systems that adapt to fluctuating environments, and finally to help humans learn (e.g., data mining).
3. What is a labeled training set?
   1. A labeled training set is a training set that contains the desired solution (a.k.a. a label) for each instance.
4. What are the two most common supervised tasks?
   1. The two most common supervised tasks are regression and classification.
5. Can you name four common unsupervised tasks?
   1. Common unsupervised tasks include clustering, visualization, dimensionality reduction, and association rule learning.
6. What type of algorithm would you use to allow a robot to walk in various unknown terrains?
   1. Reinforcement Learning is likely to perform best if we want a robot to learn to walk in various unknown terrains, since this is typically the type of problem that Reinforcement Learning tackles. It might be possible to express the problem as a supervised or semi-supervised learning problem, but it would be less natural.
7. What type of algorithm would you use to segment your customers into multiple groups?
   1. If you don't know how to define the groups, then you can use a clustering algorithm (unsupervised learning) to segment your customers into clusters of similar customers. However, if you know what groups you would like to have, then you can feed many examples of each group to a classification algorithm (supervised learning), and it will classify all your customers into these groups.
8. Would you frame the problem of spam detection as a supervised learning problem or an unsupervised learning problem?
   1. Spam detection is a typical supervised learning problem: the algorithm is fed many emails along with their labels (spam or not spam).
9. What is an online learning system?
   1. An online learning system can learn incrementally, as opposed to a batch learning system. This makes it capable of adapting rapidly to both changing data and autonomous systems, and of training on very large quantities of data.
10. What is out-of-core learning?
    1. Out-of-core algorithms can handle vast quantities of data that cannot fit in a computer's main memory. An out-of-core learning algorithm chops the data into mini-batches and uses online learning techniques to learn from these mini-batches.
11. What type of algorithm relies on a similarity measure to make predictions?
    1. An instance-based learning system learns the training data by heart; then, when given a new instance, it uses a similarity measure to find the most similar learned instances and uses them to make predictions.
12. What is the difference between a model parameter and a model hyperparameter?
    1. A model has one or more model parameters that determine what it will predict given a new instance (e.g., the slope of a linear model). A learning algorithm tries to find optimal values for these parameters such that the model generalizes well to new instances. A hyperparameter is a parameter of the learning algorithm itself, not of the model (e.g., the amount of regularization to apply).
13. What do model-based algorithms search for? What is the most common strategy they use to succeed? How do they make predictions?
    1. Model-based learning algorithms search for an optimal value for the model parameters such that the model will generalize well to new instances. We usually train such systems by minimizing a cost function that measures how bad the system is at making predictions on the training data, plus a penalty for model complexity if the model is regularized. To make predictions, we feed the new instance's features into the model's prediction function, using the parameter values found by the learning algorithm.
14. Can you name four of the main challenges in machine learning?
    1. Some of the main challenges in Machine Learning are the lack of data, poor data quality, nonrepresentative data, uninformative features, excessively simple models that underfit the training data, and excessively complex models that overfit the data.
15. If your model performs great on the training data but generalizes poorly to new instances, what is happening? Can you name three possible solutions?
    1. If a model performs great on the training data but generalizes poorly to new instances, the model is likely **overfitting** the training data (or we got extremely lucky on the training data). Possible solutions to overfitting are getting more data, simplifying the model (selecting a simpler algorithm, reducing the number of parameters or features used, or regularizing the model), or reducing the noise in the training data.
16. What is a test set, and why would you want to use it?
    1. 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.
17. What is the purpose of a validation set?
    1. A validation set is used to compare models. It makes it possible to select the best model and tune the hyperparameters.
18. What is the train-dev set, when do you need it, and how do you use it?
    1. 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).
    2. The train-dev set is a part of the training set that is 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.
19. What can go wrong if you tune hyperparameters using the test set?
    1. 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).

## Key Terms

1. Objective –
   1. we will take a look at the map and learn about the main regions and the most notable landmarks: supervised versus unsupervised learning and their variants, online versus batch learning, instance based versus model-based learning.
   2. Then we will look at the workflow of a typical ML project, discuss the main challenges you may face, and cover how to evaluate and fine-tune a machine learning system.
2. Machine Learning
   1. Machine learning is the science (and art) of programming computers so they can learn from data.
   2. Here is a slightly more general definition: [Machine learning is the] field of study that gives computers the ability to learn without being explicitly programmed.

—Arthur Samuel, 1959

* 1. And a more engineering-oriented one: 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.

—Tom Mitchell, 1997

1. Training set - The examples that the system uses to learn are called the training set.
2. Training instance - Each training example is called a *training instance* (or *sample*).
3. Model - The part of a machine learning system that learns and makes predictions is called a model.
4. Accuracy - the task *T* is to flag spam for new emails, the experience *E* is the *training data*, and the performance measure *P* needs to be defined; for example, you can use the ratio of correctly classified emails. This performance measure is called *accuracy*, and it is often used in classification tasks.
5. To summarize, machine learning is great for:
   1. Problems for which existing solutions require a lot of fine-tuning or long
   2. lists of rules (a machine learning model can often simplify code and perform better than the traditional approach)
   3. Complex problems for which using a traditional approach yields no good solution (the best machine learning techniques can perhaps find a solution)
   4. Fluctuating environments (a machine learning system can easily be retrained on new data, always keeping it up to date)
   5. Getting insights about complex problems and large amounts of data
6. Define Hyperparameters.
   1. The amount of regularization to apply during learning can be controlled by a *hyperparameter*.
   2. 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. If you set the regularization.
7. Examples of Applications
   1. Analyzing images of products on a production line to automatically classify them
      1. This is image classification, typically performed using convolutional neural networks (CNNs; see Chapter 14) or sometimes transformers
   2. Detecting tumors in brain scans
      1. This is semantic image segmentation, where each pixel in the image is classified (as we want to determine the exact location and shape of tumors), typically using CNNs or transformers.
   3. Automatically classifying news articles
      1. This is natural language processing (NLP), and more specifically text classification, which can be tackled using recurrent neural networks (RNNs) and CNNs, but transformers work even better.
   4. Automatically flagging offensive comments on discussion forums
      1. text classification, using the same NLP tools
   5. Summarizing long documents automatically
      1. a branch of NLP called text summarization, again using the same tools.
   6. Creating a chatbot or a personal assistant
      1. involves many NLP components, including natural language understanding (NLU) and question-answering modules
   7. Forecasting your company’s revenue next year, based on many performances’ metrics
      1. a regression task
      2. may be tackled using any regression model, such as a linear regression or polynomial regression model (see Chapter 4), a regression support vector machine (see Chapter 5), a regression random forest (see Chapter 7), or an artificial neural network (see Chapter 10).
      3. If you want to take into account sequences of past performance metrics, you may want to use RNNs, CNNs, or transformers.
   8. Making your app react to voice commands
      1. speech recognition
      2. typically processed using RNNs, CNNs, or transformers
   9. Detecting credit card fraud
      1. anomaly detection, which can be tackled using isolation forests, Gaussian mixture models (see Chapter 9), or autoencoders
   10. Segmenting clients based on their purchases so that you can design a different marketing strategy for each segment
       1. clustering, which can be achieved using *k*-means, DBSCAN, and more
   11. Representing a complex, high-dimensional dataset in a clear and insightful diagram
       1. dimensionality reduction techniques
   12. Recommending a product that a client may be interested in, based on past purchases
       1. Artificial neural network
   13. Building an intelligent bot for a game
       1. reinforcement learning
8. **Types of Machine Learning Systems**

There are so many different types of machine learning systems that it is useful to classify them in broad categories, based on the following criteria:

* 1. How they are supervised during training (supervised, unsupervised, semi-supervised, self-supervised, and others)
  2. Whether or not they can learn incrementally on the fly (online versus batch learning)
  3. Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model, much like scientists do (instance-based versus model-based learning)
  4. These criteria are not exclusive; you can combine them in any way as per need.

1. **Training Supervision** 
   1. ML systems can be classified according to the amount and type of supervision they get during training.
   2. There are many categories, but we’ll discuss the main ones: supervised learning, unsupervised learning, self-supervised learning, semi-supervised learning, and reinforcement learning.
   3. **Supervised learning** 
      1. In supervised learning, the training set you feed to the algorithm includes the desired solutions, called labels.
      2. A typical supervised learning task is classification.
      3. Another typical task is to predict a target numeric value, such as the price of a car, given a set of features (mileage, age, brand, etc.). This sort of task is called regression.
      4. Note that 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 (e.g., 20% chance of being spam).
      5. Note - 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. Moreover, features are sometimes called predictors or attributes. These terms may refer to individual samples (e.g., “this car’s mileage feature is equal to 15,000”) or to all samples (e.g., “the mileage feature is strongly correlated with price”).
   4. **Unsupervised learning**

In unsupervised learning, as you might guess, the training data is unlabeled (Figure 1-7). The system tries to learn without a teacher.

* + 1. You may want to run a clustering algorithm to try to detect groups of similar visitors
    2. If you use a hierarchical clustering algorithm, it may also subdivide each group into smaller groups
    3. 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.
    4. A related task is **dimensionality** **reduction**, in which the goal is to simplify the data without losing too much information.
       1. The dimensionality reduction algorithm merge multiple features into one feature that represents the car’s wear and tear. This is called feature extraction.
    5. **TIP**: It is often a good idea to try to reduce the number of dimensions in your training data using a dimensionality reduction algorithm before you feed it to another machine learning algorithm (such as a supervised learning algorithm). It will run much faster, the data will take up less disk and memory space, and in some cases it may also perform better.
    6. Yet another important unsupervised task is anomaly detection—for example, detecting unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learning algorithm.
    7. A very similar task is novelty detection: it aims to detect new instances that look different from all instances in the training set. This requires having a very “clean” training set, devoid of any instance that you would like the algorithm to detect.
    8. Finally, another common unsupervised task is association rule learning, in which the goal is to dig into large amounts of data and discover interesting relations between attributes.
  1. **Semi-supervised learning**

Since 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.

* + 1. Most semi-supervised learning algorithms are combinations of unsupervised and supervised algorithms.
  1. **Self-supervised learning**
     1. 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.
     2. For example, if you have a large dataset of unlabeled images, you can randomly mask a small part of each image and then train a model to recover the original image (Figure 1-12). During training, the masked images are used as the inputs to the model, and the original images are used as the labels.
     3. But more often than not, a model trained using self-supervised learning is not the final goal. You’ll usually want to tweak and finetune the model for a slightly different task—one that you actually care about.
     4. NOTE - 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 (i.e., neural networks composed of many layers of neurons).
     5. Some people consider self-supervised learning to be a part of unsupervised learning, since it deals with fully unlabeled datasets. But self-supervised learning uses (generated) labels during training, so in that regard it’s closer to supervised learning. In short, it’s best to treat self-supervised learning as its own category.
  2. **Reinforcement learning**
     1. 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, as shown in Figure 1- 13).
     2. 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.
     3. this is called offline learning as well.

1. **Batch Versus Online Learning**

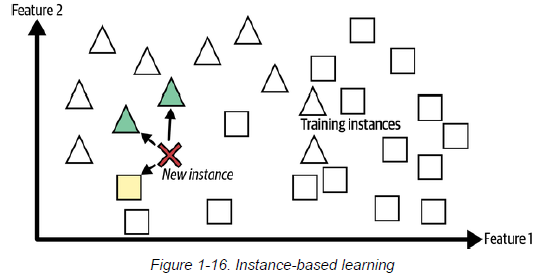
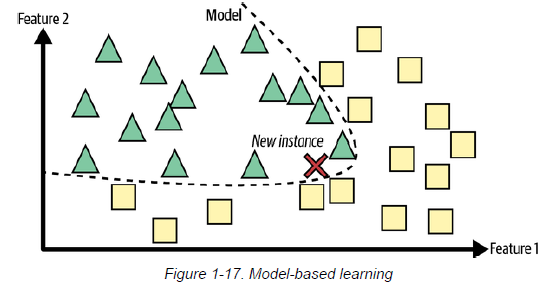
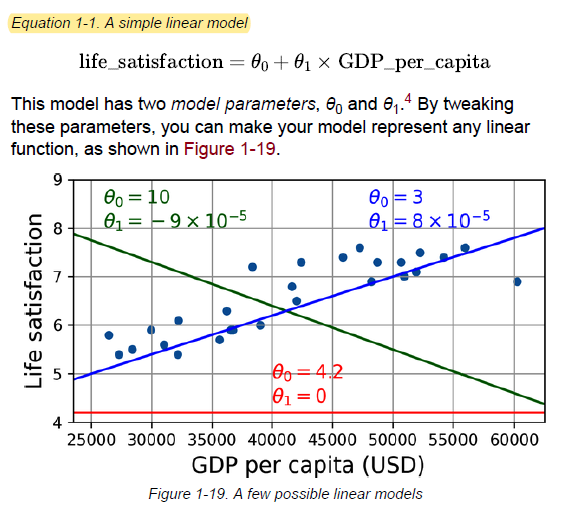
Another criterion used to classify machine learning systems is whether or not the system can learn incrementally from a stream of incoming data.

* 1. **Batch learning**
     1. 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.
     2. 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*.
     3. Unfortunately, a model’s performance tends to decay slowly over time, simply because the world continues to evolve while the model remains unchanged. This phenomenon is often called model rot or data drift.
     4. The solution is to regularly retrain the model on up-to-date data.
     5. How often you need to do that depends on the use case: if the model classifies pictures of cats and dogs, its performance will decay very slowly, but if the model deals with fast-evolving systems, for example making predictions on the financial market, then it is likely to decay quite fast.
     6. WARNING - Even a model trained to classify pictures of cats and dogs may need to be retrained regularly, not because cats and dogs will mutate overnight, but because cameras keep changing, along with image formats, sharpness, brightness, and size ratios.
     7. Simply update the data and train a new version of the system from scratch as often as needed.
     8. This solution is simple and often works fine, but training using the full set of data can take many hours, so you would typically train a new system only every 24 hours or even just weekly. If your system needs to adapt to rapidly changing data (e.g., to predict stock prices), then you need a more reactive solution.
     9. Also, training on the full set of data requires a lot of computing resources
     10. Finally, if your system needs to be able to learn autonomously and it has limited resources (e.g., a smartphone application or a rover on Mars), then carrying around large amounts of training data and taking up a lot of resources to train for hours every day is a showstopper.
  2. **Online learning**
     1. In 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.
     2. Process flow: 
     3. Online learning is useful for systems that need to adapt to change extremely rapidly (e.g., to detect new patterns in the stock market).
     4. Additionally, 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).
     5. One important parameter of online learning systems is how fast they should adapt to changing data: this is called the learning rate.
     6. WARNING - Out-of-core learning is usually done offline (i.e., not on the live system), so online learning can be a confusing name. Think of it as incremental learning.
     7. A big challenge with online learning is that if bad data is fed to the system, the system’s performance will decline, possibly quickly (depending on the data quality and learning rate).
     8. You may also want to monitor the input data and react to abnormal data; for example, using an anomaly detection algorithm.

1. **Instance-Based Versus Model-Based Learning**

One more way to categorize machine learning systems is by how they generalize:

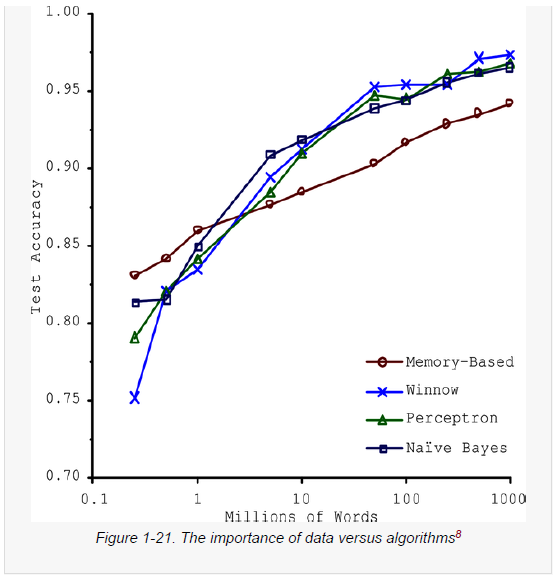
* + 1. Most machine learning tasks are about making predictions. This means that given a number of training examples, the system needs to be able to make good predictions for (generalize to) examples it has never seen before. Having a good performance measure on the training data is good, but insufficient; the true goal is to perform well on new instances.
    2. There are two main approaches to generalization: instance-based learning and model-based learning.

1. **Instance-based learning**
   * 1. Possibly the most trivial form of learning is simply to learn by heart.
     2. This is called 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 of them).
     3. Process: 
2. **Model-based learning and a typical machine learning workflow**
   * 1. 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. This is called model-based learning.
     2. Process: 
     3. you decide to model life satisfaction as a linear function of GDP per capita. This step is called model selection: you selected a linear model of life satisfaction with just one attribute, GDP per capita.
     4. 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. For linear regression problems, people typically use a cost function that measures the distance between the linear model’s predictions and the training examples; the objective is to minimize this distance.
     5. Process:
        1. 
     6. This is where the linear regression algorithm comes in: you feed it your training examples, and it finds the parameters that make the linear model fit best to your data. This is called training the model.
     7. WARNING –
        1. Confusingly, the word “model” can refer to a type of model (e.g., linear regression), to a fully specified model architecture (e.g., linear regression with one input and one output), or to the final trained model ready to be used for predictions (e.g., linear regression with one input and one output, using θ = 3.75 and θ = 6.78 × 10).
        2. Model selection consists in choosing the type of model and fully specifying its architecture. Training a model means running an algorithm to find the model parameters that will make it best fit the training data, and hopefully make good predictions on new data.
     8. In summary:
        1. You studied the data.
        2. You selected a model.
        3. You trained it on the training data (i.e., the learning algorithm searched for the model parameter values that minimize a cost function).
        4. Finally, you applied the model to make predictions on new cases (this is called inference), hoping that this model will generalize well.

This is what a typical machine learning project looks like.

1. **Main Challenges of Machine Learning**

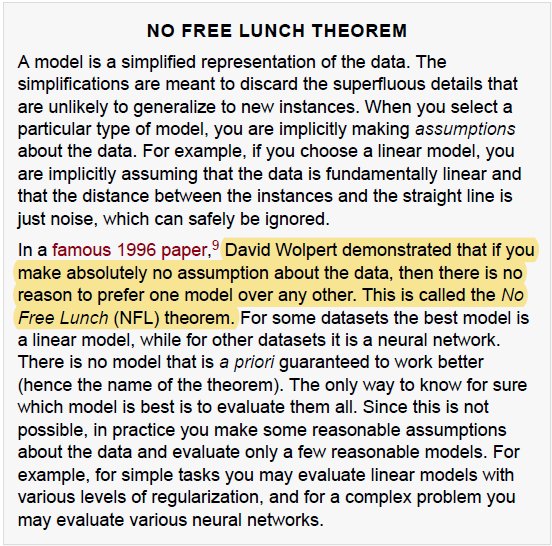
In short, since your main task is to select a model and train it on some data, the two things that can go wrong are “bad model” and “bad data”.

* 1. **Insufficient Quantity of Training Data**
     1. THE UNREASONABLE EFFECTIVENESS OF DATA
        1. 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,
        2. “these results suggest that we may want to reconsider the trade-off between spending time and money on algorithm development versus spending it on corpus development”
        3. It should be noted, however, that small and medium sized datasets are still very common, and it is not always easy or heap to get extra training data⁠—so don’t abandon algorithms just yet.
        4. Comparison: 
  2. Nonrepresentative Training Data
     1. 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.
     2. It is crucial to use a training set that is representative of the cases you want to generalize to. This is often harder than it sounds: 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.
  3. Poor Quality Data
     1. Obviously, 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.
     2. The following are a couple examples of when you’d want to clean up training data:
     3. If some instances are clearly outliers, it may help to simply discard them or try to fix the errors manually.
     4. If some instances are missing a few features (e.g., 5% of your customers did not specify their age), you must decide whether you want to ignore this attribute altogether, ignore these instances, fill in the missing values (e.g., with the median age), or train one model with the feature and one model without it.
  4. Irrelevant Features
     1. As the saying goes: garbage in, garbage out. 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:
        1. Feature selection (selecting the most useful features to train on among existing features)
        2. Feature extraction (combining existing features to produce a more useful one⁠—as we saw earlier, dimensionality reduction algorithms can help)
        3. Creating new features by gathering new data
  5. Overfitting the Training Data
     1. overfitting: it means that the model performs well on the training data, but it does not generalize well.
     2. WARNING - Overfitting happens when the model is too complex relative to the amount and noisiness of the training data. Here are possible solutions:
        1. 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, or by constraining the model.
        2. Gather more training data.
        3. Reduce the noise in the training data (e.g., fix data errors and remove outliers).
     3. Constraining a model to make it simpler and reduce the risk of overfitting is called regularization
     4. 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.
  6. Underfitting the Training Data
     1. underfitting is the opposite of overfitting: it occurs when your model is too simple to learn the underlying structure of the data.
     2. Here are the main options for fixing this problem:
        1. Select a more powerful model, with more parameters.
        2. Feed better features to the learning algorithm (feature engineering).
        3. Reduce the constraints on the model (for example by reducing the regularization hyperparameter).
  7. Stepping Back
     1. By now you know a lot about machine learning. However, we went through so many concepts that you may be feeling a little lost, so let’s step back and look at the big picture:
     2. Machine learning is about making machines get better at some task by learning from data, instead of having to explicitly code rules.
     3. There are many different types of ML systems: supervised or not, batch or online, instance-based or model-based.
     4. In an ML project you gather data in a training set, and you feed the training set to a learning algorithm.
     5. The system will not perform well if your training set is too small, or if the data is not representative, is noisy, or is polluted with irrelevant features (garbage in, garbage out).

1. Testing and Validating
   1. The only way to know how well a model will generalize to new cases is to actually try it out on new cases. One way to do that is to put your model in production and monitor how well it performs. This works well, but if your model is horribly bad, your users will complain —not the best idea.
      1. A better option is to split your data into two sets: the training set and the test set.
      2. 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.
      3. 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.
   2. **TIP** - It is common to use 80% of the data for training and hold out 20% for testing. However, this depends on the size of the dataset.
   3. Hyperparameter Tuning and Model Selection
      1. Evaluating a model is simple enough: just use a test set
      2. how can you decide between them? One option is to train both and compare how well they generalize using the test set.
         1. The question is, how do you choose the value of the regularization Hyperparameter? One option is to train 100 different models using 100 different values for this hyperparameter.
         2. 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.
         3. A common solution to this problem is called holdout validation (Figure 1-25): 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 the development set, or dev set).
         4. More specifically, you train multiple models with various hyperparameters on the reduced training set (i.e., the full training set minus the validation set), and you 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.
            1. 
         5. This solution usually works quite well. However, if the validation set is too small, then the model evaluations will be imprecise.
         6. 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. By averaging out all the evaluations of a model, you get a much more accurate measure of its performance.
   4. **Data Mismatch**
      1. In some cases, it’s easy to get a large amount of data for training, but this data probably won’t be perfectly representative of the data that will be used in production.
      2. In this case, 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 pictures - 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).
      3. One solution is to hold out some of the training pictures (from the web) in yet another set that Andrew Ng dubbed the train-dev set (Figure 1-26). After the model is trained (on the training set, not on the train-dev set), you can evaluate it on the train-dev set. If the model performs poorly, then it must have overfit the training set, so you should try to simplify or regularize the model, get more training data, and clean up the training data. But if it performs well on the train-dev set, then you can evaluate the model on the dev set.
      4. Once you have a model that performs well on both the train-dev set and the dev set, you can evaluate it one last time on the test set to know how well it is likely to perform in production.
      5. When real data is scarce (right), you may use similar abundant data (left) for training and hold out some of it in a train-dev set to evaluate overfitting; the real data is then used to evaluate data mismatch (dev set) and to evaluate the final model’s performance (test set)



* 1. No Free Lunch Theorem:



## Chap01 Notes:

1. import sys
   1. assert sys.version\_info >= (3, 7)

# The statement assert sys.version\_info >= (3, 7) ensures the Python version running the script is **3.7 or higher**.

* **sys.version\_info**: A tuple containing the Python version as (major, minor, micro, ...).
* **assert**: Raises an AssertionError if the condition is False.

1. from packaging import version
   1. import sklearn
   2. assert version.parse(sklearn.\_\_version\_\_) >= version.parse("1.0.1")
   * ## This statement ensures that the installed version of the scikit-learn library (sklearn) is **at least version 1.0.1**. Here's how it works:
   * **version.parse**: Parses version strings (e.g., "1.0.1") into comparable objects.
   * **sklearn.\_\_version\_\_**: Fetches the installed version of scikit-learn.
   * **Assertion**: Checks if the installed version is greater than or equal to "1.0.1". If not, it raises an AssertionError.
2. import matplotlib.pyplot as plt
3. import numpy as np

np.random.seed(42) - np.random.seed(42) sets the seed for NumPy's random number generator, ensuring that the sequence of random numbers produced by functions like np.random.rand() or np.random.randint() is reproducible. By using the same seed value (e.g., 42), the random numbers generated will always be identical across runs, which is helpful for debugging or ensuring consistent results in experiments.

1. from sklearn.linear\_model import LinearRegression
2. datapath = Path() / "datasets" / "lifesat"

datapath.mkdir(parents=True, exist\_ok=True)

1. url = data\_root + "lifesat/" + filename

urllib.request.urlretrieve(url, datapath / filename)

1. oecd\_bli = oecd\_bli.pivot(index="Country", columns="Indicator", values="Value")

## This line reshapes the DataFrame oecd\_bli using the **pivot** function. Here's what it does:

* **index="Country"**: Sets the rows to be indexed by the "Country" column.
* **columns="Indicator"**: Sets the column headers to be the unique values of the "Indicator" column.
* **values="Value"**: Fills the table cells with the data from the "Value" column.

It transforms the DataFrame from long format (multiple rows per country and indicator) to wide format (one row per country, with indicators as separate columns).

1. full\_country\_stats = pd.merge(left=oecd\_bli, right=gdp\_per\_capita,left\_index=True, right\_index=True)
2. loc - Access data by **label** (row/column names).
   1. Iloc - Access data by **index position** (integer-based).
3. country\_stats.plot(kind='scatter', figsize=(5, 3), grid=True, x=gdppc\_col, y=lifesat\_col)
4. t0, t1 = lin1.intercept\_[0], lin1.coef\_[0][0]
5. plt.text(max\_gdp - 20\_000, min\_life\_sat + 1.9,

fr"$\theta\_0 = {t0:.2f}$", color="b")

1. plt.annotate(country, xy=(pos\_data\_x, pos\_data\_y),

xytext=pos\_text,fontsize=12,

arrowprops=dict(facecolor='black', width=0.5,

      shrink=0.08, headwidth=5))

## This code uses matplotlib.pyplot.annotate to label a point on a plot with an annotation and an arrow. Here's a concise breakdown:

* **country**: The text to display as the annotation.
* **xy=(pos\_data\_x, pos\_data\_y)**: The coordinates of the point to annotate.
* **xytext=pos\_text**: The position of the annotation text relative to the point.
* **fontsize=12**: Font size of the annotation text.
* **arrowprops=dict(...)**: Defines the appearance of the arrow:
  + facecolor='black': Arrow color.
  + width=0.5: Thickness of the arrow shaft.
  + shrink=0.08: Shrinks the arrow length slightly.
  + headwidth=5: Width of the arrowhead.

This visually connects the text label to a specific point on the plot.

1. from sklearn import preprocessing

## from sklearn import preprocessing imports the preprocessing module from scikit-learn, a library used for preparing and transforming data in machine learning workflows. The preprocessing module provides tools like scaling, normalization, encoding categorical variables, and more to standardize and optimize datasets for model training.

1. from sklearn import pipeline

## In sklearn, the pipeline module provides tools to chain multiple data transformation and modeling steps into a single object. This simplifies workflows, ensuring that preprocessing steps (e.g., scaling, encoding) and models are applied consistently.

**Key Features:**

1. **Consistency**: Automates sequential steps (e.g., data preprocessing + model training).
2. **Code Simplicity**: Reduces redundancy by combining transformations and predictions.
3. **Cross-Validation**: Ensures transformations are applied only to training data during model evaluation.
4. poly = preprocessing.PolynomialFeatures(degree=10, include\_bias=False)

scaler = preprocessing.StandardScaler()

##   
1.) **PolynomialFeatures(degree=10, include\_bias=False)**:

* Expands input features into polynomial combinations up to the 10th degree.
* include\_bias=False excludes the constant (bias) term from the output.

2) **StandardScaler()**:

* Standardizes features by removing the mean and scaling to unit variance.
* Ensures all features have similar scales, improving model performance.

1. X

# Chap02 –End-to-End Machine Learning Project

1. From ZLIB import CRC32
   1. The crc32 function from Python's zlib module computes a **cyclic redundancy check (CRC)** checksum for data. It is commonly used to verify data integrity.
   2. The function returns a 32-bit integer representing the CRC checksum.
   3. Kwargs - keyword arguments