ESE 4170 Notes

Michael Lee

October 2, 2025

Contents

1 Syllabus / Overview

- Submit work as a pdf with Python code
- 20 min quiz after each module (in-class, open book, open note)

2 Review

What is AI?

- **Def.** Artificial Intelligence (AI) - A computer system that can perform complex tasks normally done by human, reasoning, decision making, creating, etc.
- **Def.** General AI or Artificial General Intelligence (AGI) - AI that can learn and perform tasks anywhere
- **Def.** Narow AI - AI that can perform specific tasks (playing chess, auto driving, reading text, etc.)
- Latest AI wave has been focused on AGI (ChatGPT, Claude, etc.)
- **Def.** Turing Test
  1. A computer and a human both output text
  2. A human judge tries to figure out which text output is human and which is a computer
  3. If the judge cannot figure out who is who, the computer passes the Turing Test
- How do LLMs work - They predict the most probable next word given the previous words
- **Def.** Generative Pre-trained Transformer (GPT) - Generative (can produce novel content) Pre-trained (trained on a pre-existing dataset), Transformer (the kind of neural network it uses)

How does AI work (high-level)

- In TRADITIONAL PROGRAMMING, we use an input (data) and rules to get an output

$$\begin{pmatrix}\text{Input}\\ \text{Rules}\end{pmatrix} \quad \rightarrow \quad \text{Output}$$

- In MACHINE LEARNING, we use an input (data) and the output to obtain the rules

$$\begin{pmatrix}\text{Input}\\ \text{Output}\end{pmatrix} \quad \rightarrow \quad \text{Rules}$$

Rectangle Game

- Define a game where $R = (a, b) \times (c, d) \subseteq \mathbb{R}^2$.
- We want to figure out what defines R.
- We will accomplish this by plotting N points, $p_1, \ldots, p_N \in \mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$.
- For each p_i with $i \in [N]$, we know $p_i \in R$ or $p_i \notin R$ with some fixed probability distribution $\mathbb{D}$.
- Based on this information, we will create a space R' s.t. $p_i \in R' \iff p_i \in R$.

How does machine learning work, generally?

- Let S be the input vector

$$S = (x_1, x_2, \ldots, x_n) \subseteq \mathcal{X}^n$$

- Let $\mathcal{R}$ be the target value
- The goal of ML is to find a map $f : S \rightarrow \mathcal{R}$
- **Def.** Hypothesis model set - The set of functions we will use to map $S \rightarrow \mathcal{R}$. Specifically, we typically employ a parametric model:

$$h_w(x) = h(x, w)$$

- Where $w = [w_1, \ldots w_d]^T \in \mathcal{R}^d$

- **Def.** Hypothesis class - Set of all functions induced by the possible choice of the parameter, w

$$H = \{h_w | w \in \mathcal{R}^d\}$$

- **Def.** Loss (cost) function (L) - The function we use ot measure how different our model's predicitions, $\hat{y}$, are from the true outputs, y.
- We will attempt to minimize L using w

$$w^* = \arg\min_w L(w)$$

Math Review

$$\mathbb{R} = \mathcal{R}, \quad \mathbb{R}^n = \mathcal{R}^n$$

Let the following be true:

$$\mathbf{x},\ \mathbf{y} \in \mathcal{R}^n, \quad \mathbf{A} \in \mathcal{R}^{m \times n}, \quad a \in \mathcal{R}$$

Recall the **Inner Product**

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y}$$

The see the slides lol

- Quiz will be on **Gradient Descent Search Method**

Given an objective function, $g(w)$, we want to find $w \in \mathcal{R}^d$ that minimizes $g(w)$.

$$\arg\min_w g(w)$$

Steps (conceptually):

1. Choose a point $w^0 \in \mathcal{R}^d$ and find $g(w^0)$
2. Gradually move towards the minimum by moving in the direction of the negative gradient using $w^1, w^2, \dots w^k$
3. Hence after $k+1$ steps, we will find w that minimizes $g(w)$

Looking into the maths for 1-dimensional functions,

$$g(w + \Delta w) = \sum_{i=0}^{n} \frac{g^i(w) \Delta w^i}{i!} \approx g(w) + g'(w) \Delta w$$

Let $\Delta w = -\eta g'(w)$ where $\eta > 0$ is the learning rate. Hence,

$$g(w - \eta g'(w)) \approx g(w) + g'(w)(-\eta g'(w)) = g(w) - \eta \left(g'(w)\right)^2$$
$$\forall g'(w) \neq 0, \quad g(w - \eta g'(w)) < g(w)$$

Thus, we can find

$$w^{i+1} = w^i - \eta g'(w^i) \quad (i \in [k])$$

Looking into the maths for n-dimensional functions,

$$g(w + \Delta w) \approx g(w) + \nabla g(w) \Delta w \quad \text{where} \quad \nabla g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \vdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

η Step Size Tradeoff:

- The larger η is, the more likely we are to skip the minimum
- The smaller η is, the more likely we are to take a long time to find the minimum
- Not that η does not need to be fixed

Practice Problems (Find w that minimizes g for all functions. Answers are in the slides):

1. $g(w) = w^2 + 2w + 2$
2. $g(w) = 0.5w_1^2 + w_2^2, \quad w = [w_1, w_2]^T$

3 Regression

- Our goal is to create,

$$f : \mathcal{R}^d \to \mathcal{R}, \quad \hat{y} = w_0 + \sum_{i=1}^{d} w_i x_i$$

- To rewrite this more simply, let

$$\vec{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} \quad \text{and} \quad \vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$$

- We will create an approximation of f, h_w

$$h_w(\vec{x}) = \vec{x}^T \vec{w} = \langle \vec{x}, \vec{w} \rangle$$

- We want to find h_w that best fits our data, h_{w^*}, and thus best approximates $y = f(x)$
- We can find h_{w^*} using **Ordinary Least Squares (OLS)**, which minimizes the sum of the squared errors (SSE)
- Let our **Loss Weight Function**, $L(w)$, be:

$$L(w) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 = \sum_{i=1}^{N} (h_w(\vec{x}) - y_i)^2 = \sum_{i=1}^{N} \left(\vec{x}^T \vec{w} - y_i\right)^2$$

- Let our dataset be X

$$\begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \vdots \\ \hat{y_N} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1d} \\ 1 & x_{21} & \dots & x_{2d} \\ \vdots & \vdots & \dots & \vdots \\ 1 & x_{N1} & \dots & x_{Nd} \end{bmatrix} \begin{bmatrix} \hat{w_1} \\ \hat{w_2} \\ \vdots \\ \hat{w_d} \end{bmatrix} = X\vec{w}$$

- Hence,

$$L(w) = \sum_{i=1}^{N} \left(\vec{x}^T \vec{w} - y_i\right)^2 = ||X\vec{w} - y||_2^2$$

- We want to find ...

$$w^* = \arg\min_w \{L(w) = ||X\vec{w} - y||_2^2\}$$

Proof. **Claim:** $w^* = (X^T X)^{-1} X^T y$

Expanding, $L(w) = ||X\vec{w} - y||_2^2$

$$L(w) = ||X\vec{w} - y||_2^2 = (X\vec{w} - y)^T (X\vec{w} - y) = (X\vec{w})^T X\vec{w} - 2(X\vec{w})^T y + y^T y$$

Thus,

$$\nabla_w L(w) = 2X^T X\vec{w} - 2X^T y = \vec{0} \quad \Rightarrow \quad 2X^T X\vec{w} = 2X^T y \quad \Rightarrow \quad X^T X\vec{w} = X^T y$$

If $X^T X$ is full rank, $w^* = (X^T X)^{-1} X^T y$. We must now show this is a *global minimum* by showing the Hessian of L is positive-definite.

$$\nabla^2 L(w) = 2X^T X$$

$$\forall w, \quad w^T (2X^T X) w = 2(Xw)^T Wx = 2||Xw||_2^2 > 0$$

□

- We can also use non-linear functions (i.e. non-linear feature mapping)

- **Def.** Feature map: $\phi : \mathcal{R}^l \to \mathcal{R}^{d+1}$, $\vec{x_i} \mapsto \phi(\vec{x_i})$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1l} \\ x_{21} & x_{22} & \dots & x_{2l} \\ \vdots & \vdots & \dots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nl} \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} \phi(x_1) \\ \phi(x_2) \\ \vdots \\ \phi(x_N) \end{bmatrix} = \begin{bmatrix} \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_d(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \dots & \phi_d(x_2) \\ \vdots & \vdots & \dots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \dots & \phi_d(x_N) \end{bmatrix} = \Phi$$

- Thus, our hypothesis function is:

$$h_w(\vec{x}) = \sum_{j=0}^{d} w_j \phi_j(\vec{x}) = \phi(\vec{x})^T w$$

- Similarly, to before, we can optimize the weights, $\vec{w}$, on $||\Phi\vec{w} - y||^2$, giving us w^*:

$$w^* = (\Phi^T\Phi)^{-1}\Phi^T y$$

- We can use polynomials to match nearly any function, so we can use polynomials for linear regression:

$$\phi(x)^T = \begin{bmatrix} 1 & x & x^2 \cdots x^m \end{bmatrix}^T$$

- Hence,

$$\hat{y} = \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \vdots \\ \hat{y_N} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & (x_1)^2 & \cdots & (x_1)^m \\ 1 & x_2 & (x_2)^2 & \cdots & (x_2)^m \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_N & (x_N)^2 & \cdots & (x_N)^m \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = X\vec{w}$$

- Be wary of overfitting: too many variables causes you to accurately predict data on the training set, but fail massively on test or real datasets
- One way to correct for overfitting is **Ridge Regularization** or **L2 Regularization**:

$$L(w) = \arg\min_w \left\{ (||Xw - y||_2)^2 + \lambda(||w||_2)^2 \right\} \quad \text{where } \lambda > 0$$

Proof. **Claim:** $w_{ridge} = (X^TX + \lambda I)^{-1}X^Ty$

$$L(w) = (||Xw - y||_2)^2 + \lambda(||w||_2)^2 = w^TX^TXw - 2w^TX^Ty + y^Ty + \lambda w^Tw$$

Thus,

$$\nabla_w L(w) = 0 = 2X^TXw - 2X^Ty + 2\lambda w = X^TXw - X^Ty + \lambda w \quad \Rightarrow \quad X^Ty = (X^TX + \lambda I)w$$

Finally,

$$w_{ridge} = (X^TX + \lambda I)^-1X^Ty$$

We know this minimum is unique

$$\nabla^2 L(w) = 2X^Tx + 2\lambda I$$

Thus,

$$\forall w \neq \vec{0},\ w^T(2X^TX + 2\lambda I)w = w^T(2X^TX)w + w^T\lambda Iw = ||Xw||^2 + \lambda||w||^2 > 0$$

And so $\nabla^2 L(w)$ is positive definite. □

- We can also use **Lasso Regularization** or **L1 Regularization**.

$$w_{lasso} = \arg\min_w \left\{ (||Xw - y||_2)^2 + \lambda \sum_{j=0}^{d} |w_j| \right\}$$

- **Maximium Likelihood Estimate (MLE)**

– Let $X \sim N(\mu, 1)$. To estimate μ from a sample size, n:

$$L(\mu) = P\left(\bigcap_{i=1}^{n} x_i\right) = \prod_{i=1}^{n} p(x_i \,|\, \mu) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2}}$$

$$l(\mu) = \ln(L(\mu)) = \sum_{i=1}^{n} \ln\left(\frac{1}{\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2}}\right)$$

$$\frac{dl}{d\mu} = 0 = \sum_{i=1}^{n} (x_i - \mu) \quad \Rightarrow \quad \hat{\mu} = ???$$

– In a ML context, we have $Y_i = h_w(x_i) + Z_i$ where x_i is fixed and Z_i is a random variable.

– In most contexts, we assume $Z_i \sim N(0, \sigma^2)$ are identically distributed zero-mean Gaussians. Hence,

$$Y_i \sim N(h_w(x_i), \sigma^2)$$

– And so,

$$\hat{w_{MLE}} = \arg\min_w L(w, \mathcal{D}) \quad \text{where:}$$

$$L(w, \mathcal{D}) = p(Y_1 = y_1, \ldots, Y_N = y_n \,|\, x_1, \ldots, x_n, w) = \prod_{i=1}^{n} p(y_i \,|\, x_i, w)$$

$$l(w; X, y) = \sum_{i=1}^{n} ln\big(p(y_i \,|\, x_i, w)\big)$$

- Recall that:

$$p(Y_i = y_i | x_i, w) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\left(y_i - h_w(x_i)\right)^2}{2}}$$

- Hence,

$$l(w; X, y) = -\sum_{i=1}^{n} \frac{\left(y_i - h_w(x_i)\right)^2}{2\sigma^2} - N \cdot \ln(\sigma\sqrt{2\pi})$$

- Maximizing for w,

$$\hat{w_{MLE}} = \arg\min_w \left\{ -\sum_{i=1}^{n} \frac{\left(y_i - h_w(x_i)\right)^2}{2\sigma^2} - N \cdot \ln(\sigma\sqrt{2\pi}) \right\} = \arg\min_w \left\{ -\sum_{i=1}^{n} \left(y_i - h_w(x_i)\right)^2 \right\}$$

- Recall $h_w(x_i) = (x_i)^T w$. Thus,

$$\hat{w_{MLE}} = \arg\min_w \left\{ -\sum_{i=1}^{n} \left(y_i - (x_i)^T w\right)^2 \right\} = \arg\min_w \left\{ (||y_i - (x_i)^T w||_2)^2 \right\}$$

□

- Recall Bayes Theorem:

$$P(h|E) = \frac{P(E|h) \cdot P(h)}{P(E)}$$

- Let the following be true:

$$h_{MAP} = \arg\max_{h \in H} \{P(h|\mathcal{D})\} = \arg\max_{h \in H} \left\{ \frac{P(\mathcal{D}|h)P(h)}{\mathcal{D}} \right\} = \arg\max_{h \in H} \{P(\mathcal{D}|h)P(h)\}$$

- In our use case, we parameterized h by w. Thus,

$$\hat{w_{MAP}} = \arg\max_w \{P(h_w|\mathcal{D})\} = \arg\max_w \{P(\mathcal{D}|h_w)P(h_w)\} = \arg\max_w \{\ln(P(\mathcal{D}|h_w)) + \ln(P(h_w))\}$$

$$= \arg\min_w \{-\ln(P(\mathcal{D}|h_w)) - \ln(P(h_w))\}$$

- Assuming every hypothesis in H is equally likely, $P(h_w)$ is constant and so:

$$h_{MAP} = \arg\max_{h \in H} P(\mathcal{D}|h) = h_{ML}$$

- Thus, we can look to minimize w:

$$\ldots$$

4 Model Performance and Selection

- Model selection: How do we pick the "best" model?
- Performance evalution: Can we use training error (SSE) to measure the performance of fitted curves
- **Def.** True Error: expected loss over the entire dataset. However, this is impossible to observe

$$R(w) = E_{x,y}[l(h_w(x), y)]$$

- **Def.** Training Error: Expected loss over the training set
- How to get training error:
  1. Separate data in **Test Set**, **Training Set**, and **Validation Set (optional)**
  2. Train model on **Training Set**
  3. Tune hyperparameters on **Validation Set**
  4. Find training error by inputting **Test Set** and comparing estimated outputs to true outputs
- **Def.** k-fold Cross Validation
  1. Split data into Test & Training sets.
  2. Split training data in k equally sized groups.
  3. Train the model on all data except for one block k-times. Using the left out block, compute validation error. Do NOT repeat left out blocks.
  4. Average the k validation errors. This is the true error
  5. Find Training error using Test set.
- Tuning hyperparameters
  - **Def.** Grid Search: Build a model for each possible combination of all hyperparameters provided. Evaluate each model and select the model that performs the best.
    - Very expensive computationally.
  - **Def.** Random Search: Try random combinations of hyperparameters
    - Reaches a very good hyperparameter combination quickly, but it does not always find the best methods
    - Can find combinations of parameters with 5% optima in only 60 iterations with probability 95%
- Goodness of Fit
  - Mean Squared Error: Average squared error

$$MSE = \frac{1}{n} \cdot \sum_i (y_i - \hat{y}_i)^2$$

  - Mean Absolute Error: Average absolute error

$$MAE = \frac{1}{n} \cdot \sum_i |y_i - \hat{y}_i|$$

  - R^2: What percentage (%) of the data can be explained by your model

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

- Bias-Variance Decomposition of Model Predictor Error

5 Classification

- **Def.** Class: What type something is
- **Def.** Binary Classification: Classifying something into one of two classes ($k = 2$)
- **Def.** Multiclass Classification: Classifying something into one of k classes ($k > 2$)
- Multiclass classification is usually just repeated binary classification
  - **Def.** One-vs-Rest (OVR): Train k binary classifiers, each of which is trained to distinguish between one, unique class and the rest
  - **Def.** One-vs-One (OVO): Train $\frac{k(k-1)}{2}$ binary classifiers, each of which is trained to distinguish between two, unique classes. See if a data point is more likely to be one class or another. Choose the winning class until there are no more classes to compare.
- In most cases, each class is disjoint (i.e. each input is assigned to exactly one class)
- The input space is divided in R_k regions, one for each class (C_k)
- The boundaries between each region are **decision boundaries**
- One way to represent classifiers in using discriminant functions: $g_i(x)$ where $i \in [k]$.
- The classifier assigns a feature vector, x, if a class, C_i if $g_i(x) > g_j(x)$ for all $j \neq i$.
- One popular discriminant function is $g_i(x) = P(C_i|x)$ (maximum posterior probability).

$$g_i(x) = P(C_i|x) = \frac{P(x|C_i)P(C_i)}{\sum_{j=1}^{k} P(x|C_j)P(C_j)} = P(x|C_i)P(C_i) = \ln P(x|C_i) + \ln P(C_i)$$

- For the two-class case, we use two discriminant functions: $g_1(x)$ and $g_2(x)$ and assign x to C_1 if $g_1(x) > g_2(x)$ and C_2 otherwise.
- We can also define a discriminant function:

$$g(x) = g_1(x) - g_2(x) = \ln P(x|C_1) + \ln P(C_1) - \ln P(x|C_2) - \ln P(C_2) = \ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)} = \ln \frac{P(C_1)}{P(C_2)} + \ln \frac{P(x|C_1)}{P(x|C_2)}$$

- **Def.** Confusion Matrix: A table that shows the number of true positives, true negatives, false positives (type I error), and false negatives (type II error)

		Predicted
	Positive	**Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

	Predicted Class				
	C_1	C_2	$\cdots$	C_k	**Total**
Actual C_1	n_{11}	n_{12}	$\cdots$	n_{1k}	$\sum_{j=1}^{k} n_{1j}$
Actual C_2	n_{21}	n_{22}	$\cdots$	n_{2k}	$\sum_{j=1}^{k} n_{2j}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
Actual C_k	n_{k1}	n_{k2}	$\cdots$	n_{kk}	$\sum_{j=1}^{k} n_{kj}$
Total	$\sum_{i=1}^{k} n_{i1}$	$\sum_{i=1}^{k} n_{i2}$	$\cdots$	$\sum_{i=1}^{k} n_{ik}$	$\sum_{i=1}^{k}\sum_{j=1}^{k} n_{ij}$

- Accuracy: What percent of predictions are correct?

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{\sum_{i=1}^{k} n_{ii}}{\sum_{i=1}^{k}\sum_{j=1}^{k} n_{ij}}$$

- Error Rate: What percent of predictions are incorrect?

$$\text{Error Rate} = 1 - \text{Accuracy} = 1 - \frac{TP}{TP + TN + FP + FN} = 1 - \frac{\sum_{i=1}^{k} n_{ii}}{\sum_{i=1}^{k}\sum_{j=1}^{k} n_{ij}}$$

- Precision: For a class, C_i, percent of predicted C_i's are correct?

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{n_{11}}{\sum_{j=1}^{k} n_{ij}}$$

- F1 Score: Harmonic mean of precision and recall

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- As $F_1 \to 0$, either Precision or Recall approaches 0. However, we want both to be high.
- As $F_1 \to 1$, both Precision and Recall approach 1.
- F_β Score: Generalization of F1 Score

$$F_\beta = \frac{(1 + \beta^2) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

- Generative models vs. Discriminative models:
  - **Def.** Generative models: Explicitly use prior probability to compute posterior probability
  - Ex. Naive Bayes, Hidden Markov Models, and Bayesian Networks

$$\hat{y} = \arg\max_i P(C_i|x) = \arg\max_i P(x|C_i)P(C_i)$$