Data 37711: Machine Learning & AI Part I **Fall 2025**

Homework 1

Student: Chenxi Peng Due date: October 18^{th}

1 Q1

Let d denote Victor's total wealth, $b \in [0, d]$ be the amount bet on Biden winning, and y is the amount of money he has.

Victor's subjective probability: $P(\text{Biden wins}) = 0.85$.

Betting markets give even money bets at $P(\text{Biden wins}) = 0.6$

Then, if Biden wins, Victor's wealth becomes $d - b + 2b = d + b$; if Biden loses, his wealth becomes $d - b$.

1.1 (a)

Linear Utility: $U(y) = y$

The expected utility is

$$\mathbb{E}[U] = 0.85(d + b) + 0.15(d - b) = d + (0.85 - 0.15)b = d + 0.7b.$$

This is strictly increasing in b, so

$$b^* = d.$$

Hence, Victor would bet his entire bankroll on Biden. However, this linear utility corresponds to a risk-neutral agent, which is unrealistic since it ignores risk aversion and the diminishing marginal utility of wealth.

1.2 (b)

Log Utility: $U(y) = \log(y)$

The expected utility is

$$EU(b) = 0.85 \log(d + b) + 0.15 \log(d - b).$$

Differentiate with respect to b:

$$\frac{dEU}{db} = \frac{0.85}{d + b} - \frac{0.15}{d - b}.$$

Setting the derivative equal to zero gives

$$\frac{0.85}{d + b} = \frac{0.15}{d - b} \quad \Rightarrow \quad b^* = 0.7d.$$

Thus, the optimal fraction of wealth to bet under log utility is 70%.

1.3 (c)

This depends on the proportion of liquid funds in Net Worth. If this ratio is greater than 0.7, I will accept it; otherwise, I will not. Because if I lose the bet, I will only have 30% of my net worth left, and all of this

is fixed funds, such as houses, cars, and other necessities for life. I cannot liquidate them, so there will be no liquid assets available for purchasing daily necessities and groceries. My living standard will be significantly reduced.

2 Q2

Let $Y \in \{0, 1\}$ be a binary random variable, and let X be covariates. Denote $p(x) = P(Y = 1 \mid X = x)$. Suppose the predictive function f. The range $f(x) \in [0, 1]$.

The **Cross-Entropy Loss** (or Log Loss) is defined as:

$$L(Y, f(X)) = -[Y \log f(X) + (1 - Y) \log(1 - f(X))].$$

We want to show that this loss is a *proper scoring rule*, i.e.

$$\mathbb{E}_{Y|X=x}[L(Y, f(X))]$$

is minimized when $f(x) = P(Y = 1|X = x) = p(x)$

Step 1: Compute the conditional risk.

For fixed x, define the conditional expected loss:

$$R(f(x)) = \mathbb{E}_{Y|X=x}[L(Y, f(X))].$$

Substituting the definition of the cross-entropy loss:

$$R(f(x)) = -\left[p(x) \log f(x) + (1 - p(x)) \log(1 - f(x))\right].$$

Step 2: Differentiate to find the extremum, i.e. $F.O.C = 0$

Take the derivative of $R(f(x))$ with respect to $f(x)$:

$$\frac{dR}{df} = -\left[\frac{p(x)}{f(x)} - \frac{1 - p(x)}{1 - f(x)}\right].$$

Setting this equal to zero gives:

$$\frac{p(x)}{f(x)} = \frac{1 - p(x)}{1 - f(x)}.$$

Solving for $f(x)$:

$$p(x)(1 - f(x)) = (1 - p(x))f(x).$$
$$p(x) = f(x).$$

Thus, the minimizer is $f^*(x) = p(x) = P(Y = 1|X = x)$.

Step 3: Verify it is a minimum, i.e. $SOC > 0$

The second derivative is:

$$\frac{d^2R}{df^2} = \frac{p(x)}{f(x)^2} + \frac{1 - p(x)}{(1 - f(x))^2} > 0,$$

so $R(f(x))$ is convex and when $f^*(x) = p(x) = P(Y = 1|X = x)$, the conditional risk $R(f^*(x))$ is a minimum. The proof is completed.

3 Q3

1. In realistic training, we do not run SGD indefinitely. We stop after a finite number of epochs. Hence, true convergence to the global minimum is not required.

2. A constant step size in SGD maintains stochastic noise in the updates. This noise prevents the model from overfitting by making it difficult to settle into sharp minima, leading to flatter, wider minima that are known to generalize better.

Therefore, while a decaying learning rate is required for theoretical convergence to a precise point, a well-chosen constant learning rate is a common and practical approach that leads to robust, generalizable models in a finite sample setting.

4 Q4

According to the type of target variable, I chose four regression models to test the normality of the residuals of the four fitted models: Linear Model, Support Vector Machine, Decision Trees, and Neural Network.

In the Q-Q plot, if the residuals fall approximately along the diagonal line, the normality is held. From the Figure 1, we can see residuals from the fitted Decision Trees model fall approximately along the diagonal line, while the other three models don't show the sign.

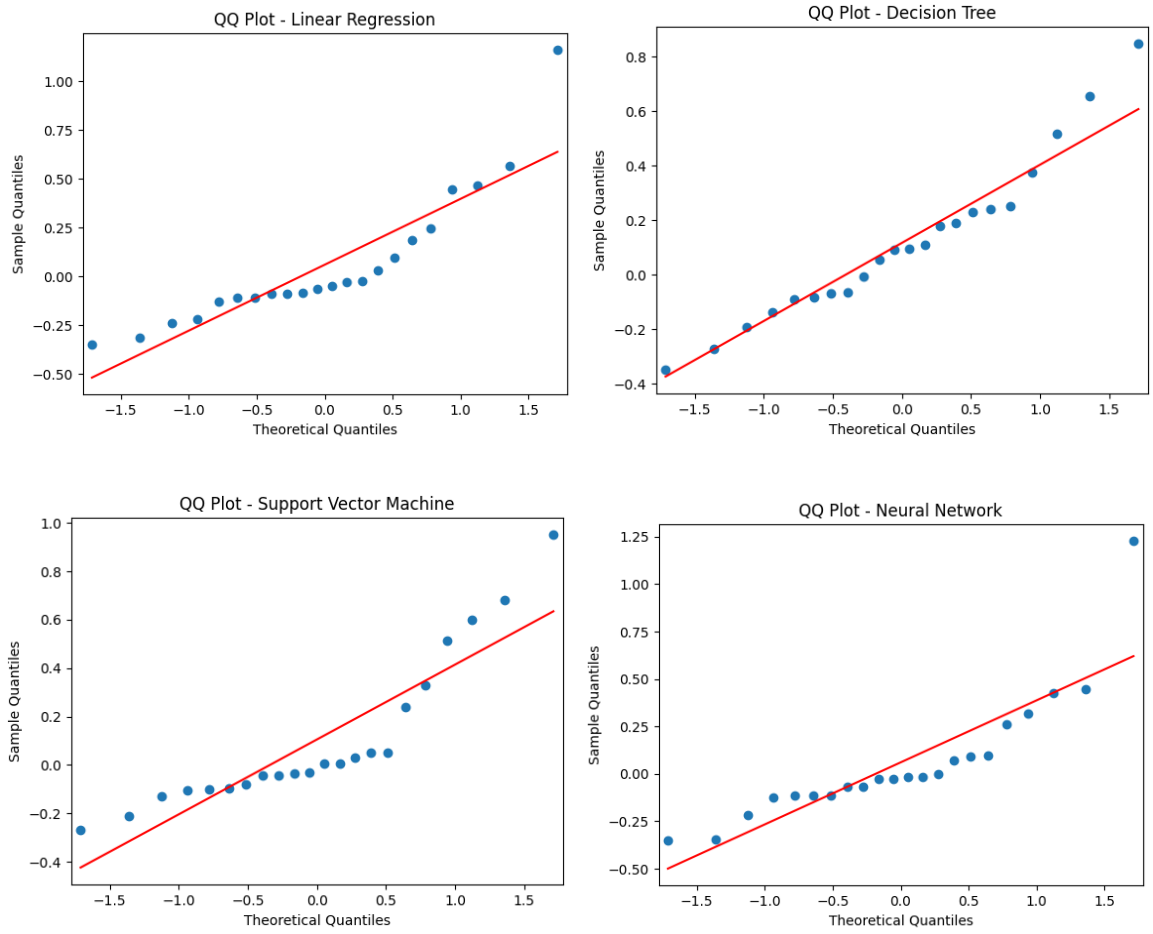

Figure 1: Q-Q plot

The `normaltest` from `scipy.stats` will create statistical testing for the residuals' normality. A small p-value

(e.g. $p < 0.05$) suggests non-normal residuals. Table 1 shows that among the fitted models, only the residuals from the Decision Trees model follow the normal distribution.

Table 1: Test for whether the residuals follow a normal distribution

Metric	**Linear Regression**	**Decision Trees**	**Support Vector Machine**	**Neural Network**
Normaltest statistic	16.494	3.699	8.974	23.552
P-value	0.000	0.157	0.011	0.000

Therefore, we should not represent the loss functions of all machine learning models using the squared error loss. Because not all the residuals of the fitted machine learning models follow a normal distribution.

5 Q5

I used `JAX` and `Optax` to implement a Linear regression problem with a squared error loss function. The loss is defined as:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

where $\hat{y}_i = f_\theta(x_i)$ denotes the model's prediction given parameters θ.

The code below is the implementation in JAX. The function `jax.grad(loss_fn)` computes the gradient $\nabla_\theta L(\theta)$ automatically. The `optax.sgd` optimizer updates the parameters using gradient descent.

```
import jax
import jax.numpy as jnp
import optax

# Generate simple synthetic data
key = jax.random.key(0)
x = jnp.linspace(-1, 1, 20)
true_w, true_b = 2.0, -1.0
y = true_w * x + true_b + 0.1 * jax.random.normal(key, x.shape)

# Define prediction model: y_hat = w * x + b
def predict(params, x):
    w, b = params
    return w * x + b

# Define squared loss function
def loss_fn(params, x, y):
    y_hat = predict(params, x)
    return jnp.mean((y - y_hat) ** 2)

# Initialize parameters (w, b)
params = jnp.array([1.0, 0.0])  # initial guess
print("Initial loss:", loss_fn(params, x, y))

# Set up optimizer
optimizer = optax.sgd(learning_rate=0.1)
opt_state = optimizer.init(params)
```

```
# Optimization loop
for step in range(20):
    grads = jax.grad(loss_fn)(params, x, y)
    updates, opt_state = optimizer.update(grads, opt_state, params)
    params = optax.apply_updates(params, updates)
    print(f"Step {step+1}: loss = {loss_fn(params, x, y):.4f}")

print("Estimated parameters:", params)
```

```
Initial loss: 1.3094845
Step 1: loss = 0.9187
Step 2: loss = 0.6578
Step 3: loss = 0.4815
Step 4: loss = 0.3608
Step 5: loss = 0.2767
Step 6: loss = 0.2170
Step 7: loss = 0.1738
Step 8: loss = 0.1419
Step 9: loss = 0.1177
Step 10: loss = 0.0991
Step 11: loss = 0.0844
Step 12: loss = 0.0727
Step 13: loss = 0.0632
Step 14: loss = 0.0554
Step 15: loss = 0.0490
Step 16: loss = 0.0436
Step 17: loss = 0.0390
Step 18: loss = 0.0352
Step 19: loss = 0.0319
Step 20: loss = 0.0292
Estimated parameters: [ 1.7623509  -0.96247524]
```

From the result, we can see that the loss decreases over iterations, showing that automatic differentiation and gradient-based optimization successfully minimize the squared loss.

6 Classification

The dataset `breast cancer wisconsin` contains 30 features and a binary label. The sample size is 569.

The baseline is from sklearn's implementation. The one line can implement the regression directly. The number of iterations was set to 10,000, and the same standard was applied to the other two models as well.

```
LogisticRegression(random_state=0, max_iter=10000)
```

By using the JAX framework, I manually create the Logistic and Probit regression. The difference between these two models is the conversion from the logits to the probabilities. Logistic regression uses the sigmoid function, while Probit regression uses the cumulative distribution function of the standard normal distribution. The complete code can be found here.

Table 2 shows that the Logistic regression implemented in the JAX framework outperforms the baseline. The Probit regression implemented in the JAX framework gets the same accuracy rate as the baseline.

Table 2: Comparison of the performance of Logistic and Probit regressions with binary cross-entropy loss

Metric	Baseline	Logistic	Probit
Accuracy rate (%)	0.9737	0.9825	0.9737

7 Regression

The dataset `Average Localization Error (ALE) in sensor node localization process in WSNs` contains 4 features and one target variable (here, we only need the ALE variable). The sample size is 107.

The baseline is from sklearn's implementation. The one line can implement the regression directly.

```
LinearRegression()
```

By using the JAX framework, I manually create the two Linear regression model with two different loss functions. The difference between them is the underlying assumption of the residuals' distribution. The squared error loss function needs the normality of the residuals. In question 4, we see that the residuals in Linear regression significantly deviate from normality. Since the t-distribution has heavier tails, it matches what we see in the actual residuals distribution. So we use the negative log-likelihood to display the loss function.

Table 3 shows that the performance of linear regression with the loss function in squared error format has the same MAE as the baseline, while the loss function expressed by negative log-likelihood of the t-distribution performs better than the baseline. The complete code can be found here.

Table 3: Comparison of the performance of two linear regression models with different loss functions

Metric	Baseline	Squred error	Negative log-pdf of the t-distribution
Mean absolute error (MAE)	0.2319	0.2319	0.2315

Note: The performance of the Linear regression model is measured by mean absolute error.