Week 4 - ML

Start Date	@24 November 2025
Weeks	Week4

Data Processing (Chapter 4)

- Critical → Data is examined → preprocessed before learning

Missing Data

- Very common in real-world scenario, where data is missing and dataset is "not complete"
- Missing values → Seen as `NaN` → "Not a Number"
  - Most computational tools won't be able to process `NaN`

> ⭐ Crucial we take care of missing values before further analysis

Missing Values in Tabular Data

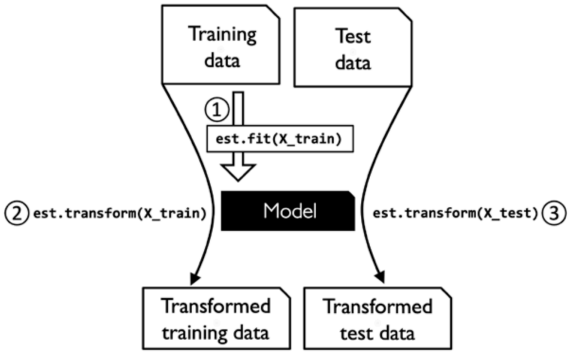
- **Use NumPy arrays when possible** → most scikit-learn functions support DataFrames
- **Removing data** → May seem convenient, but comes with disadvantages
  - Too many samples removed → Unreliable analysis
  - Too many feature columns removed → Risk of losing valuable information → Classifier needs discrimination between classes.

Imputing Missing Values

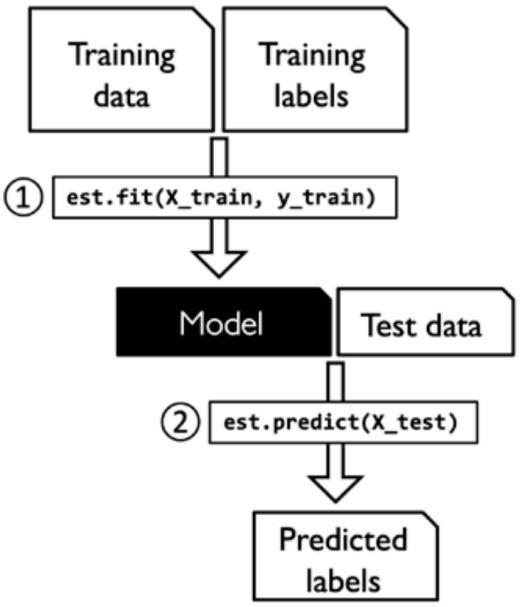
- **Interpolation Techniques**
  - **Estimates unknown values within the range of known data point.**
- Most common interpolation technique → **Mean Imputation** → Replace missing values with mean value of feature column

Scikit-Learn Estimator API

- **Simple Imputer →** Scikit-learn Transformer API → `SimpleImputer()`
  - Replaces the `NaN` values with a specified placeholder
- **Two essentials methods of transformer →** `Fit` **+** `Transform`
  - `Fit` → Learn parameters from training data
  - `Transform` → Uses parameters to transform data
- Data array to be transformed → Same number of features as data used to fit.



Scikit-Learn API for transformation



Using the Scikit-Learn API for predictive models such as classifiers

- Estimators (Chapter 3) → Can have both `fit` and `transform` method too.
  - `Fit` also used to learn parameters of a model → when estimators trained for classification

Categorical Data

- Real-world data → Will include more than just numerical data
- Categorical Data

    - Ordinal Features → Categorical values that can be sorted e.g. T-shirt (S, M, L, XL)
    - Nominal Features → Don't have a specific order (no hierarchy)

Ordinal Features

- Convert categorical string to integer for learning algorithm → simple dictionary mapping
- Define mapping manually
- Can transform back to string → reverse mapping

Encoding Class Labels

- Many ML libraries → Require class labels encoded as integer values
- Scikit-Learn classification converts → but good practice to provide class label as integers
- Class labels → Not ordinal
- Scikit-Learn → Label Encoder

One-Hot Encoding on Nominal Features

- Scikit-Learn estimators for classification → Class label treated as categorical data → Nominal → Used `LabelEnconder`

```
X = df[['color','size','price']].values

color_le = LabelEncoder()

X[:, 0] = color_le.fit_transform(X[:, 0])

X
```

✓ 0.0s

```
array([[1, nan, 10.1],
       [2, nan, 13.5],
       [0, nan, 15.3]], dtype=object)
```

- Image:
  - Blue = 0
  - Green = 1
  - Red =2

⭐ **Don’t feed this info into classifier!! (Image above)**

- Most common mistake in dealing with categorical data
- Common classification models → **Assume green is larger than blue, and red larger than blue**
- **The workaround:**
  - Create new primary feature for each unique value in nominal feature column
  - Convert the new features → Binary values for indication → e.g. Blue=1, Green=0, Red=0
- `ColumnTransformer` → Multi-feature array
- One-Hot Encoding → Introduces multi-collinearity → Issue for some models → Methods that require matrix inversion

- Features → Highly correlated → Matrices computationally difficult to invert
    - Remove on feature column (that won't mess with the data)

Other Options Than One-Hot Encoding:

- Useful working with categorical features that have High Cardinality (a large number of unique labels)
- **Binary Encoding:** Produces multiple binary features similar to One-Hot but requires fewer feature column
- **Count or Frequency Encoding:** Replaces label of each category by number of times or frequency it occurs in training set

Partitioning Training and Test Datasets

- `train_test_split` → Scikit-Learn → Convenient for splitting data
- Appropriate ratio for partitioning:
    - Smaller test set → more inaccurate estimation of generalisation error
    - **Most common split** → **60:40, 70:30, 80:20** → depends on the size of dataset
        - Large datasets → **90:10 or 99:1** is very common and appropriate

Features On The Same Scale

- **Feature Scaling** → **Critical step** in preprocessing pipeline
- Decision Trees + Random Forest → Don't need to worry about feature scaling → Scale-invariant
- Two ways to bring onto same scale: Normalization + Standardization
- Most often → Normalization means → Rescaling feature to range (0,1) → **Min-Max Scaling**

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$$

- **Min-Max Scaling** → **Commonly used technique** → Useful when we need values in bounded interval

- **Standardization** → More practical for many machine learning algorithms → e.g. Gradient Descent
  - Many Linear Models → Logistic Regression + SVM → Initialise weight to `0` or small values close to `0`
  - Centre feature column at mean `0` and std `1` → Standard Normal Distribution (zero mean + unit variance)
  - Standarization → does not change shape of distribution
  - Maintains useful information = outliers → Algorithm less sensitive

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

- μ_x = Sample mean of particular feature column
- σ_x = Standard deviation

Input	Standardized	Min-max normalized
0.0	-1.46385	0.0
1.0	-0.87831	0.2
2.0	-0.29277	0.4
3.0	0.29277	0.6
4.0	0.87831	0.8
5.0	1.46385	1.0

Comparison between standardization and min-max normalization

- `StandardScaler` → Fit only once on training data
- `RobustScaler` → Scikit-Learn
  - Helpful + recommended for small datasets that contain many outliers

    - If machine learning algorithm prone to overfitting
- Each feature column → Removes median value and scales dataset according to 1st + 3rd quartile of dataset

Selecting Meaningful Features

- **Overfitting** → Model fits to parameters too closely, but does not generalise well with new data → Model has high variance
- **Common solutions for Generalization error:**
    - Collect more training data
    - Introduce penalty for complexity via regularization
    - Choose a simpler model with fewer parameters
    - Reduce dimensionality of data

L1 and L2 Regularisation As Penalties Against Model Complexity

- L2 Regularization → Reduce complexity of model → penalise large individual weights

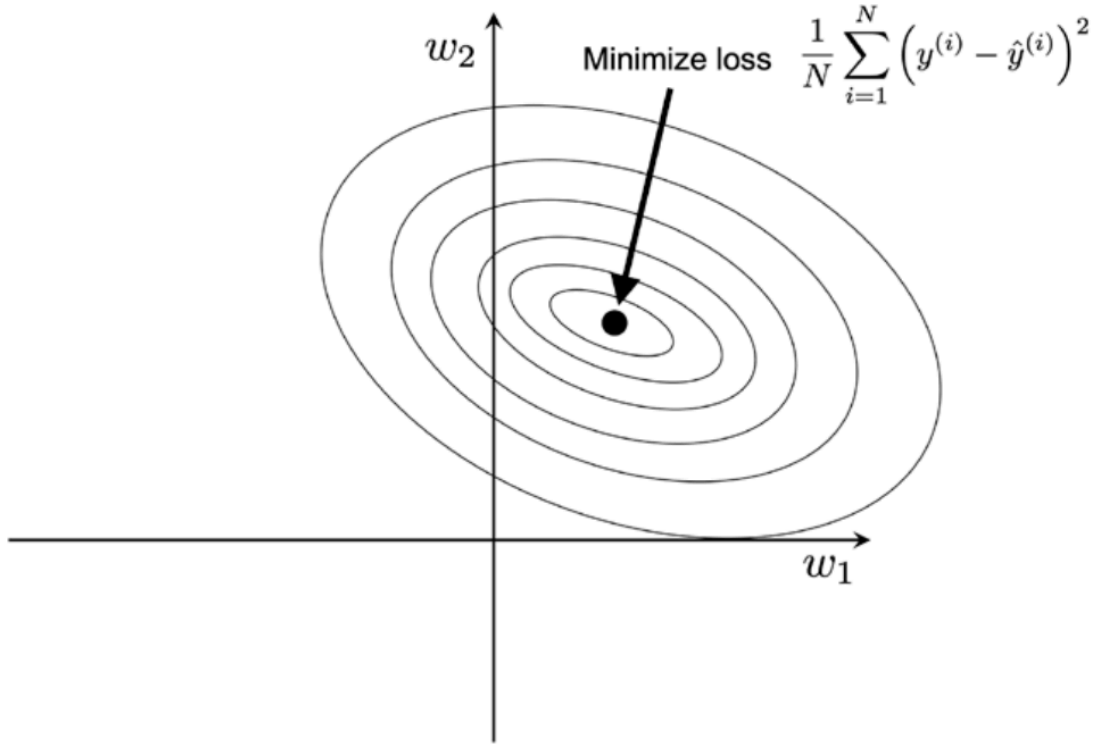
$$L2 : ||w^2|| = \sum_{j=1}^{m} w_j^2, L1 : ||w||_1 = \sum_{j=1}^{m} |w_j|$$

- w = Weight vector
- Replaced square of weights with sum of absolute values of weight
- L1 → Sparse feature vectors + most feature weights will be zero
    - Sparsity → Useful in practice (If we have high-dimensional dataset with many irrelevant features)
    - Technique for feature selection

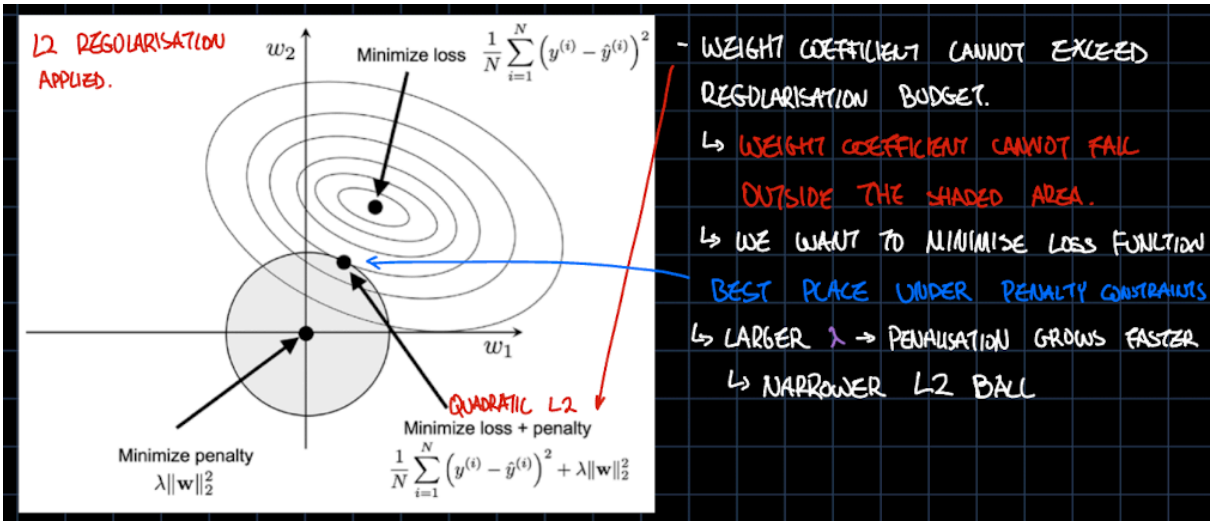
L2 - Geometrically

- L2 → adds penalty term to loss function → less extreme weight value
- Regularization → Penalty term to loss function

- Increasing regularization with λ → shrink towards zero + decrease dependance of model on training dataset



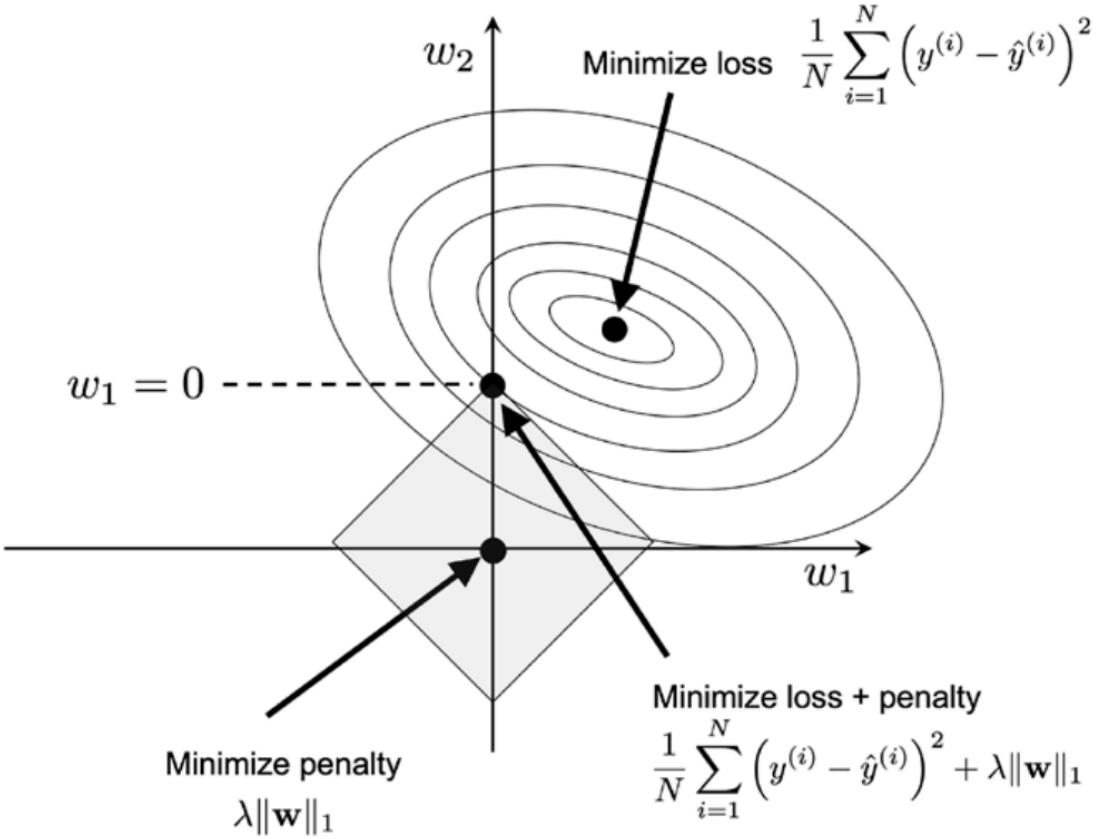
Minimising the mean squared error loss function



Apply L2 Regularization to the loss function

- In Summary → Our goal → Minimise sum of unpenalised loss + penalty term
  - Adding Bias + preferring simpler model to reduce variance in absence of sufficient training data

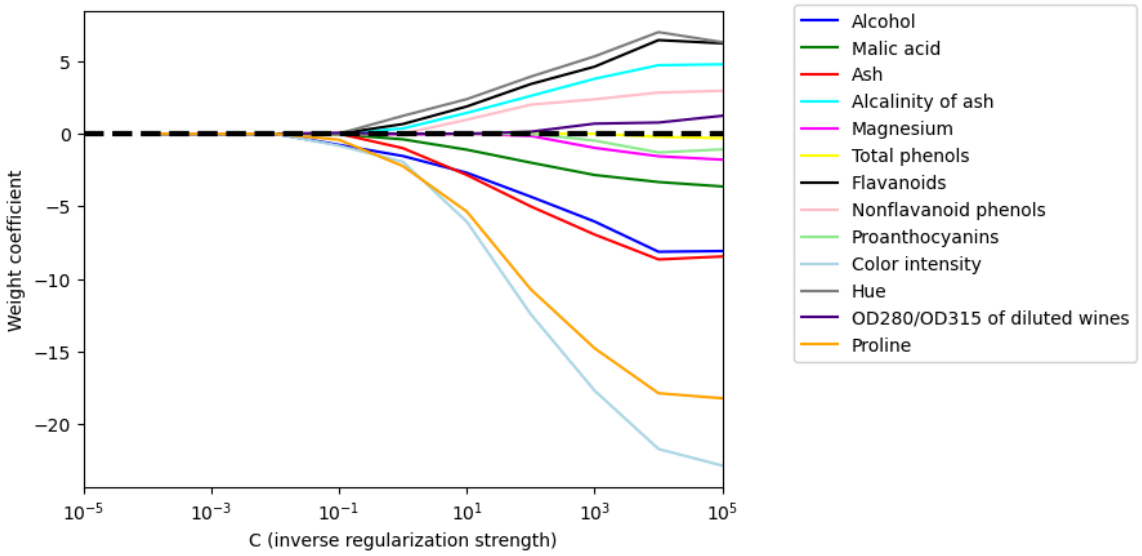
Sparse Solutions With L1



Applying L1 regularisation to the loss function

- Wine Dataset → 13 weights for each row → each weight multiplied by the respective feature:

$$z = w_1x_1 + \cdots + w_mx_m + b = \sum_{j=1}^{m} x_jw_j + b = w^Tx + b$$



Impact of the value of the regularization strength hyperparameter C

Sequential Feature Selection Algorithms

- Alternative way to reduce complexity + overfitting → Dimensionality Reduction via feature selection → useful for unregularised models
- Dimensionality Reduction:
  1. **Feature selection** - Select subset of original features
  2. **Feature extraction** - Derive information from features et to construct new feature subspace

Feature Selection

- Sequential feature selection algorithm → family of Greedy Search algorithm
  - Reduce initial d -dimensional feature space to k -dimensional feature subspace $k < d$
  - **Goal**:
    1. Automatically select subset of features, that are most relevant to problem
    2. Improve computational efficiency
    3. Reduce generalization error of model, which removes irrelevant feature or noise
       a. Useful for algorithms without regularization

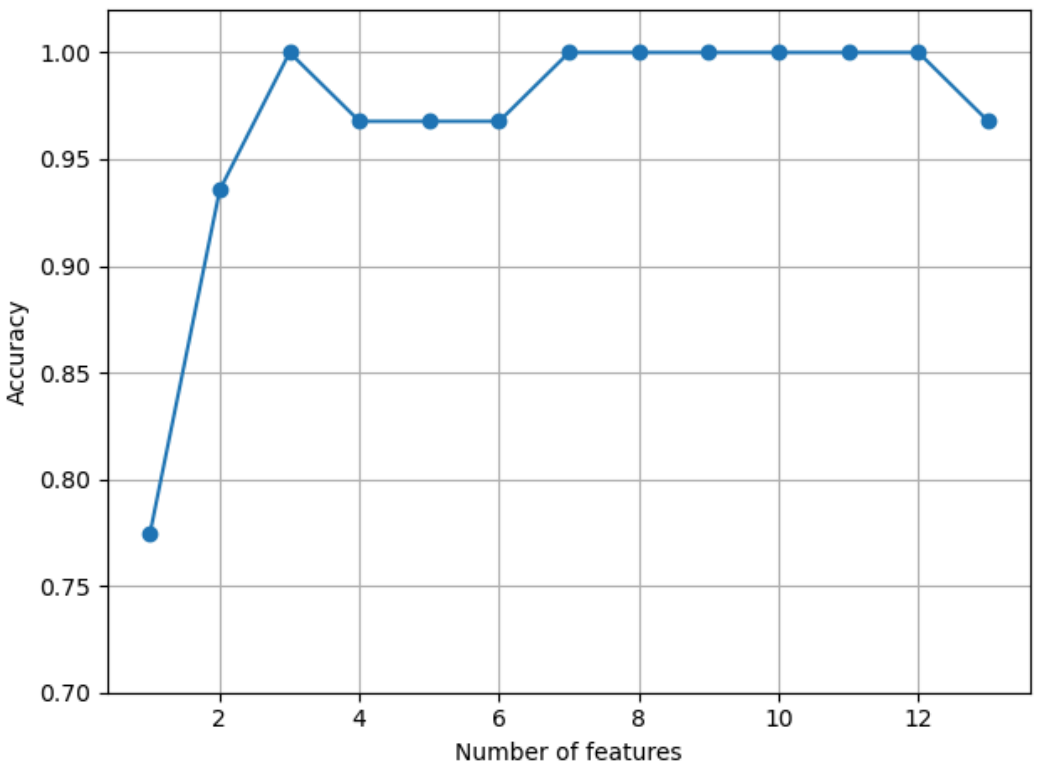
- Classic example - **Sequential Backward Selection (SBS)**
  - Reduce dimensionality to initial feature subspace with minimum decay in performance to classifier to improve upon computational efficiency.
- SBS can improve predictive power in some cases

Greedy Search Algorithm

- Attempts to find the most promising path from a give starting point to a goal.
- Evaluates costs of all paths and goes with lowest cost path. This process repeats until the goal reached.

SBS Algorithm

- Sequentially removes features from full subset → until new feature subspace has the desired number of features.
- Feature to be removed at each stage → criterion function J → Want to minimise this
- At each stage, eliminate the feature that causes the least performance loss, after removal.
- **4 Steps:**
  - Define $k = d$, where d = Dimensionality of full feature space
  - Define x^-, where we want to maximise $x^- = argmax J(X_k - x)$ $x \in X_k$
  - Remove feature x^-, where feature set: $x_{k-1} = X_k - x^- \quad k = k - 1$
  - Terminate k where k = number of desired features, otherwise go back to step 2



Impact of number of features on model accuracy

- KNN Classifier → Improves validation dataset as we reduce number of features → Due to decrease in the Curse of Dimensionality
  - Event he closest neighbours seem far away

> ⭐ Here we covered L1 Regularisation, where we zero out irrelevant features via Logistic Regression, how to use SBS for feature selection and apply it to a KNN algorithm

Feature Importance With Random Forest

- Another useful approach → Feature selection with **Random Forest**
  - **Measure feature importance without making any assumptions about whether data is linearly separable**
  - Feature importance values already collected → Scikit-Learn Random Forest → `feature_importance_`
- **Careful** → If two or more features are highly correlated → One feature may be ranked very highly, whilst information on other features may not be fully

captured.

    - This isn't important if we are only interest in predictive performance of the model

- Scikit-Learn → `SelectFromModel` object → Selects based on user-specific threshold after model fitting
    - Useful with Random Forest → Intermediate step in pipeline → Connects different preprocessing steps with estimator