

Machine Learning, W1 to W3

W1 Intro to ML

What is ML

- Samuel (1959): computers learn without being explicitly programmed.
- Mitchell: a program learns if its performance P on task T improves with experience E .
 - Weather app: E = past weather data. T = predict today's weather. P = was the forecast right.
- ML around us: face recognition (Instagram), recommendations (Netflix), spam filter (Gmail), fraud detection (PayPal), generative AI (ChatGPT).
- Why it matters: data keeps growing and problems get more complex. Hand-written rules are costly and error prone.

AI vs ML

- AI = any technique that makes a program sense, act and adapt to mimic human behaviour.
- ML = part of AI. The machine improves by learning from experience.
 - Chess program with hand-written rules: AI, not ML. Nothing is learned.
 - Spam filter trained on my mails: ML. It improves with data.

	AI	ML
Works as	program doing smart work	machine learning from data
Goal	simulate intelligence	learn a task, maximise performance
Does	decision making	learns new things from data
Leads to	intelligence, wisdom	knowledge



Non-ML way vs ML way

- Non-ML: data + rules written by a human $\rightarrow$ answers.
 - Fruit: IF smooth and orange THEN orange. IF smooth and red THEN apple.
- ML: data + answers $\rightarrow$ learning algorithm $\rightarrow$ rules (the model).

3 branches

- Supervised: data has labels (the right answer). Classification, regression.
 - Mails already marked spam or not spam.
- Unsupervised: no labels. clustering, association.
 - Purchase records with no group names.
- Reinforcement: learns from rewards. A game program that improves by winning and losing. Not the focus of this course.

4 kinds of task

- Association (basket analysis): things that happen together. $P(Y|X)$.
 - $P(\text{spread} | \text{bread}) = 0.7$: 70% of bread buyers also buy chocolate spread.
 - With customer attributes D : $P(Y|X, D)$.
- Classification: predict a label.
 - Rule: IF $X > \theta_1$ AND $Y > \theta_2$ THEN A. The boundary is the discriminant.
 - IF temperature > 31 AND humidity $> 90\%$ THEN sunny.
- Regression: predict a number. $y = wx + w_0$, w slope, w_0 intercept.
 - ML score from Statistics, Discrete Math and Programming scores. Training finds each weight.
- Clustering: group similar items; no labels. Learn the regularities in the input.
 - Group customers by past transactions. Find similar pictures.
- Which one? Number $\rightarrow$ regression. Label $\rightarrow$ classification. No labels $\rightarrow$ clustering.
 - Bought together $\rightarrow$ association.

Python stack (W1 has no notebook)

- NumPy: arrays. SciPy: numerical algorithms. matplotlib: plots. pandas: tables.
- scikit-learn: ML algorithms.

W2 Regression

Regression

- Supervised. Predicts a continuous value from one or more variables.
- X = independent variables = features. Y = dependent variable = target.
 - Boston data: the 13 columns CRIM to LSTAT are X . House price is Y .
- Uses: wind speed from temperature and humidity. House price from rooms and land size. Stock price. Car MPG from weight and horse power.

Simple linear regression

$$\hat{y} = \theta_0 + \theta_1 x_1$$

- $\hat{y}$ predicted variable. θ_0 intercept. θ_1 slope. x_1 the single predictor.
- Training = find the θ whose line fits the points best.



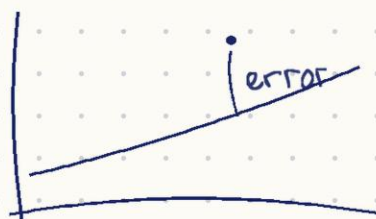
How good is the model

$$\text{error} = y - \hat{y}$$

- Real $y = 16.5$, predicted $\hat{y} = 15$. Error = 1.5.

$$\text{MSE} = (1/n) \sum (y - \hat{y})^2$$

- Squaring: plus and minus errors cannot cancel, and big errors are punished more.
- Objective: find the θ that minimise the MSE.
- Also: MAE = mean of $|y - \hat{y}|$. R^2 = share of variance explained, 1 is perfect.



Least squares by hand

- $x = 1, 2, 3, 4$. $y = 3, 4, 6, 8$. Means: $\bar{x} = 2.5$, $\bar{y} = 5.25$.

$$\theta_1 = \sum (x - \bar{x})(y - \bar{y}) / \sum (x - \bar{x})^2$$

$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$

x	y	$x - \bar{x}$	$y - \bar{y}$	product	$(x - \bar{x})^2$
1	3	-1.5	-2.25	3.375	2.25
2	4	-0.5	-1.25	0.625	0.25
3	6	0.5	0.75	0.375	0.25
4	8	1.5	2.75	4.125	2.25
sum				8.5	5

- $\theta_1 = 8.5 / 5 = 1.7$. $\theta_0 = 5.25 - 1.7 \times 2.5 = 1.0$. So $\hat{y} = 1.0 + 1.7x$.

x	y	$\hat{y}$	error	error ²
1	3	2.7	0.3	0.09
2	4	4.4	-0.4	0.16
3	6	6.1	-0.1	0.01
4	8	7.8	0.2	0.04

- Sum = 0.30. MSE = 0.30 / 4 = 0.075.

Multiple linear Regression

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

$$\hat{y} = \theta^T X \quad (x_0 = 1)$$

- Price = $\theta_0 + \theta_1 \text{CRIM} + \theta_2 \text{LSTAT} + \theta_3 \text{RM} + \theta_4 \text{DIS}$

- Slide example: $\theta = [-4.56, -0.09, -0.66, 5.47]$ Row 8: CRIM 0.211124, LSTAT 29.93, RM 5.631.

$$\text{price} = -4.56 - 0.019 - 16.46 + 30.80 = 9.76$$

- real price 16.5, so the error is 6.74.

- Signs: more crime or more LSTAT $\rightarrow$ lower price. More rooms $\rightarrow$ higher price.

Non-linear regression

- Use it when a straight line cannot fit. Price vs crime drops fast, then flattens.

- Shapes: linear, quadratic, cubic, polynomial.



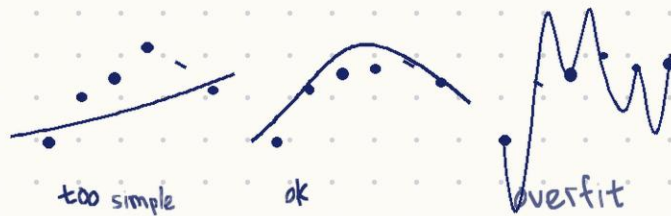
Polynomial regression

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

- Trick: rename $x_1 = x$, $x_2 = x^2$, $x_3 = x^3$. Now it is multiple linear regression, solved by least squares.

- Goal: minimise the sum of squared residuals.

- Overfitting: degree too high. The model memorises the training points and fails on new ones. Train error falls, test error rises.



True non-linear

- Non-linear in θ . Cannot be renamed into a line. Solved by iterating, no closed formula.

$$\hat{y} = \theta_0 + \theta_2^2 x$$

$$\hat{y} = \log(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

$$\hat{y} = \theta_0 / (1 + \theta_1^x (x - \theta_2)) \text{ S-curve}$$

Regularization

- Prevents over-fitting by penalising large coefficients.
- Ridge: squared penalty. Shrinks all coefficients. Use when many features are highly correlated.
- Lasso: forces some coefficients to exactly 0, so it selects features. Use when only a few features matter.
 Ridge: $\text{MSE} + \alpha(\theta_1^2 + \theta_2^2 + \dots)$
 Lasso: $\text{MSE} + \alpha(|\theta_1| + |\theta_2| + \dots)$
- α = penalty strength. $\alpha = 0$ is plain regression. Pick α with GridSearchCV.
- CAREFUL: the slide says Ridge (L2) and Lasso (L1). The standard is Ridge = L2, Lasso = L1.

Lab code (Boston, scikit-learn)

- Every model is the same 4 steps: pick, fit, predict, score.

Load and split

```
import numpy as np, pandas as pd
url = "https://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(url, sep="t", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
bdf = pd.DataFrame(np.c_[data, target], columns=names + ['target'])
dat = np.random.rand(len(bdf)) < 0.8
train, test = bdf[dat], bdf[~dat]
```

- 13 feature columns plus the price. About 80% of rows train, the rest test. No seed, so numbers change every run.

Simple linear regression

```
from sklearn import linear_model
train_x = np.asarray(train[['CRIM']])
train_y = np.asarray(train[['target']])
regr = linear_model.LinearRegression()
regr.fit(train_x, train_y)
print(regr.coef_, regr.intercept_)
test_x = np.asarray(test[['CRIM']])
test_y = np.asarray(test[['target']])
test_y_ = regr.predict(test_x)
print(np.mean(np.absolute(test_y_ - test_y))) # MAE
print(np.mean((test_y_ - test_y)**2)) # MSE
```

- fit needs 2-D X, hence the double brackets. coef_ is θ_1 , intercept_ is θ_0 .
- Lab: coef -0.408, intercept 23.99, MAE 5.92, MSE 63.20.

Multiple linear regression

```
cols = ['CRIM', 'LSTAT', 'RM']
x_tr = np.asarray(train[cols])
y_tr = np.asarray(train[['target']])
regr.fit(x_tr, y_tr)
x = np.asarray(test[cols])
y = np.asarray(test[['target']])
print(regr.score(x, y)) # R2
```

- Lab: coeffs -0.094, -0.605, 5.07. Intercept -1.52. R^2 0.73.

Polynomial, degree 3

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=3)
train_x_poly = poly.fit_transform(train_x)
clf = linear_model.LinearRegression()
clf.fit(train_x_poly, train_y)
test_y_ = clf.predict(poly.transform(test_x))
```

- Builds the columns 1, x, x^2 , x^3 . Test rows use transform, not fit_transform.
- Lab: MAE 5.33, MSE 54.19. Better than the straight line (63.20).

Ridge and Lasso

```
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import GridSearchCV
alphas = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
ridge = GridSearchCV(Ridge(), alphas, cv=5, scoring='neg_mean_absolute_error')
ridge.fit(train_x, train_y)
```

- ```
print(ridge.best_params_)
test_y_ = ridge.best_estimator_.predict(test_x)
```
- Tries each alpha on 6 folds and keeps the best. Lasso: same, with Lasso().
  - Lab: Ridge (alpha 100) MAE 7.42. Lasso (alpha 1) MAE 7.39. A different random split, so do not compare with the rows above.

### Scores

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,
 r2_score
mean_absolute_error(test_y, test_y_)
mean_squared_error(test_y, test_y_)
r2_score(test_y, test_y_)
```

## W3 Time Series

### Time series

- Data in time order. The model reads the past and predicts future values.
  - Every row has a timestamp. It tracks variables across time. Points depend on each other, so never shuffle.
  - Univariate: time is the only independent variable. Monthly sales alone.
  - Multivariate: more variables. Sales plus weather.
- Uses: forecasting (next month's sales), anomaly detection (odd network activity), pattern analysis (weekend peaks).

### Autocorrelation

- How a series relates to its own past. Lag  $k = k$  steps back.
- ACF at lag  $k =$  correlation between  $y(t)$  and  $y(t-k)$ .
- A tall bar at lag 12 = a yearly season.
- PACF = the same, with the shorter lags removed.
- By hand: 12, 15, 14, 18, 20, 19, 23. Mean 17.29.
  - Deviations: -5.29, -2.29, -3.29, 0.71, 2.71, 1.71, 5.71.
  - Top = sum of  $dev(t) \times dev(t-1) = 33.63$ . Bottom = sum of  $dev^2 = 87.43$ .
  - $ACF(1) = 33.63 / 87.43 = 0.385$ .

### Trend, seasonality, cycle

- Trend: long-term rise or fall. Sales grow as the customer base grows.
- Seasonality: repeats at a fixed gap. Electricity up every evening. Ice cream up every summer.
- Cycle: rises and falls, gap not fixed. Economic booms and recessions.

- Test: can I say how many steps until it repeats? Yes = season. No = cycle.

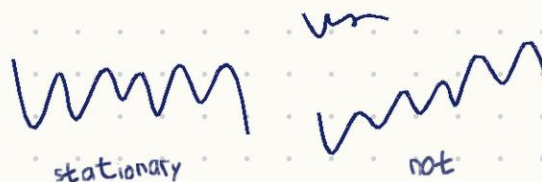


## Stationarity

- The statistics stay stable over time.

3 conditions: constant mean. Constant variance. Covariance depends on the lag, not the time.

- Use that climbs every month: not stationary. Use that wobbles around one average: stationary.
- Fix a trend by differencing:  $y(t) - y(t-1)$ . Once is  $d=1$ .  
-12, 15, 14, 18, 20, 19, 23  $\rightarrow$  +3, -1, +4, +2, -1, +4.
- ADF test:  $p < 0.05$  means stationary.



## ARIMA(p, d, q)

- Autoregressive Integrated Moving Average. Forecasts from past values and past errors.

-  $p$  = autoregressive order: number of lags (past values).

-  $d$  = differencing: times differenced to become stationary. Usually 1.

-  $q$  = moving average order: number of past errors.

- ARIMA(1,1,1): 1 past value, differenced once, 1 past error.

- The lab tries many orders and keeps the lowest AIC (fit, with a penalty for extra parameters).

- Steps: visualise, ACF and PACF plots, seasonal check, stationary check. pick  $p, d, q$ .

## SARIMA and SARIMAX

- SARIMA(p,d,q)(P,D,Q,s): ARIMA plus a seasonal copy.  $s$  = season length, 12 for monthly data.

-  $P$  seasonal AR order.  $D$  seasonal differencing.  $Q$  seasonal MA order.

- Seasonal differencing:  $y(t) - y(t-12)$ .

- SARIMAX: SARIMA plus  $x$ , outside (exogenous) variables. A holiday flag.

-  $S$  seasonal. AR past values.  $I$  differencing. MA past errors.  $x$  external

variables.

## Seasonal index

How strong a period is against an average period. 100 = average.

- Index = period average / overall mean \* 100.

- Sales Y1: 20, 30, 25, 45. Y2: 24, 35, 28, 62. Y3: 28, 40, 31, 59.

| Quarter | Average | Index  |
|---------|---------|--------|
| Q1      | 24      | 69.06  |
| Q2      | 35      | 100.72 |
| Q3      | 28      | 80.58  |
| Q4      | 62      | 149.64 |

- Overall mean 34.75. Deseasonalise Q4 of Y3:  $59 / 1.4964 = 39.43$ .

## Lab code (store sales, statsmodels)

Load, add up per month

url =

`("https://raw.githubusercontent.com/trdsaputr/sample/refs/heads/main/store_data")`

`data_df = pd.read_csv(url)`

`sales_df = data_df[['Order Date', 'Sales']].copy()`

`sales_df['Order Date'] = pd.to_datetime(sales_df['Order Date'])`

`daily_sales = sales_df.groupby('Order Date')['Sales'].sum().sort_index()`

`df1 = sales_df.set_index('Order Date')`

`monthly_sales = df1.resample('ME').sum()`

`monthly_sales['Sales'].plot(title="Monthly Sales")`

- 9994 order rows become 48 monthly points. 'ME' = month end. Daily is too jumpy.

Stationarity, ACF, PACF

`from statsmodels.tsa.stattools import adfuller`

`from statsmodels.graphics.tsaplots import plot_acf, plot_pacf`

`p_value = adfuller(monthly_sales['Sales'], autolag='AIC')[1]`

`plot_acf(monthly_sales)`

`plot_pacf(monthly_sales)`

- adfuller returns a tuple, index 1 is the p-value. Lab: 0.0002, stationary. Lag 12 stands out in the ACF.

ARIMA, lowest AIC

`import itertools`

`from statsmodels.tsa.arima.model import ARIMA`

`from sklearn.metrics import mean_absolute_error as mae`

```

sales = monthly_sales['Sales']
train, test = sales.iloc[:-12], sales.iloc[-12:]
best_aic, best_order = np.inf, None
for order in itertools.product(range(4), range(3), range(4)):

```

```

 try:

```

```

 aic = ARIMA(train, order=order).fit().aic

```

```

 except Exception:

```

```

 continue

```

```

 if aic < best_aic:

```

```

 best_aic, best_order = aic, order

```

```

 fc = ARIMA(train, order=best_order).fit().forecast(steps=12)

```

```

 print(mae(test, fc))

```

- Holds out the last 12 months. Tries all 48 combinations of p, d, q.
- Lab: order (0,2,2), AIC 791.77, MAE 36598.

## SARIMA

```

from statsmodels.tsa.state_space.sarimax import SARIMAX

```

```

p, d, q = 1, 1, 1

```

```

P, D, Q, s = 1, 1, 1, 12

```

```

model = SARIMAX(train, order=(p, d, q), seasonal_order=(P, D, Q, s))

```

```

fc = model.fit().forecast(steps=12)

```

- Lab: MAE 17017. Much better than ARIMA, because it knows the 12-month season.

## SARIMAX with a holiday flag

```

months = sales.index.month

```

```

holiday = pd.DataFrame({'holiday': np.isin(months, [11, 12]) * 1, index=sales.index})

```

```

base = dict(order=(1, 0, 0), trend='c', seasonal_order=(1, 0, 0, 12))

```

```

plain = SARIMAX(train, **base).fit(dispatch=False)

```

```

with_x = SARIMAX(train, exog=holiday.loc[train.index], **base).fit(dispatch=False)

```

```

f1 = plain.forecast(steps=12)

```

```

f2 = with_x.forecast(steps=12, exog=holiday.loc[test.index])

```

- The flag is 1 in November and December. The forecast needs exog for the new months too.
- Lab: MAE 18221 without the flag, 17408 with it.

## Seasonal index

```

from statsmodels.tsa.seasonal import seasonal_decompose

```

```

dec = seasonal_decompose(sales, model='multiplicative', period=12)

```

```

index = dec.seasonal.groupby(dec.seasonal.index.month).first() * 100

```

- Lab: Nov 174 and Dec 176 are the strongest months. Feb 39 is the weakest.

- Repeating last year's month scores MAE 15468, which beats SARIMA here.
- Always compare with that simple baseline.
- Split by time. Score on months the model never saw.