Machine Learning

Compiled Revision Notes

Table of Contents

Week 1: Introduction and Regression

Introduction to Machine Learning

What is Machine Learning?

- **Definition:** A field of study giving computers the ability to learn without being explicitly programmed. It is about changing behavior to improve performance in the future.
- **Core Concept:** Instead of hand-crafting complex rules (lots of `if` statements), we use data to fit parameters of a model.
  - **The Equation:** We want to find the function f in the equation $y = f(x)$, where we figure out the right-hand side by gathering data and fitting parameters.

Types of Learning

- **Supervised Learning:**
  - **Definition:** Learning a function mapping inputs to outputs based on training examples where we already know the correct result (labeled data).
  - **Classification:** The output is a category or discrete label (e.g., Face/No Face, Digit 0–9).
  - **Regression:** The output is a continuous value (e.g., Stock prices, Turkey cooking time).
- **Unsupervised Learning:**

  - Techniques where there is no "right" answer known; the algorithm tries to find structure or patterns in the data.
  - Examples: Clustering (grouping data), Dimensionality Reduction.
- **Reinforcement Learning:** Learning through interaction (e.g., AlphaGo playing games against itself).

The ML Pipeline

To perform Machine Learning, you need 4 components:

1. **The Task:** Define or limit what you are trying to do.
2. **The Experience:** The data (more data = more experience).
3. **The Performance Measure:** A way to measure success.
4. **The Learning Algorithm:** The recipe by which we will improve our performance.

Linear Regression

Variables

- **Independent Variable (x):** The input. It changes independently (e.g., Time).
- **Dependent Variable (y):** The output. It depends on the input (e.g., House Value).
- **Visualizing:** Plot independent variables on the horizontal axis (x) and dependent on the vertical axis (y).

Linear Relationships

- **Definition:** A linear relationship means a change in x always produces the same proportionate change in y.
- **School Math Formula:** $y = mx + c$, where m is the slope and c is the y-intercept.
- **Machine Learning Notation:**
  We scale this up for larger models using **Weights (w)**. Weights are the parameters or coefficients that the model learns during training.

$$y = w_0 + w_1 x$$

  - w_0 = bias or intercept (previously c).
  - w_1 = weight or slope (previously m).

The Turkey Example

- **Scenario:** Predicting cooking time (t) based on weight (w).
- **Linear Model:** $t = mw + c$ (Rule of thumb: 20 mins per pound + 20 mins).
  - *Critique:* Implies you cook a 0 lb turkey for 20 mins (the y-intercept).
- **Power Model (Pief Panofsky):**

$$t = \frac{w^{2/3}}{1.5}$$

- *Comparison:* Linear works well for small weights (1–6 lbs) but diverges significantly at higher weights.

- **Lesson:** Data points might follow a curve. If data is scattered on a curve, a simple linear model is an **inappropriate fit** as it assumes values keep increasing indefinitely.

Training the Model

- **Goal:** Find the best parameters (w_0, w_1) that minimize the error for our training sample.
- **Prediction vs. Observation:**
  - y **(Observation):** The actual observed data point.
  - $\hat{y}$ **(Prediction):** The value predicted by the function.
  - **Residual:** The difference between the prediction $(\hat{y})$ and the observation (y).
- **Cost Function (Mean Squared Error, MSE):**

$$L = \frac{1}{2m} \sum_{i=0}^{m} (\hat{y}_i - y_i)^2$$

  - m: number of samples.
  - Note: The $\frac{1}{2m}$ term is for mathematical convenience to make derivatives simpler.
- **Gradient Descent:**
  - The algorithm typically used to solve for the parameters (w).
  - It performs multiple iterations to find the parameters that give the smallest loss (L).

Multiple Linear Regression

Real-world problems have more than one input feature.

- **Polynomial Regression:** $y = w_0 + w_1 x + w_2 x^2 + ...$ (fits curves).
- **Multiple Linear Regression:** Input x is a vector of features $(x_1, x_2, ...)$.

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + ...$$

Week 2: Generalisation and Model Evaluation

Generalisation

The Goal of Machine Learning

- **Definition:** Generalisation is the model's ability to give sensible outputs to sets of input that it has never seen before.

- It is not enough to perform well on training data; the model must perform well on **previously unseen data**.
- **Measurement:** We often use root-mean-squared (RMS) error to measure performance, but calculating this on training data is misleading because the model might be overfit.

Underfitting vs. Overfitting

- **Underfitting (High Bias):**
  - Occurs when the model is too simple to capture the underlying trend of the data.
  - **Symptoms:** Poor performance on both the training data and the test data. The model has not learned enough.
- **Overfitting (High Variance):**
  - Occurs when the model fits the training data too well, capturing noise rather than the signal.
  - **Symptoms:** Very low training error, but high test error. The model is unreliable on new data.
  - **Confidence:** Overfit models often predict incorrect results with very high confidence.

Bias and Variance Trade-off

- **Bias:** Making assumptions about the underlying model. High bias leads to underfitting.
- **Variance:** Sensitivity to training data. High variance leads to overfitting.
- **The Goal:** Find the sweet spot between high bias and high variance to ensure good generalisation.

Testing and Validation Procedures

Train and Test Split

- To test a model, we split available data into two distinct sets:
  1. **Training Set:** Used to fit the model and learn parameters such as weights.
  2. **Test Set:** Used only to evaluate performance and acts as unseen data.
- **Typical Splits:** 80/20 or 75/25 (Train/Test).
- **Crucial Rule:** The test set must be guarded carefully and used sparingly/carefully. If you use it to make decisions such as choosing model complexity, it is no longer unseen and the evaluation becomes invalid.

Data Leakage

- **Definition:** Leakage occurs when information from the test or validation set unintentionally influences the training process. This leads to overly optimistic performance estimates that fail in the real world.
- **The Cause:** Often caused by non-independent samples. If correlated data appears in both training and testing sets, the model effectively memorizes the answer via hidden context.
- **Example: Bird Call Recordings**
  - *Scenario:* Recordings from Jan 1st (Recording A, Recording B), Jan 2nd, etc.
  - *Bad Split:* Randomly assigning files. Recording A goes to Train, Recording B goes to Test. The model learns background noise rather than bird species.

  - *Correct Approach:* **Group-based splitting.** Split by day or week. If Jan 1st is in Test, all recordings from Jan 1st must be in Test.
- **Key Takeaway:** Ensure split groups are meaningful, such as blocks of time, to test true generalisation.

Model Selection and Hyperparameters

Hyperparameters

- **Definition:** Parameters that are chosen by the engineer, not learned by the algorithm.
- **Examples:**
  - Degree of polynomial (linear vs quadratic vs cubic).
  - Learning rate (a).
  - Number of layers in a neural network.

The Validation Set

- We cannot use the **test set** to choose hyperparameters.
- **Solution:** Split training data further into training and validation sets (often 70/30).
- **Procedure:**
  1. Train different models on the training portion.
  2. Check error on the validation portion.
  3. Pick the hyperparameter with the lowest validation error.
  4. **Final Step:** Evaluate the chosen model on the test set, which has remained hidden.

Cross-Validation (K-Fold)

- **Problem:** A single validation split might be biased.
- **Solution (K-Fold):**
  1. Split data into k equal folds (e.g., $k = 5$ or $k = 10$).
  2. Train on $k - 1$ folds and validate on the remaining fold.
  3. Repeat k times so each fold is used as validation once.
  4. Average the error across all runs to obtain a robust metric.
- **Benefit:** Reduces the risk that a specific random split skews results.

Polynomial Regression

- Complex curved data can be modeled by adding powers of x (e.g., x^2, x^3).
- **Degree (M):** The hyperparameter controlling complexity.
  - Low M (e.g., 1): High bias, underfitting.
  - High M (e.g., 9): High variance, overfitting.
- Cross-validation is used to find the optimal M.

Week 3: Classification & Logistic Regression

Introduction to Classification

What is Classification?

- **Definition:** Unlike regression which predicts continuous values, classification predicts which **category** or group a sample belongs to.
- **Outputs:** The output is restricted to a limited set of possible classes (e.g., 0 or 1).
- **Examples:** Spam detection (Spam/Not Spam), Fraud detection, Medical diagnosis (Malignant/Benign).

Binary Classification

- **Definition:** A problem with only two classes:
  - **Negative Class (0):** The absence of the event (e.g., Benign).
  - **Positive Class (1):** The presence of the event (e.g., Malignant).
- **Categorical Data:** We represent categorical outcomes numerically, typically as 0 and 1.

  Lab Note: Encoding Categorical Features When converting text labels to numbers (Encoding), be careful: - **Label Encoding:** Assigning numbers (0, 1, 2) can mislead the model into thinking there is an order or hierarchy (e.g., $2 > 1$), which might not exist. - **One-Hot Encoding:** A safer approach for non-ordinal data. It splits one feature into multiple binary features (e.g., `Color_Red`, `Color_Blue`), preventing the model from assuming false relationships.

Logistic Regression & The Logit Function

Logistic Regression is a **classification algorithm**, named "regression" only because of its mathematical roots. Despite its name, we are not trying to predict a continuous value. The response variable y is either 0 or 1.

We do predict a continuous value between 0 and 1, but we interpret it as a probability (p) and then apply a threshold to obtain a discrete class label.

The Core Problem with Linear Regression

With standard linear regression:

$$\hat{y} = w_0 + w_1 x_1$$

- Predictions range from $-\infty$ to $+\infty$.
- For binary classification, $\hat{y}$ can easily be < 0 or > 1.
- We want an estimate of probability p strictly between 0 and 1.

Linear regression therefore does not naturally constrain outputs to the valid probability range.

The Sigmoid Function

The **sigmoid function**, also called the **logistic function**, takes any real-valued number and maps it into the range $(0, 1)$.

$$y = f(x) = \frac{1}{1 + e^{-x}}$$

- As $x \to +\infty$, $y \to 1$.
- As $x \to -\infty$, $y \to 0$.
- The function maps values from $(-\infty, +\infty)$ to $(0, 1)$.

Since our linear model output

$$\hat{y} = w_0 + w_1 x_1 + \cdots + w_n x_n$$

also ranges from $-\infty$ to $+\infty$, we apply the sigmoid to map it into a probability:

$$p = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + \cdots + w_n x_n)}}$$

This is the **logistic regression function**. The weights $w_0, \dots, w_n$ are often written as $\beta_0, \dots, \beta_n$ in the literature.

Odds and the Logit Function

The term

$$\frac{p}{1 - p}$$

is known as the **odds**.

Examples: - If $p = 0.5$: $\frac{0.5}{0.5} = 1 : 1$ - If $p = 0.8$: $\frac{0.8}{0.2} = 4 : 1$ - If $p = 0.05$: $\frac{0.05}{0.95} \approx 0.053 : 1$

The odds function is asymmetrical: - If $0 < p < 0.5$, odds lie between 0 and 1. - If $0.5 < p < 1$, odds lie between 1 and ∞.

To address this asymmetry, we take the natural logarithm of the odds:

$$\ln\left(\frac{p}{1 - p}\right)$$

This is called the **logit function**:

$$\text{logit}(p) = \ln\left(\frac{p}{1 - p}\right)$$

The logit function forms the basis of logistic regression.

The linear model in logistic regression is therefore written as:

$$\ln\left(\frac{p}{1-p}\right) = w_0 + w_1x_1 + \cdots + w_nx_n$$

Logit vs. Logistic Function

The logit and logistic functions are inverses of each other.

Feature	**Logistic (Sigmoid) Function**	**Logit Function**
Formula	$f(x) = \frac{1}{1+e^{-x}}$	$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right)$
Purpose	Maps log-odds (x) to probability (p)	Maps probability (p) to log-odds (x)
Input Range	$-\infty < x < \infty$	$0 < p < 1$
Output Range	$0 < p < 1$	$-\infty < x < \infty$
Use in Logistic Regression	Converts linear model output to probability	Converts probability to a linear form

Decision Boundaries

- **Thresholding:** The model outputs a probability (e.g., 0.7). We must choose a threshold (e.g., 0.5) to classify it as Class 1 or Class 0.
- **Trade-off:** This threshold is technically a hyperparameter, often chosen based on the problem (e.g., lowering the threshold for cancer diagnosis to minimize False Negatives).

  Is Threshold a Hyperparameter? - Short Answer: It is complicated. Technically yes, but it is often determined by **human decision** and domain constraints rather than just data. - **Example:** In cancer diagnosis, you might lower the threshold (e.g., to 0.1) to catch more cases. You would rather tolerate False Positives (scaring a healthy patient) than False Negatives (missing cancer).

Training & Loss Functions

Why MSE Fails for Classification

- **Non-Convexity:** If you use Mean Squared Error (MSE) with the Sigmoid function, the resulting cost function is **non-convex**.
- **Result:** The graph looks “wavy” with many local minima, making it impossible for Gradient Descent to reliably find the global minimum (the best weights).

Log Loss (Binary Cross-Entropy)

- We use **Log Loss** instead. It is **convex**, ensuring a single global minimum that makes optimization efficient.

- **The Logic:**
  - If correct answer is 1: We want predicted p to be close to 1. If p is low, error is massive ($-\log(p)$).
  - If correct answer is 0: We want predicted p to be close to 0. If p is high, error is massive ($-\log(1-p)$).
- **Formula:**

$$L(w) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot \log(\hat{y}_i) + (1-y_i)\cdot \log(1-\hat{y}_i)]$$

Metrics & Evaluation

L1 vs. L2 Norms

These are ways to measure the "magnitude" or distance of a vector, often used in calculating errors.

- **L1 Norm (Manhattan):** The sum of absolute values. $||x||_1 = \sum |x_i|$
- **L2 Norm (Euclidean):** The square root of the sum of squared values (straight line distance). $||x||_2 = \sqrt{\sum x_i^2}$

Regression Metrics

When evaluating regression models (predicting continuous values), we use several key metrics:

- **MSE (Mean Squared Error):**
  - **Formula:** $\frac{1}{n}\sum(y_i - \hat{y}_i)^2$
  - **Use:** Standard for **Training** loss because it is differentiable (convex).
  - **Pros/Cons:** Highly sensitive to outliers because it squares the error (large errors become huge).
- **RMSE (Root Mean Squared Error):**
  - **Formula:** $\sqrt{MSE}$
  - **Use: Model Evaluation**.
  - **Pros:** It returns the error to the same units as the target variable (e.g., dollars, minutes), making it easier to interpret.
- **MAE (Mean Absolute Error):**
  - **Formula:** $\frac{1}{n}\sum |y_i - \hat{y}_i|$
  - **Use: Model Evaluation**.
  - **Pros/Cons:** Less sensitive to outliers than MSE. However, it is **not differentiable** at 0 (has a sharp "kink"), so it cannot be used as a loss function for training algorithms like Gradient Descent.
- R^2 **(Coefficient of Determination):**
  - **Formula:** $1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$
  - **Interpretation:** Represents the proportion of variance in the dependent variable explained by the model (Unitless, max value is 1).

Classification Metrics

- **Accuracy:** Fraction of correct predictions. Good for balanced classes, misleading for imbalanced ones.
- **Confusion Matrix:** A table comparing Actual vs. Predicted values (TP, TN, FP, FN).
- **Precision:** $\frac{TP}{TP+FP}$ (Accuracy of positive predictions).
- **Recall:** $\frac{TP}{TP+FN}$ (Ability to find all positive instances).
- **F1-Score:** Harmonic mean of Precision and Recall. Best for imbalanced datasets.

Trade off between Precision and Recall (PR Curve and PR AUC) Precision and Recall usually trade off. Increasing the threshold increases Precision but lowers Recall (and vice versa). The PR Curve and the Area under the curve (AUC), can also be a good single number estimator of model performance.

> **Note on Optimization** We typically **train** using one metric (like Log-Loss for classification or MSE for regression because they are differentiable/convex) but **evaluate** using others (like Accuracy, F1-Score, or MAE) that are more interpretable for humans.

Week 4: k-Nearest Neighbours, Feature Engineering & Tree Ensembles

k-Nearest Neighbours (kNN)

The Algorithm

- **Definition:** kNN is a simple, two-part algorithm used for either classification or regression.
- **Procedure:**
  1. Given a target instance (x_j), calculate the distance to all recorded training cases (x_i).
  2. Retrieve the k most similar (nearest) recorded cases.
  3. **For Classification:** Take the maximum of k votes (the most common category among the neighbors).
  4. **For Regression:** Calculate the "average" across these k cases.

Distance Metrics

- We typically use **Euclidean distance** to calculate the "nearest" neighbor.
- **Formula:** $d(t, s) = \sqrt{(t_1 - s_1)^2 + ... + (t_p - s_p)^2}$.
- **The Scale Problem:** Scale matters significantly. If one predictor/feature has a much larger value range than another, it will disproportionately influence the distance measurement.

Choosing k

- $k = 1$ **(1-Nearest Neighbour):** Looking at only the single closest point tends to overfit the training data and captures noise.
- **Higher k (e.g., $k = 5$):** Taking a vote among multiple neighbors smooths out the decision boundary and reduces overfitting.
- **The Trade-off:** If k is too small, the model overfits. If k is too large, the model underfits reality.

- **How to choose:** We treat k as a hyperparameter and select the best value using cross-validation (e.g., 10-fold cross-validation).

Lazy vs. Eager Learning

- **Lazy Learning (e.g., kNN):** Defers computation until a prediction is needed.
  - *Pros:* Very quick to train (it just stores the data), less memory usage during training.
  - *Cons:* Prediction is slow, relies heavily on training data during prediction, uses more memory during prediction.
- **Eager Learning (e.g., SVM, Neural Networks, Decision Trees):** Precomputes a model during the training phase.
  - *Pros:* Faster predictions, less memory usage during prediction, less dependent on original data once trained.
  - *Cons:* Slower and more memory-intensive during the training phase.

Feature Engineering & Preprocessing

Feature Scaling

Because distance metrics like Euclidean distance are heavily influenced by the scale of the data, we must bring features into a similar range.

Normalization vs. Standardization

Feature	Min-Max Scaling (Normalization)	Standardization
Output Range	Fixed (0 to 1)	Not bounded
Outlier Handling	Very sensitive	More robust
Distribution	Changes the shape	Maintains shape (mostly)
Common Use Case	When you need exact boundaries from min to max value	When you need a “standard normal curve” (Mean 0, Std Dev 1)

Pipelines

- We should avoid altering the entire raw dataset directly before training, as future data would also need to be manually transformed.
- **Solution:** Build the transformation directly into the model using `make_pipeline`.
- Any data that goes into the pipeline automatically passes through the transformation step (e.g., `StandardScaler()`) before reaching the classifier step.

Feature Engineering & Custom Transforms

The features we provide to a model drastically dictate what it can learn. We can design custom features to give the model better predictive power.

- **Custom Calculations:** Using tools like `FunctionTransformer`, we can add derived features, such as calculating a point's radius/distance from the center $(0, 0)$ to help classify circular data.
- **1D to 2D Signals:** Converting 1-dimensional signals into 2-dimensional representations. For example, converting audio data or stock market data into spectrograms allows models (especially computer vision models) to find complex visual patterns in the data.

The Curse of Dimensionality

The Problem with High Dimensions

- **Sparsity:** As the number of dimensions (features) increases, data points spread out, increasing sparsity.
- **Failing Distance Metrics:** In high dimensions (e.g., >10), Euclidean distance becomes unhelpful because all vectors become almost equidistant from the search query vector. This misleads nearest-neighbor algorithms.

The Bioinformatics Problem

- In fields like bioinformatics (e.g., genome sequencing), datasets often have very few subjects (e.g., 1000) but millions of features (genes).
- This creates a mathematical problem similar to simultaneous equations: if you have millions of unknown variables (features) but only 1000 equations (subjects), you cannot reliably solve the system.
- **The Solution:** This issue is typically addressed using **Regularization** (especially L1 Regularization, which forces the model to ignore irrelevant features by shrinking their weights to zero).

Dimensionality Reduction Techniques

To fix the curse of dimensionality before applying algorithms like kNN, we can use:

- **Feature Grouping:** Grouping common features based on domain knowledge (e.g., averaging 365 daily weather readings into 12 monthly averages).
- **Extraction/Embedding Algorithms:** Using techniques like Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA) to compress the data into a lower-dimensional space.

Decision Trees & Ensembles

Decision Trees

- **How it works:** Decision Trees mimic human decision-making by incrementally splitting a dataset into smaller subsets based on feature values (e.g., "Is it a mammal? -> Does it have stripes? -> Tiger").
- **Anatomy of a Tree:**
  - **Root Node:** First split, entire dataset.

  - **Decision Node:** Internal split based on a feature condition.
  - **Leaf Node (Terminal Node):** Final node that outputs a prediction.
  - **Depth:** Number of split layers.

How Trees Learn: Splitting & Purity

- At each node, the algorithm selects the feature and threshold that produce the **purest** child nodes.
- It evaluates all possible splits and chooses the one that minimises impurity.

Impurity Measures:

- **Gini Impurity**
- **Entropy (Information Gain)**

Gini Index (Binary Case):

$$\text{Gini} = 1 - (p_{yes}^2 + p_{no}^2)$$

- $0 \rightarrow$ perfectly pure node.
- $0.5 \rightarrow$ maximum impurity (50/50 split).
- Lower Gini = better split.
- For a full split, compute the **weighted average** of child-node Gini values and choose the lowest.

Continuous Features:

1. Sort feature values.
2. Use midpoints as candidate thresholds.
3. Compute impurity for each.
4. Select threshold with minimum impurity.

Overfitting:

- Without limits, trees split until leaves are perfectly pure.
- Leads to memorisation and poor generalisation.
- Controlled using hyperparameters such as:
  - `max_depth`
  - `min_samples_split`
  - `min_samples_leaf`

Ensemble Learning

- **Concept:** Combine multiple models instead of relying on a single tree.
- Leverages the "wisdom of the crowd" for improved accuracy and robustness.

Bagging & Random Forests

- **Structure:** Models trained in **parallel**.

- **Bagging:**
  - Train models on bootstrap samples (random samples with replacement).
  - Aggregate via majority vote (classification) or averaging (regression).
  - Primarily **reduces variance**.
- **Random Forests:**
  - Bagging + random subset of features at each split.
  - Decorrelates trees so they do not all make the same mistakes.
- **Goal:** Strong variance reduction, robust, hard to overfit.

Boosting

- **Structure:** Models trained **sequentially**.
- **How it works:**
  - Tree 1 makes predictions.
  - Tree 2 focuses more on misclassified samples.
  - Tree 3 focuses on remaining errors, and so on.
- **Popular Algorithms:** AdaBoost, Gradient Boosting, XGBoost, LightGBM.
- **Goal:** Primarily **reduces bias**.
- Can overfit if not properly tuned.

Bagging vs. Boosting

- **Bagging:** Parallel, reduces variance, resistant to overfitting.
- **Boosting:** Sequential, reduces bias, more prone to overfitting but often higher performance when tuned well.