2025-02-17 02:23

tags :

- Machine Learning
- Hands on ML - book

Chapter 1 - Hands on

intro

machine learning is great for:

- Problems for which existing solutions require a lot of fine-tuning or long lists of rules (a machine learning model can often simplify code and perform better than the traditional approach)
- Complex problems for which using a traditional approach yields no good solution (the best machine learning techniques can perhaps find a solution)
- Fluctuating environments (a machine learning system can easily be retrained on new data, always keeping it up to date)
- Getting insights about complex problems and large amounts of data

Types of Machine Learning Systems:

- How they are supervised during training (supervised, unsupervised, semi-supervised, self-supervised, and others)
- Whether or not they can learn incrementally on the fly (online versus batch learning)
- 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)

1. Training Supervision

- ML systems can be classified according to the amount and type of supervision they get during training. There are many categories, but we'll discuss the main ones: supervised learning, unsupervised learning, self supervised learning, semi-supervised learning, and reinforcement learning.

Supervised learning

- In supervised learning, the training set you feed to the algorithm includes the desired solutions, called labels
- Note that some regression models can be used for classification as well, and vice versa

Unsupervised learning

- In unsupervised learning, as you might guess, the training data is unlabeled. The system tries to learn without a teacher.
- A related task is dimensionality reduction, in which the goal is to simplify the data without losing too much information. One way to do this is to merge several correlated features into one.
- 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.

Semi-supervised learning

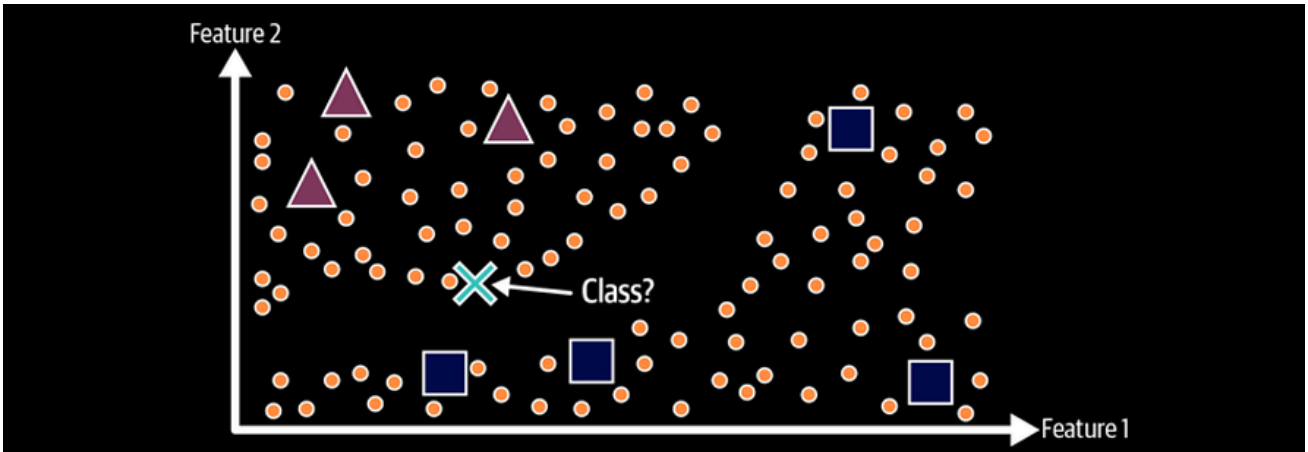
- labeling data is time-consuming and costly, you will often have plenty of unlabeled instances, and few labeled instances. Some algorithms can deal with data that's partially labeled This is called semi supervised learning
- 

Figure 1-11. Semi-supervised learning with two classes (triangles and squares): the unlabeled examples (circles) help classify a new instance (the cross) into the triangle class rather than the square class, even though it is closer to the labeled squares
- Most semi-supervised learning algorithms are combinations of unsupervised and supervised algorithms

Self-supervised learning

- actually generating a fully labeled dataset from a fully unlabeled one

- Self-supervised learning is a type of machine learning that helps a model learn from unlabeled data by creating its own labels. It is a special case of supervised learning where the model generates labels automatically instead of relying on human-labeled data.
- **Simple Example to Understand the Concept** [1]

Reinforcement learning

- 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

2. 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.

Batch learning (offline learning)

- the system is incapable of learning incrementally: it must be trained using all the available data, which take alot of time and resources
- after the model is trained it launched into production and runs without learning anymore
- a model's performance tends to decay slowly over time because the world continues to evolve while the model unchanged This is called `model rot` or `data drift`
- batch learning system can adapt to change. Simply update the data and train a new version of the system from scratch as often as needed.

Online learning (incrementally)

- 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
- Online learning is useful for systems that need to adapt to change rapidly, It is also a good option if you have limited computing resources; for example, if the model is trained on a mobile device
- 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 is how fast they adapt to changing data: this is called the `learning rate`
  - If the learning rate high, the system will rapidly adapt to new data but it will also tend to quickly forget the old data
  - If the learning rate low, the system will have more inertia; that is, 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)
- 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**.
- 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)
- online learning can be done offline `or` offline learning can be done online , the two types don't depend on the internetconnection

3. Instance-Based Versus Model-Based Learning

- Having a good performance measure on the training data is good, but insufficient; the true goal is to perform well on new instances.

Instance-based learning

- involves memorizing training examples and generalizing to new cases using similarity measures.
- **Key mechanism**: Compare new instances to stored examples (or a subset) using a similarity metric (e.g., counting shared words in emails).
- **Example**: A spam filter flags emails similar to known spam (not just identical ones). like in KNN
- **Core principle**: Classification/decision depends on the majority class or pattern of the most similar stored instances.

Model-based learning and a typical machine learning workflow

- build a model of these examples and then use that model to make predictions.
- like linear or logistic regression
- steps
  - 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 of Machine Learning

- the two things that can go wrong are “bad model” and “bad data”. Let’s start with examples of bad data.

Bad Data :-

1. Insufficient Quantity of Training Data

- the more data the more accuracy

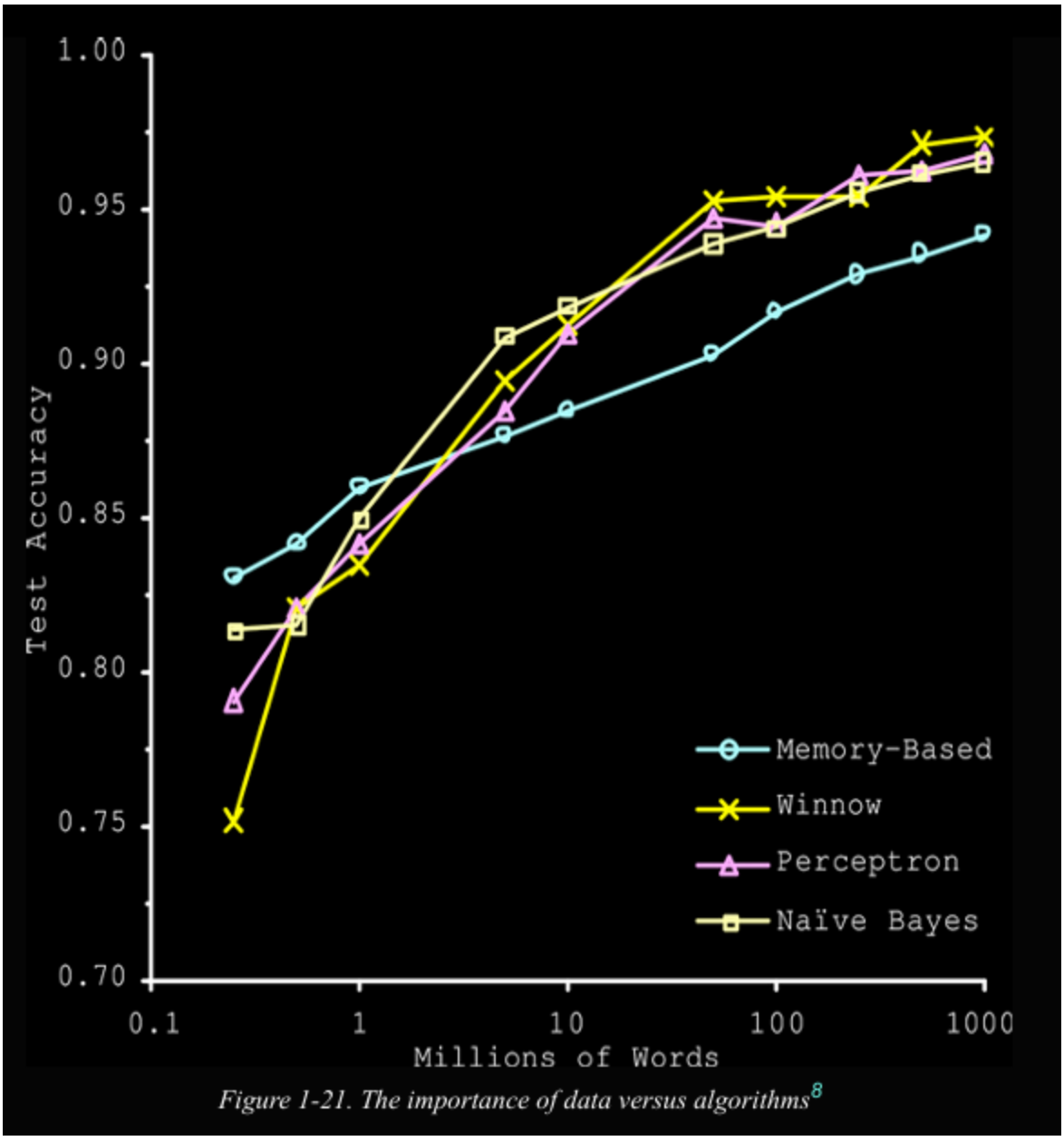


Figure 1-21. The importance of data versus algorithms[8]

2. 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.
- a GDP per capita lower than $23,500 or higher than $62,500. Figure 1-22 shows what the data looks like when you add such countries.

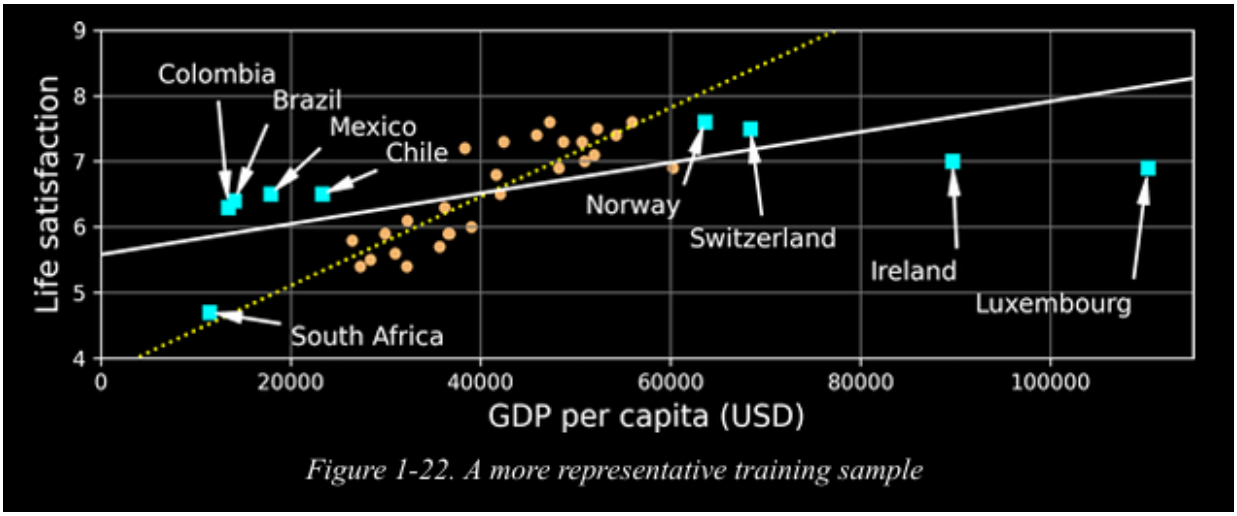


Figure 1-22. A more representative training sample

  - If you train a linear model on this data, you get the solid line, while the old model is represented by the dotted line.
  - It seems that very rich countries are not happier than moderately rich countries
- By using a nonrepresentative training set, you trained a model that is unlikely to make accurate predictions, especially for very poor and very rich countries.
- Using a representative training dataset is essential for effective generalization but challenging to achieve. Small samples risk **sampling noise** (nonrepresentative data due to chance), while large samples may still suffer from **sampling bias** if data collection methods are flawed. Both issues undermine model performance, emphasizing the need for thoughtful sampling design.

3. Poor-Quality Data

- Obviously, if your training data is full of errors, outliers, and noise , it will make it harder for the system to detect the underlying patterns
- The following are a couple examples of when you'd want to clean up training data:
  - If some instances are clearly outliers, it may help to simply discard them or try to fix the errors manually.
  - 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

- The principle `garbage in, garbage out` underscores that a machine learning system's performance hinges on **high-quality training data** with sufficient relevant features and minimal irrelevant ones. **Feature engineering**
  - **Feature selection**: Identifying the most useful existing features
  - **Feature extraction**: Combining or transforming features (e.g., via dimensionality reduction) to enhance their utility.
  - **Creating new features**: Collecting additional data to improve model learning.

Bad Model :-

1. Overfitting the Training Data

- it 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. these patterns will not generalize to new instances
- how to fix :-
  - Gather more training data.
  - 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.
  - Reduce the noise in the training data (e.g., fix data errors and remove outliers).
- a way to Simplify the model to fix the overfitting is `regularization`
  - 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)
    - it is not affected by the learning algorithm itself; it must be set prior to training and remains constant during training

2. Underfitting the Training Data :-

- occurs when a model is too simplistic to capture the underlying patterns in the data, leading to poor performance even on training examples (e.g., a linear model applied to complex, nonlinear phenomena like life satisfaction).
- **Solutions to address underfitting include:**
  1. **Using a more powerful model** (e.g., increasing parameters or complexity).
  2. **Improving feature engineering** (enhancing input features to better represent the data).
  3. **Reducing model constraints** (e.g., lowering regularization to allow the model greater flexibility).

- Fixing underfitting ensures the model can learn meaningful relationships in the data.

Testing and Validating

- To evaluate a model's generalization ability, test it on new cases.
- Deploying the model in production can help, but poor performance may lead to user complaints.
- A better approach is to split the data into a **training set** (for training) and a **test set** (for evaluation).
- The **generalization error** (out-of-sample error) measures performance on unseen data.
- Testing on the test set provides an estimate of the generalization error.
- A low **training error** but high **generalization error** indicates **overfitting**.
- `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: if it contains 10 million instances, then holding out 1% means your test set will contain 100,000 instances, probably more than enough to get a good estimate of the generalization error.`

Hyperparameter Tuning and Model Selection

- **Model selection** involves comparing different models (e.g., linear vs. polynomial) based on their generalization performance.
- **Hyperparameter tuning** requires testing multiple values, but overfitting to the test set can lead to poor real-world performance.
- **Holdout validation** mitigates this by splitting the training set into a smaller training set and a **validation set** for model selection.
- After selecting the best model using the validation set, it is retrained on the full training set and evaluated on the test set.
- **Validation set size trade-off**:
  - Too small → Model selection may be inaccurate.
  - Too large → Models are trained on much less data, leading to misleading comparisons.
- **Cross-validation** (e.g., k-fold CV) improves model selection by averaging performance across multiple validation sets, though it increases training time.

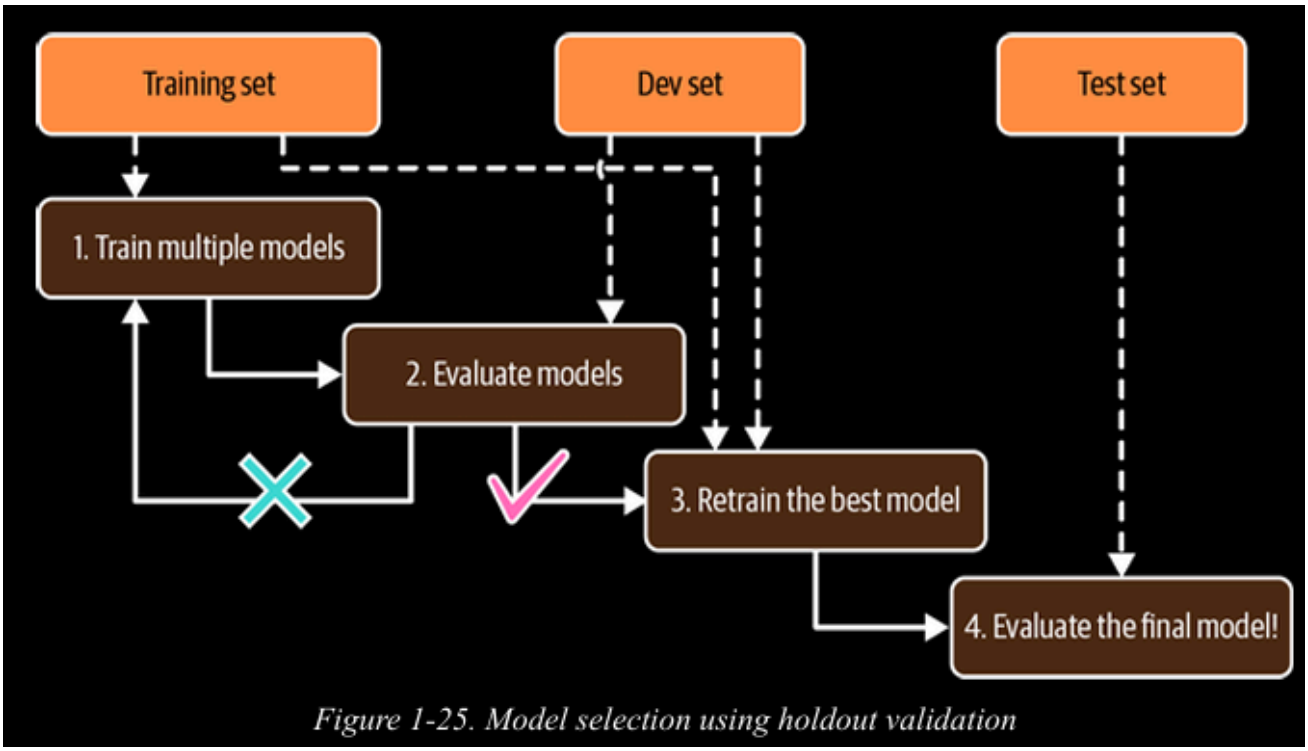
- 

Figure 1-25. Model selection using holdout validation

Data Mismatch

- **Training data may not always be representative** of real-world production data.
- Example: A flower recognition app trained on web images may not perform well on mobile-taken photos.
- **Validation and test sets** must be representative of production data to ensure accurate evaluation.
- **Data mismatch issue:** Poor validation performance could be due to either **overfitting** or **mismatch** between training and production data.
- **Solution:** Introduce a **train-dev set** (from training data but not used for training) to diagnose the issue:
  - Poor performance on train-dev → Overfitting (need regularization, more data, or data cleaning).
  - Good performance on train-dev but poor on validation → Data mismatch (adjust preprocessing to make training data more like real-world data).
- Once the model performs well on both **train-dev** and **validation sets**, test it on the **test set** for a final performance estimate.

NO FREE LUNCH THEOREM

- **Model Assumptions:** A model simplifies data by discarding unnecessary details, assuming certain patterns.

- **Example:** A linear model assumes data follows a linear relationship, treating deviations as noise.
- **No Free Lunch (NFL) Theorem :**
  - Without assumptions about data, no model is inherently better than another.
  - The best model varies by dataset (e.g., linear models work best for some, neural networks for others).
- **Implication:** Since testing all models is impractical, we make reasonable assumptions and evaluate a few suitable models.
- **Application:** Simple problems may use linear models; complex ones may require neural networks.

Resources :

-

Related notes :

-

References :

- ## Internal :
  -
- ## External :
  - the notebook of the chapter
  - hegab videos
  - the book
  -

---

1. Imagine you want to teach a child to recognize cats and dogs, but instead of telling them directly, you give them a puzzle to solve!

   ### Step-by-Step Explanation:

1. **Hide Part of the Data and Force the Model to Guess It**
   - Suppose you have an image of a cat. You randomly mask (hide) a small part of the image.
   - The model's task is to predict the missing part based on the rest of the image.
   - If the model correctly reconstructs the missing part, it means it has learned the important patterns of what a cat looks like.
2. **Learning Through Repetition**
   - The model repeats this process with thousands of images, getting better over time at predicting missing parts.
   - Eventually, it understands what features distinguish cats from dogs **without ever being explicitly told**.
3. **Using the Learned Knowledge for Other Tasks**
   - Once the model has learned to understand images, we can **fine-tune** it for other tasks.
   - For example, we can now train it to classify pet species by using a small labeled dataset.
   - The model already knows what cats and dogs look like, so it only needs a little extra training to map these patterns to labels like "cat" or "dog."

↩