Chapter 1

Introduction

1.1 Machine Learning Tasks

A partial list of common tasks:

- Regression
  - Predict real-valued quantity
- Classification
  - Discrete-valued quantity
  - Dog/Cat
  - Spam/Ham
  - Disease Recognition
  - LLMs predicting the next word
  - Facial detection
  - Named Entity Recognition (NER)
  - Recommendation Systems (Rec-Sys)
- Dimensionality reduction
- Density estimation (Stat. word for generative models)
- Graph Learning
- Routing Problem (Operation Research)

1.2 (Row) Vectors

$$\vec{v} = [v_1 \ v_2 \ \dots \ v_d] \in \mathbb{R}^d$$

Can represent

- tabular data
- image
- word embedding
- model parameters

1.3 Matrices

$$\mathbb{A} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

Can be interpreted as

- A stack of row vectors
- a linear transformation

```
A.shape = (m, n)
```

1.4 Matrix Multiplication

$$\vec{A}\vec{B} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1p} \\ B_{21} & B_{22} & \dots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{n1} & B_{n2} & \dots & B_{np} \end{bmatrix} =: \vec{C}$$

```
A = np.random.randn(3, 2)
B = np.random.randn(2, 5)
assert (A @ B).shape == (3, 5)
```

1.5 Inverse

$\vec{M}^{-1}$ is the matrix inverse of $\vec{M}$.

$$\vec{M}^{-1}\vec{M} = \vec{I}$$
$$\vec{M}\vec{M}^{-1} = \vec{I}$$

```
M = np.random.randn(3, 3)
M_inv = np.linalg.inv(M)
M @ M_inv
```

1.6 Application: Curve Fitting

```
m = 6
x = np.random.randn(m)
noise = 0.25 * np.random.randn(m)
y = 1.2 * np.sin(2*x) + 0.5 + noise
```

Find a polynomial of degree $m - 1 = 5$ that goes through these points.

1.7 Polynomial of degree 5

We don't know the real world model, we can only guess. Let's use degree 5 polynomial.

$$y = b + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5$$

$$= \begin{bmatrix} 1 & x & x^2 & x^3 & x^4 & x^5 \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix}$$

$$\vec{y} = \tilde{\mathbf{X}}\vec{w}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^5 \\ 1 & x_2 & x_2^2 & \dots & x_2^5 \\ \vdots & \vdots & \vdots & \ddots & \vdots \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_5 \end{bmatrix}$$

Why 5? It's a modeling assumption that the practitioner makes. We have 6 data points and 6 parameters.

Constant + degree 1 + ... + degree 5 coeff = 6 parameters.

unknowns = # equalities

"In general", we expect a unique solution.

```
m = 6
X_tilde = np.vstack([ x**p for p in range(m) ]).T

# how to do this with `np.hstack'
np.hstack([ x**p for p in range(m) ]).reshape(1,6)
```

$$\vec{y} = \vec{\tilde{X}}\vec{w}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^5 \\ 1 & x_2 & x_2^2 & \dots & x_2^5 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^5 \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_5 \end{bmatrix}$$

$$\vec{\tilde{X}}^{-1}\vec{y} = \vec{\tilde{X}}^{-1}\vec{\tilde{X}}\vec{w}$$

$$\vec{\tilde{X}}^{-1}\vec{y} = \vec{w}$$

```
# X_tilde @ w = y
X_tilde_inv = np.linalg.inv(X_tilde)
w = X_tilde_inv @ y # RHS only has known quantities.

x_grid = np.linspace(np.min(x), np.max(x))
X_tilde_grid = np.vstack([ x_grid ** p for p in range(m) ]).T
y_grid = X_tilde_grid @ w
plt.plt(x_grid, y_grid)

assert (np.sum(np.square(y - X_tilde @ w)) < 1e-16) # Perfect fit
```

- We fitted a degree 5 polynomial to 6 points.
- What if we want to fit a degree 1 polynomial ($y = b + wx$) to 6 points?

$$\vec{y} = \vec{\tilde{X}}\vec{w} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix} \begin{bmatrix} b \\ w \end{bmatrix}$$

```
X_tilde = np.vstack([ x**p for p in range(2) ]).T
# X_tilde @ w = y ?
X_tilde_inv = np.linalg.inv(X_tilde)
w = X_tilde_inv @ y
# ...
# LinAlgError: Last 2 dimensions of the array must be square

X_tilde = np.vstack([ x**p for p in range(2) ]).T
# X_tilde @ w = y ?
X_tilde_inv = np.linalg.pinv(X_tilde) # <- replace `inv' with `pinv'
w = X_tilde_inv @ y
# ...
# pinv == pseudo inverse, more on this later
```

1.8 Fitted Line

```
x_grid = np.linspace(np.min(x), np.max(x))
X_tilde_grid = np.vstack([ x_grid**p for p in range(2) ]).T
y_grid = X_tilde_grid @ w
plt.plot(x_grid, y_grid)

# np.sum( np.square(y - X_tilde @ w) ) == 1.80 (approximately)
```

1.9 Exercise 1: Training Error Versus Degree

```
X_tilde = np.vstack([ x**p for p in range(maxdeg+1) ]).T

# lines skipped

training_error = np.sum( np.square(y - X_tilde @ w) )
```

1.10 Generalization: Performance on "fresh" data

```
m_test = 25
x_test = np.random.randn(m_test)
# filter to within same range as original training data [...]

noise = 0.25 * np.random.randn(len(x_test))
y_test = 1.2 * np.sin(2 * x_test) + 0.5 + noise
```

1.11 Linear Regression (1-dimensional samples)

Let $\vec{\theta} = \begin{bmatrix} b \\ w \end{bmatrix}$

$$\min_{w \in \mathbb{R}, b \in \mathbb{R}} J(\vec{\theta}) := \frac{1}{m} \sum_{i=1}^{m} \left(y_i - f\left(x_i; \vec{\theta} \right) \right)^2$$

$J\left(\vec{\theta}\right)$ is the risk .... Risk is the average loss.

1.12 The derivative test for local minimality

Let $J : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function.
Let $\vec{\theta} = [\theta_1, \ldots, \theta_d]^T$. The gradient of J with respect to...

- If $\vec{v} \in \mathbb{R}^d$ is a vector, then
$$\nabla_{\vec{\theta}} \left(\vec{\theta}^T \vec{v} \right) = \vec{v}$$
- If $\mathbf{A}$ is a matrix, then
$$\nabla_{\vec{\theta}} \left(\vec{\theta}^T \mathbf{A} \vec{\theta} \right) = 2\mathbf{A}\vec{\theta}$$

1.13 Gradient Descent (GD)

Let $\epsilon_k > 0$ be learning rates, $k = 1, 2, \ldots$

- Initialize $\vec{\theta}$
- While not converged (k = iteration counter):
  - Compute gradient: $\vec{g} \leftarrow \nabla_{\vec{\theta}} J\left(\vec{\theta}\right)$
  - Compute update: $\vec{\theta} \leftarrow \vec{\theta} - \epsilon_k \vec{g}$