Notes on *Hands-on Machine Learning*

Jonathan Chen

September 15, 2025

Contents

1 Machine Learning Landscape

- Public perception: robots (butlers, Terminators), but ML is already real
- Has existed for decades in specialized tasks (e.g., OCR)
- First mainstream application: spam filters in 1990s
  - Learned to block spam with high accuracy
  - Impact: improved email usability for hundreds of millions
- Now powers many everyday features (recommendations, voice search, etc.)
- Learning $\neq$ storing information (downloading Wikipedia does not count)
- ML = systems improve performance on tasks through experience
- Main categories: supervised vs. unsupervised, online vs. batch, instance-based vs. model-based
- Chapter also covers ML workflow, challenges, evaluation, fine-tuning
- High-level overview, minimal code

1.1 What is Machine Learning

- ML = science and art of programming computers to learn from data
- Arthur Samuel (1959): field of study giving computers the ability to learn without explicit programming
- Tom Mitchell (1997): program learns from experience E w.r.t. task T and performance measure P if performance on T (measured by P) improves with E
- Example: spam filter
  - T: flag spam in new emails
  - E: training data (spam + ham emails)
  - P: accuracy (ratio of correctly classified emails)
- Terminology
  - Training set = collection of examples
  - Training instance/sample = individual example
- Downloading Wikipedia = more data, but no learning (not better at any task)

1.2 Why Use Machine Learning

- Traditional spam filter
  - Manually define rules (keywords, sender patterns, etc.)
  - Program becomes long, complex, hard to maintain
  - Needs constant updating when spammers change tactics (e.g., "4U" → "For U")
- ML-based spam filter
  - Learns predictive words/phrases automatically from data
  - Shorter, easier to maintain, more accurate
  - Adapts automatically as new spam patterns appear
- Other use cases
  - Problems too complex for traditional programming (e.g., speech recognition)
  - No known algorithmic solution
- ML can also help humans learn
  - Trained models can be inspected to reveal predictive features
  - Reveals new trends or correlations (data mining)

1.3 Examples of Applications

- Image classification → CNNs
- Semantic segmentation (tumors) → CNNs
- Text classification (news, moderation) → RNNs/CNNs/Transformers
- Text summarization → Transformers
- Chatbots and assistants → NLU, QA, Transformers
- Forecasting and regression → Linear/Polynomial Regression, SVM, Random Forest, neural networks; sequences → RNNs/CNNs/Transformers
- Speech recognition → RNNs/CNNs/Transformers
- Anomaly detection (fraud) → anomaly detection models
- Clustering (customer segments) → k-means, GMM, hierarchical
- Dimensionality reduction and visualization → PCA, t-SNE, UMAP
- Recommender systems → matrix factorization, sequence models, deep nets
- Reinforcement learning (game bots) → RL agents

1.4 Types of Machine Learning

- Supervision: supervised, unsupervised, semisupervised, reinforcement learning
- Learning mode: batch vs. online (incremental)
- Approach: instance-based vs. model-based
- Criteria can be combined (e.g., spam filter = online, model-based, supervised)

1.4.1 Supervised Learning

- Training data includes input + desired output (labels)
- Tasks
  - Classification (e.g., spam filter: spam vs ham)
  - Regression (predict numeric value, e.g., car price from features)
- Terminology
  - Attribute = data type (e.g., mileage)
  - Feature = attribute + value (e.g., mileage = 15,000)
- Algorithms
  - k-Nearest Neighbors
  - Linear Regression
  - Logistic Regression
  - Support Vector Machines
  - Decision Trees, Random Forests
  - Neural networks
- Note: some regression algorithms also used for classification (e.g., Logistic Regression)

1.4.2 Unsupervised Learning

- Training data is unlabeled; system learns without teacher
- Algorithms
  - Clustering: K-Means, DBSCAN, Hierarchical Cluster Analysis
  - Anomaly/novelty detection: One-class SVM, Isolation Forest
  - Visualization/dimensionality reduction: PCA, Kernel PCA, LLE, t-SNE
  - Association rule learning: Apriori, Eclat
- Examples

  - Clustering visitors to a blog → groups with similar behavior
  - Visualization: 2D/3D representation of complex data, highlight clusters
  - Dimensionality reduction: merge correlated features (e.g., mileage + age → wear/tear)
  - Anomaly detection: fraud, defects, outlier removal
  - Novelty detection: detect unseen instances distinct from training data
  - Association rules: market basket analysis (e.g., chips + sauce → steak)

1.4.3 Semisupervised Learning

- Used when most data is unlabeled and only a few labeled instances available
- Example: Google Photos
  - Clusters faces (unsupervised)
  - User provides one label per person → names applied to all photos
- Algorithms often combine supervised + unsupervised methods
- Example: Deep Belief Networks (DBNs)
  - Built from Restricted Boltzmann Machines (RBMs) trained unsupervised
  - Then fine-tuned with supervised learning

1.4.4 Reinforcement Learning

- Agent interacts with environment
  - Observes state
  - Selects and performs actions
  - Receives rewards or penalties
- Goal: learn optimal policy (strategy for choosing actions to maximize long-term reward)
- Examples
  - Robots learning to walk
  - DeepMind AlphaGo: learned policy by analyzing games + self-play; applied policy to beat world champion

1.5 Batch or Online Learning

1.5.1 Batch Learning

- Learns from all available data at once $\rightarrow$ no incremental updates
- Training is offline; system applies learned model in production without further learning
- Updating requires retraining from scratch on full dataset and redeploying
- Pros: simple, automatable retraining pipeline
- Cons: slow (hours+), resource-intensive, expensive, not suitable for rapidly changing data or limited-resource settings

1.5.2 Online Learning

- Trains incrementally on sequential data (single instances or mini-batches)
- Suitable for continuous data streams, rapid adaptation, limited resources
- Out-of-core learning: train on huge datasets by loading chunks sequentially
- Learning rate controls adaptation speed
  - High rate: fast adaptation, but quickly forgets old data
  - Low rate: slow adaptation, more stable, less sensitive to noise
- Risks: bad data can degrade performance over time
- Requires monitoring and ability to stop/revert if performance drops

1.6 Instance-Based vs. Model-Based Learning

- Goal of ML: make predictions that generalize well to unseen examples
- Good training performance alone is insufficient; true goal is performance on new data
- Two approaches to generalization:
  - Instance-based learning
  - Model-based learning

1.6.1 Instance-Based Learning

- Trivial form of learning = memorize examples
- Spam filter example
  - Flags emails identical to known spam
  - Can be extended to flag emails similar to known spam
  - Requires similarity measure (e.g., count common words)
- Definition: learns examples by heart, generalizes by comparing new cases with learned examples (or subset) using similarity
- Example: new instance classified as triangle if most similar stored instances are triangles

1.6.2 Model-Based Learning

- Builds a model of examples, then uses it for predictions
- Example: GDP per capita vs. life satisfaction
  - Data shows linear trend
  - Model selection: linear model with parameters θ_0, θ_1
  - Training = algorithm finds parameters minimizing cost function (distance between predictions and data)
  - Result: $\theta_0 = 4.85$, $\theta_1 = 4.91 \times 10^{-5}$
- Model can then be used for prediction (e.g., Cyprus GDP $\Rightarrow$ life satisfaction ≈ 5.96)
- Code example: Scikit-Learn Linear Regression (load data, train, predict)
- Instance-based comparison: k-Nearest Neighbors regression (e.g., $k = 3$ with Slovenia, Portugal, Spain $\Rightarrow$ prediction = 5.77)
- Improvements: add more attributes, more/better training data, or more powerful model (e.g., Polynomial Regression)
- Typical ML project steps
  - Study data
  - Select model
  - Train model (learn parameters minimizing cost function)
  - Apply model to make predictions (inference)

1.7 Challenges of Machine Learning

1.7.1 Insufficient Quantity of Training Data

- Toddlers can learn concepts (e.g., apple) from a few examples
- ML algorithms usually require large amounts of data
  - Thousands of examples for simple problems
  - Millions of examples for complex problems (e.g., image, speech recognition) unless reusing parts of existing models
- Unreasonable Effectiveness of Data
  - Banko & Brill (2001): very different algorithms perform similarly well given enough data (natural language disambiguation)
  - Suggests investing in corpus development may be more valuable than algorithm tweaks
  - Norvig et al. (2009): data matters more than algorithms for complex problems
  - Caveat: small- and medium-sized datasets are still common, and extra data is not always easy or cheap

1.7.2 Nonrepresentative Training Data

- To generalize well, training data must be representative of future cases
- Example: GDP vs. life satisfaction
  - Missing countries → model fit changes significantly
  - Reveals that a simple linear model is not sufficient
- Small samples → sampling noise
- Large samples with flawed method → sampling bias
- Famous case: US 1936 election poll by Literary Digest
  - Used biased lists (phones, magazines, clubs) → overrepresented wealthy Republicans
  - Only 25% responded → nonresponse bias
  - Predicted Landon win (57%) vs. Roosevelt actual win (62%)
- Other example: YouTube "funk music" search
  - Biased toward popular artists
  - Regional bias (e.g., "funk carioca" in Brazil)

1.7.3 Poor-Quality Data

- Errors, outliers, and noise in training data make pattern detection harder and reduce performance
- Data cleaning is crucial and time-consuming
- Examples
  - Outliers → discard or fix manually
  - Missing features → options:
    - Ignore attribute
    - Ignore instances
    - Fill missing values (e.g., median)
    - Train separate models (with/without feature)

1.7.4 Irrelevant Features

- Garbage in, garbage out: system needs enough relevant features and not too many irrelevant ones
- Success of ML project depends on good feature set
- Feature engineering steps
  - Feature selection: choose most useful features
  - Feature extraction: combine existing features (e.g., dimensionality reduction)
  - Create new features: gather new data

1.7.5 Overfitting the Training Data

- Overfitting = model performs well on training data but fails to generalize
- Example: high-degree polynomial fits training data better than linear model but unreliable
- Complex models (e.g., deep neural nets) may detect patterns in noise if training set is noisy or too small
- Example: life satisfaction model detecting false "w-satisfaction" rule from country names
- Overfitting occurs when model is too complex relative to data
- Possible solutions
  - Simplify model (fewer parameters, fewer attributes, constraints)
  - Gather more training data

  - Reduce noise in training data (fix errors, remove outliers)
- Regularization = constrain model to make it simpler and reduce overfitting risk
- Regularization strength controlled by hyperparameter (set before training, constant during training)
- Example: regularization reduces slope in linear model → worse fit to training data but better generalization

1.7.6 Underfitting the Training Data

- Underfitting = model too simple to capture underlying structure
- Example: linear model of life satisfaction underfits (reality is more complex)
- Fixes
  - Use a more powerful model (more parameters)
  - Provide better features (feature engineering)
  - Reduce constraints (e.g., lower regularization hyperparameter)

1.8 Testing and Validating

- Only way to know if model generalizes = try it on new cases
- Production testing: possible but risky if model is very bad
- Better: split data into training set and test set
- Train on training set, test on test set
- Generalization error (out-of-sample error) = error rate on new cases
- If training error low but generalization error high → overfitting

1.8.1 Hyperparameter Tuning and Model Selection

- Comparing models (e.g., linear vs. polynomial): train both, compare generalization error on test set
- Problem: repeated evaluation on test set adapts model to test set → poor performance on new data
- Solution: holdout validation
  - Split training set into reduced training set + validation set
  - Train candidate models on reduced training set
  - Select model that performs best on validation set

  - Retrain best model on full training set (including validation set)
  - Final evaluation done once on test set

- Trade-offs
  - Validation set too small → imprecise evaluations
  - Validation set too large → smaller training set, suboptimal comparison

- Alternative: repeated cross-validation
  - Many small validation sets
  - Evaluate each model once per validation set, average results
  - More accurate but multiplies training time

1.8.2 Data Mismatch

- Large training data may not represent production data
- Example: flower app
  - Millions of web flower pictures available
  - Only 10,000 representative pictures (taken with app)
- Validation and test sets must be representative of production data
  - Use only representative pictures
  - Split into validation/test without duplicates
- Problem: poor validation performance → unclear if due to overfitting or data mismatch
- Solution: add a train-dev set (subset of training data, e.g., web pictures)
  - Good performance on train-dev but poor on validation → data mismatch
  - Poor performance on train-dev → overfitting
- Fix mismatch: preprocess training data to resemble production data, retrain

1.8.3 No Free Lunch Theorem

- Models simplify observations by discarding details unlikely to generalize
- Assumptions are required to decide what to keep or discard
- Example: linear model assumes data is fundamentally linear, deviations = noise
- Wolpert (1996): without assumptions, no reason to prefer one model over another
- NFL theorem: no model is guaranteed to work better on all datasets

- Best model depends on dataset; only way to know is to evaluate
- In practice: make reasonable assumptions and evaluate a few models (e.g., linear models with regularization for simple tasks, neural networks for complex tasks)

2 End-to-End Machine Learning Project

- Example project: role of a newly hired data scientist at a real estate company
- Main steps
  1. Look at the big picture
  2. Get the data
  3. Discover and visualize the data to gain insights
  4. Prepare the data for Machine Learning algorithms
  5. Select a model and train it
  6. Fine-tune your model
  7. Present your solution
  8. Launch, monitor, and maintain your system

2.1 Working with Real Data

- Best to experiment with real-world data rather than artificial datasets
- Sources of open datasets
  - Popular repositories: UC Irvine ML Repository, Kaggle, AWS datasets
  - Meta portals: Data Portals, OpenDataMonitor, Quandl
  - Other lists: Wikipedia ML datasets, Quora, datasets subreddit
- Dataset used in this chapter: California Housing Prices (StatLib, based on 1990 census)
  - Modified for teaching (categorical attribute added, some features removed)

2.2 Look at the Big Picture

- Task: build a model of California housing prices using census data
- Data includes metrics such as population, median income, and median housing price
- Unit of analysis: block groups (called "districts"), population 600–3,000
- Goal: predict median housing price in any district from other metrics
- First step: consult and adapt the Machine Learning project checklist (Appendix B)
- Chapter will cover many checklist items, some will be skipped or discussed later

2.2.1 Frame the Problem

- First question: clarify business objective
  - Building a model is not the end goal
  - Output will feed into downstream ML system for real estate investment decisions
  - Critical for revenue
- Pipelines
  - Sequence of data processing components
  - Components run asynchronously, connected by data stores
  - Advantages: modular, robust, easy to visualize
  - Risk: broken component may go unnoticed without monitoring
- Current solution: manual expert estimates
  - Costly, time-consuming, often off by more than 20%
  - Model trained on census data could improve predictions
- Problem framing
  - Supervised learning task (labeled data: district housing prices)
  - Regression problem (predict value)
  - Multiple regression (multiple features)
  - Univariate regression (predict single value per instance)
  - Batch learning suitable (data small, no rapid changes)
- If data were huge $\rightarrow$ use MapReduce or online learning

2.2.2 Select a Performance Measure

- Typical regression measure: Root Mean Square Error (RMSE)
  - Higher weight for large errors
  - Formula:

$$RMSE(X, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2}$$

- Notation
  - m = number of instances
  - $x^{(i)}$ = feature vector of ith instance, $y^{(i)}$ = label
  - X = matrix of all features

- h = prediction function (hypothesis), $\hat{y}^{(i)} = h(x^{(i)})$
- RMSE(X, h) = cost function
- Conventions: scalars italic, vectors bold lowercase, matrices bold uppercase

- Mean Absolute Error (MAE)
  - Formula:

$$MAE(X, h) = \frac{1}{m} \sum_{i=1}^{m} |h(x^{(i)}) - y^{(i)}|$$

- Norms
  - RMSE = ℓ_2 norm (Euclidean)
  - MAE = ℓ_1 norm (Manhattan)
  - General ℓ_k norm: $\|v\|_k = (|v_0|^k + |v_1|^k + \cdots + |v_n|^k)^{1/k}$
  - ℓ_0: number of nonzeros, ℓ_∞: max absolute value
- RMSE more sensitive to outliers; preferred when outliers are rare

2.2.3 Check the Assumptions

- Good practice: list and verify assumptions early to avoid serious issues
- Example
  - Assumption: downstream system uses predicted prices directly
  - If downstream system converted prices into categories (cheap, medium, expensive), the task would be classification, not regression
- After confirming with downstream team: actual prices are needed → regression framing is correct

2.3 Get the Data

2.3.1 Create the Workspace

- Install Python (https://www.python.org) if not already installed
- Create workspace directory

```
$ export ML_PATH="$HOME/ml"
$ mkdir -p $ML_PATH
```

- Required modules: Jupyter, NumPy, pandas, Matplotlib, SciPy, Scikit-Learn
- Install/upgrade pip

```
$ python3 -m pip --version
$ python3 -m pip install --user -U pip
```

- Optional: create isolated environment with virtualenv

```
$ python3 -m pip install --user -U virtualenv
$ cd $ML_PATH
$ python3 -m virtualenv my_env
$ source my_env/bin/activate   # Linux/macOS
$ .\my_env\Scripts\activate    # Windows
```

- Install required modules

```
$ python3 -m pip install -U jupyter matplotlib numpy pandas scipy scikit-learn
```

- Register env with Jupyter

```
$ python3 -m ipykernel install --user --name=python3
```

- Launch Jupyter Notebook

```
$ jupyter notebook
```

- Create new notebook "Housing.ipynb", run first cell:

```
print("Hello world!")
```

- A notebook consists of cells (code or formatted text)

2.3.2 Download the Data

- Typical environments: data in relational databases or multiple files; need credentials and schema knowledge
- For this project: single compressed file `housing.tgz` with `housing.csv`
- Best practice: write a function to fetch and extract data (useful for updates, automation, multi-machine setups)
- Example function:

```python
import os, tarfile, urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    os.makedirs(housing_path, exist_ok=True)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

- After calling `fetch_housing_data()`, `datasets/housing/housing.csv` is available
- Load the data with pandas:

```python
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

- Function returns a pandas DataFrame with all data