Homework 1

CSE 541: Interactive Learning
Instructor: Kevin Jamieson
Due: 11:59 PM on April 27, 2025

Problem 1 — Gradient Descent and Exponential Weights via Regularization

In this problem, we will explore how gradient descent and the exponential weights algorithm can both be derived as instances of a general framework: minimizing a linearized loss plus a regularization term. Let $\mathcal{K}$ denote a convex decision set (e.g., the probability simplex).
Let $f_1, f_2, \ldots, f_T$ be a sequence of convex loss functions. Define $g_t = \nabla f_t(x_t)$ as a subgradient of f_t at the point x_t.[1]

(a) Let $\mathcal{K} \subseteq \mathbb{R}^d$ be a convex and compact set and define $\Pi_{\mathcal{K}}(y) = \arg\min_{x \in \mathcal{K}} \|y - x\|_2$. Show that the standard online gradient descent (OGD) update:

$$x_{t+1} = \Pi_{\mathcal{K}}\left(x_t - \eta g_t\right)$$

can be written equivalently as:

$$x_{t+1} = \arg\min_{x \in \mathcal{K}} \left\{ \eta g_t^\top x + \frac{1}{2}\|x - x_t\|_2^2 \right\}$$

That is, the OGD update is the solution to minimizing the linearized loss plus an ℓ_2 regularization centered at x_t.

Hint: Complete the square or derive the optimality condition.

(b) Now suppose the domain $\mathcal{K}$ is the probability simplex:

$$\Delta^d = \left\{ x \in \mathbb{R}^d : x_i \geq 0, \sum_{i=1}^d x_i = 1 \right\}$$

Instead of ℓ_2 regularization, consider the *KL divergence* regularizer:

$$D_{\mathrm{KL}}(x \| x_t) = \sum_{i=1}^d x_i \log \frac{x_i}{x_{t,i}}$$

Derive the update:

$$x_{t+1} = \arg\min_{x \in \Delta^d} \left\{ \eta g_t^\top x + D_{\mathrm{KL}}(x \| x_t) \right\}$$

and show that it corresponds to the exponential weights update:

$$x_{t+1,i} \propto x_{t,i} \exp(-\eta g_{t,i}) \quad \text{for } i = 1, \ldots, d$$

Hint: Use Lagrange multipliers to enforce the simplex constraint.

[1] If you're not familiar with the concept of subgradient, you may simply assume f_t's are differentiable and g_t's are their gradients.

Problem 2 — The Doubling Trick for Anytime Exponential Weights

The exponential weights algorithm (also known as Hedge) for the expert setting typically requires knowledge of the total time horizon T in order to set the learning rate:

$$\eta = \sqrt{\frac{8 \log d}{T}}$$

to achieve the standard regret bound:

$$\text{Regret}_T \leq \sqrt{T \log(d)/2}$$

where d is the number of experts. But in practice, we often do not know T in advance. One approach to overcome this limitation is the *doubling trick*, which allows us to construct an **anytime** version of the algorithm.
Suppose we divide time into epochs of exponentially increasing length: epoch 1 lasts for 1 round, epoch 2 lasts for 2 rounds, epoch 3 lasts for 4 rounds, epoch 4 for 8 rounds, and so on. That is, epoch m lasts for 2^{m-1} rounds. Let $T_m = 2^{m-1}$ be the length of epoch m, and define the learning rate in epoch m as

$$\eta_m = \sqrt{\frac{8 \log d}{T_m}}.$$

(a) How many total epochs M will be run before reaching a time horizon of T? Express M in terms of T.

(b) For each epoch m, write the regret bound for that epoch using the Hedge algorithm with learning rate η_m.

(c) Sum the regret over all epochs to obtain a bound on the total regret up to time T. Show that the total regret satisfies:

$$\text{Regret}_T \leq C\sqrt{T \log d}$$

for some small constant C (specify the value you obtain).

Problem 3 — Exponential Weights on Real Stock Data

In this problem, you will use the `yfinance` Python package to download real stock market data and apply the exponential weights algorithm (Hedge) in two settings:

1. A standard setting where each asset is treated as an expert (Part (a)),
2. A richer setting where each expert is a portfolio sampled from the simplex (Part (c)).

As a first step, use the `yfinance` package and the following Python code to download daily adjusted close prices for the seven given stocks over a one-year period. Then, the variable **`returns`** gives the multiplicative daily returns, which is defined as

$$r_{t,i} = \frac{P_{t,i}}{P_{t-1,i}},$$

where $P_{t,i}$ is the price of stock i on day t.

```
import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)  # For reproducibility

tickers = ['AAPL', 'MSFT', 'GOOG', 'AMZN','META','BIL','BND']
data = yf.download(tickers, start ="2022-01-01", end ="2025-04-11", actions=True)
```

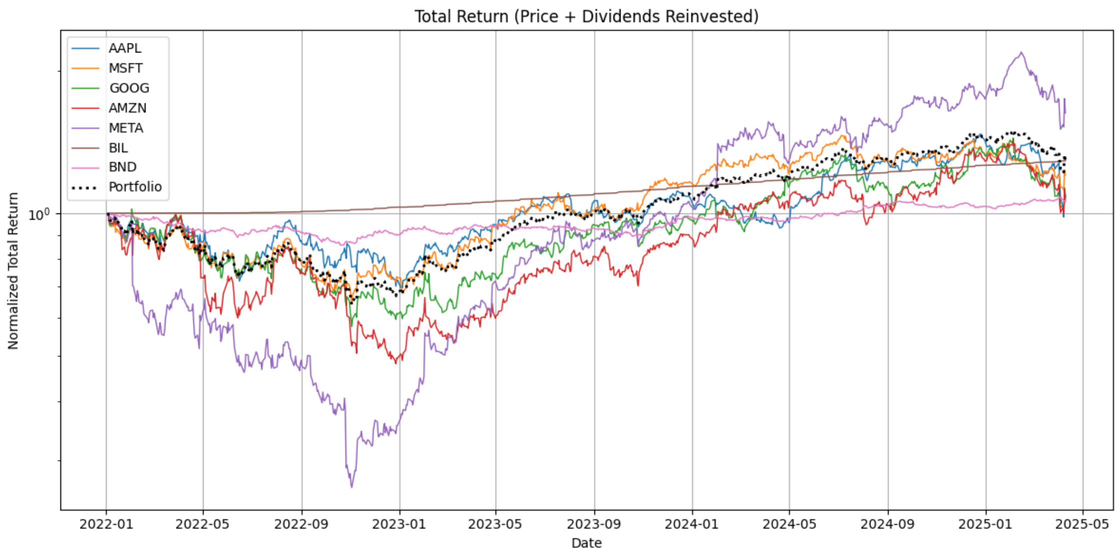

Figure 1: Total return of individual stocks and equally weighted daily rebalanced portfolio (log scale).

```
close = data['Close']
dividends = data['Dividends']
individual_total_return = pd.DataFrame(index=close.index, columns=close.columns)
individual_total_return.iloc[0] = 1  # Normalize all series to 1

p = np.ones(data['Close'].shape[1])/len(data['Close'].columns)  # Equal weights for each
    stock
portfolio_total_return = pd.DataFrame(index=close.index, columns=['Portfolio'])
portfolio_total_return.iloc[0] = 1.

for t in range(1, len(close)):
    r_t = (close.iloc[t] + dividends.iloc[t]) / close.iloc[t - 1]
    individual_total_return.iloc[t] = individual_total_return.iloc[t - 1] * r_t
    portfolio_total_return.iloc[t] = portfolio_total_return.iloc[t-1] * np.dot(p, r_t)

plt.figure(figsize=(12, 6))
for ticker in tickers:
    plt.plot(individual_total_return.index, individual_total_return[ticker], label=ticker,
    linewidth=1)
plt.plot(portfolio_total_return.index, portfolio_total_return['Portfolio'], label='Portfolio
    ', linewidth=2, color='black',linestyle=':')
plt.title("Total Return (Price + Dividends Reinvested)")
plt.xlabel("Date")
plt.ylabel("Normalized Total Return")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.yscale('log')
plt.show()
```

(a) Let each of the d assets be an expert. Implement the Hedge algorithm using

$$x_{t+1,i} \propto x_{t,i} \cdot \exp\left(\eta \cdot \log r_{t,i}\right)$$

with different values of $\eta \in \{0.1, 0.5, 1, 2, 5\}$.

Your cumulative wealth and the wealth of the best fixed asset in hindsight are respectively defined as

$$W_T = \prod_{t=1}^{T} x_t^\top r_t \quad \text{and} \quad W_T^* = \max_{i \in [d]} \prod_{t=1}^{T} r_{t,i}.$$

Then, for each η, report the regret: $\log W_T^* - \log W_T$.

(b) As a reflection, How does η impact performance?

(c) Instead of treating individual assets as experts, suppose each expert is a fixed portfolio over the assets (i.e., a point in the simplex). Since there are infinitely many such portfolios, we can randomly sample N of them uniformly from the simplex. Now, your task is to implement the following procedures:

- For various values of $N \in \{10, 50, 200, 1000\}$, generate N portfolios $\{v^{(1)}, \dots, v^{(N)}\} \in \Delta^d$ by sampling uniformly from the simplex.
- Treat each sampled portfolio as an "expert." On day t, observe the return vector r_t, and for each expert $v^{(j)}$, compute the expert's return: $v^{(j)\top} r_t$.
- Try multiple values of $\eta \in \{1, 10, 50, 100, 200\}$ and run exponential weights:

  $$w_{t+1,j} \propto w_{t,j} \cdot \exp\left(\eta \cdot \log(v^{(j)\top} r_t)\right)$$

  Normalize the weights and play the aggregate portfolio:

  $$x_t = \sum_{j=1}^{N} w_{t,j} \cdot v^{(j)}$$

- For each configuration of η and N, track and store the cumulative wealth over time:

  $$W_t = \prod_{s=1}^{t} x_s^\top r_s$$

Then, report the following in your submission:

(i) For each η and N, report the regret relative to the best sampled portfolio:

$$\text{Regret} = \log\left(\max_{j \in [N]} \prod_{t=1}^{T} v^{(j)\top} r_t\right) - \log W_T$$

(ii) On the same axes, plot the following curves over time:

- Your algorithm's wealth (with your choice of η and N): W_t.
- The wealth of each individual asset: $W_t^{(i)} = \prod_{s=1}^{t} r_{s,i}$ for all $i \in [d]$.[2]
- The wealth of the best sampled portfolio: $W_t^{\text{best}} = \max_{j \in [N]} \prod_{s=1}^{t} v^{(j)\top} r_s$.
- Uniform allocation wealth: $W_t^{\text{uniform}} = \prod_{s=1}^{t} \left(\frac{1}{d} \sum_{i=1}^{d} r_{s,i}\right)$.

You need to produce **two** plots for this part. The one is in linear scale, in which the raw wealth is plotted, and the other is in log scale, in which the log wealth is plotted.

(d) Analyze the above results by answering the following questions:

(i) How does increasing N affect your regret and wealth?

(ii) How does your final wealth in this setting compare to the best individual asset in hindsight?

(iii) Discuss the trade-offs between computational cost (large N) and expressivity (more diverse portfolios).

[2]Consider using arguments `linestyle='--'` and `linewidth=0.5` for these curves to make plots look cleaner.

Problem 4 — The Upper Confidence Bound Algorithm

Consider the following algorithm for the multi-armed bandit problem.

```
Algorithm 1: Upper Confidence Bound (UCB)

    Input: Time horizon $T$, 1-subGaussian arm distributions $P_1, \cdots, P_n$ with unknown
    means $\mu_1, \cdots, \mu_n$ such that $\mathbb{E}_{X \sim P_i}[X] = \mu_i$
    Initialize: Let $T_i(t)$ denote the number of times arm $i$ has been pulled up to
    (inclusive) time $t$ and let $T_i = T_i(T)$. Pull each arm once.
    for $t = n+1, \cdots, T$ do
        Pull arm $I_t = \text{argmax}_{i=1,\cdots,n} \widehat{\mu}_{i,T_{i(t-1)}} + \sqrt{\frac{2\log(2nT^2)}{T_i(t-1)}}$ and observe draw from $P_i$
        Let $\widehat{\mu}_{i,T_i(t)}$ be the empirical mean of the first $T_i(t)$ pulls.
```

In the following exercises, we will compute the regret of the UCB algorithm and show it matches the regret bound from lecture. Without loss of generality, assume that the best arm is μ_1. For any $i \in [n]$, define the *sub-optimality gap* $\Delta_i = \mu_1 - \mu_i$. Define the regret at time T as $R_T = \mathbb{E}[\sum_{t=1}^{T} \mu^* - \mu_{I_t}] = \sum_{i=1}^{n} \Delta_i \mathbb{E}[T_i]$.

(a) Consider the event

$$\mathcal{E} = \bigcap_{i \in [n]} \bigcap_{s \leq T} \left\{ |\widehat{\mu}_{i,s} - \mu_i| \leq \sqrt{\frac{2\log(2nT^2)}{s}} \right\}.$$

Show that $\mathbb{P}(\mathcal{E}) \geq 1 - \frac{1}{T}$.

(b) On event $\mathcal{E}$, show that $T_i \leq 1 + \frac{8\log(2nT^2)}{\Delta_i^2}$ for $i \neq 1$.

(c) Show that $\mathbb{E}[T_i] \leq \frac{8\log(2nT^2)}{\Delta_i^2} + 2$. When $n \leq T$, conclude by showing that $R_T \leq \sum_{i=2}^{n} \left(\frac{24\log(2T)}{\Delta_i} + 2\Delta_i \right)$.

Problem 5 — Empirical Experiments of UCB, TS and ETC

Implement UCB, Thompson Sampling (TS), and Explore-then-Commit (ETC). The TS algorithm and ETC algorithm are given below.

```
Algorithm 2: Thompson Sampling (TS)

    Input: Time horizon $T$
    Assume the prior distribution $p_0$ over $\mathbb{R}^n$ is known and that $\theta^* \sim p_0$ (so that
    $\theta^* \in \mathbb{R}^n$). Assume each arm shares the same conditional likelihood function such
    that an observation $X$ from arm $i$ follows $X \sim f(\cdot|\theta_i^*)$ (e.g., $X \sim \mathcal{N}(\theta_i^*, 1)$). Let
    $p_t(\theta|I_1, X_{I_1,1}, \cdots, I_t, X_{I_t,t}) \propto \prod_{s=1}^{t} f(X_{I_s,s}|\theta_{I_s})p_0(\theta)$ be the posterior distribution on
    $\theta^*$ at time $t$.
    for $t = 1, \cdots, T$ do    Sample $\theta^{(t)} \sim p_{t-1}$ (Note: $\theta^{(t)} \in \mathbb{R}^n$)
        Pull arm $I_t = \text{argmax}_{i \leq n} \theta_i^{(t)}$ to observe $X_{I_t,t}$
        Compute exact posterior update $p_t$
```

```
Algorithm 3: Explore-then-Commit (ETC)

    Input: Time horizon $T$, $m \in \mathbb{N}$, 1-sub-Gaussian arm distributions $P_1, \cdots, P_n$ with
    unknown means $\mu_1, \cdots, \mu_n$
    for $t = 1, \cdots, T$ do
      If $t \leq mn$, choose $I_t = (t \mod n) + 1$
      Else, $I_t = \operatorname{argmax}_i \widehat{\mu}_{i,m}$
```

Let $P_i = \mathcal{N}(\mu_i, 1)$ for $i = 1, \ldots, n$. For Thompson sampling, define the prior for the ith arm as $\mathcal{N}(0, 1)$ and the likelihood function as $f(\cdot|\mu_i) = P_i$.

(a) Let $n = 10$ and $\mu_1 = 0.1$ and $\mu_i = 0$ for $i > 1$. On a single plot, for an appropriately large T to see expected effects, plot the regret for the UCB, TS, and ETC for several values of m.

(b) Let $n = 40$ and $\mu_1 = 1$ and $\mu_i = 1 - 1/\sqrt{i-1}$ for $i > 1$. On a single plot, for an appropriately large T to see expected effects, plot the regret for the UCB, TS, and ETC for several values of m.