DATA 221 Homework 4

Chris Low

Problem 1

```
from sklearn.datasets import load_breast_cancer
bc = load_breast_cancer()
X = bc.data
y = bc.target
```

1(a)

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
```

1(b)

Standardization is unnecessary for decision trees because splits depend only on the ordering of feature values, such as whether a feature exceeds a threshold. Scaling a feature is a monotonic transformation and does not change this ordering, so the resulting tree structure is unchanged.

In contrast, **logistic regression is trained using gradient-based optimization.** When features are on very different scales, the loss surface becomes poorly conditioned, which leads to unstable updates and slow convergence. Standardizing the predictors puts them on comparable scales, so we get more stable gradients and faster convergence.

1(c)

The standardization parameters, such as the mean and standard deviation, should be computed using only the training data to **avoid data leakage.** The test set is meant to represent unseen future data. If its information is used when scaling the predictors, the model indirectly gains access to the test distribution, which can lead to overly optimistic performance estimates. Doing this keeps the test set truly out of sample and matches how the model would be applied to new data in practice.

1(d)

Let

$$p_i \;=\; P(y_i = 1 \mid X_i) \;=\; \sigma(\eta_i), \qquad \eta_i = w^\top X_i, \qquad \sigma(z) = \frac{1}{1+e^{-z}}.$$

Since $y_i \in \{0, 1\}$, the Bernoulli likelihood for observation i is

$$P(y_i \mid X_i) = p_i^{y_i}(1-p_i)^{1-y_i}.$$

Assuming independent observations, the log-likelihood for $\{(X_i, y_i)\}_{i=1}^n$ is

$$\ell(w) = \sum_{i=1}^{n} \log P(y_i \mid X_i) = \boxed{\sum_{i=1}^{n} \Big[y_i \log(p_i) + (1-y_i)\log(1-p_i) \Big].}$$

where $p_i = \sum_{j=1}^{d} \sigma(w_j X_{ij})$.

1(e)

The binary cross-entropy loss (average form) is

$$J(w) = -\frac{1}{n} \sum_{i=1}^{n} \Big[y_i \log(p_i) + (1-y_i)\log(1-p_i) \Big].$$

Comparing with the log-likelihood in part (d),

$$J(w) = -\frac{1}{n}\,\ell(w).$$

Multiplying by $-1/n$ only rescales and flips the optimization direction, so the optimizer is the same.

So, we can conclude that minimizing cross-entropy is equivalent to maximizing the log-likelihood.

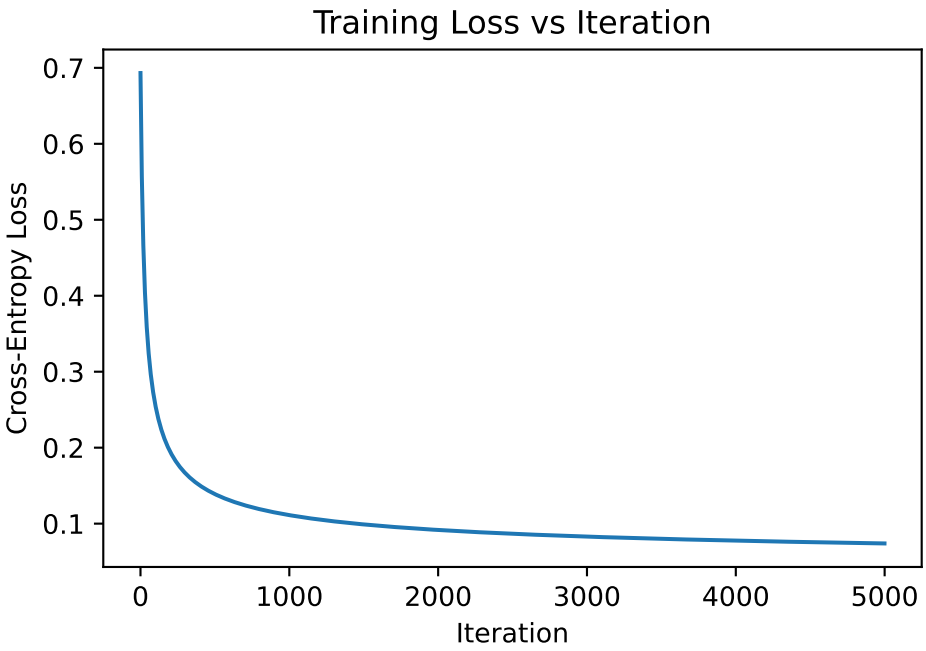
1(f)

```python
import numpy as np

def sigmoid(z):
    z = np.clip(z, -500, 500)
    return 1.0 / (1.0 + np.exp(-z))

def cross_entropy(y, p):
    eps = 1e-12
    p = np.clip(p, eps, 1 - eps)
    return -np.mean(y * np.log(p) + (1 - y) * np.log(1 - p))

def fit_logistic_regression(
    X, y,
    lr=0.01,
    max_iter=1000,
    tol=1e-4
):
    # Initialize coefficients
    n, d = X.shape
    w = np.zeros(d)
    b = 0.0

    losses = []

    for t in range(max_iter):
        scores = X @ w + b
        probs = sigmoid(scores)

        # Compute cross-entropy loss
        loss = cross_entropy(y, probs)
        losses.append(loss)

        # Compute gradients
```

```
        grad_w = (1 / n) * X.T @ (probs - y)
        grad_b = (1 / n) * np.sum(probs - y)

        # Update coefficients
        w -= lr * grad_w
        b -= lr * grad_b

        # Reasonable stopping criterion
        if t > 0 and abs(losses[-2] - losses[-1]) < tol:
            break

    return w, b, losses
```

1(g)

```
import matplotlib.pyplot as plt

w_hat, b_hat, losses = fit_logistic_regression(
    X_train_std, y_train,
    lr=0.01,
    max_iter=5000,
    tol=1e-6
)

plt.figure()
plt.plot(losses)
plt.xlabel("Iteration")
plt.ylabel("Cross-Entropy Loss")
plt.title("Training Loss vs Iteration")
plt.show()
```



The loss drops quickly at the beginning and then flattens out, which suggests that gradient descent is converging. After a certain point, additional iterations do not noticeably improve the loss.

1(h)

Due to using L2 regularization by default, we need to set a large value of C to use it without regularization.

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(
    C=1e6,
    solver="lbfgs",
    max_iter=5000
)

clf.fit(X_train_std, y_train)


w_sklearn = clf.coef_.flatten()
b_sklearn = clf.intercept_[0]
```

```
probs_sklearn = clf.predict_proba(X_train_std)[:, 1]
sklearn_loss = cross_entropy(y_train, probs_sklearn)

print("Final GD loss:", losses[-1])
print("Final sklearn loss:", sklearn_loss)

print("First 5 GD coefficients:", w_hat[:5])
print("First 5 sklearn coefficients:", w_sklearn[:5])
print("GD intercept:", b_hat)
print("Sklearn intercept:", b_sklearn)
```

```
Final GD loss: 0.0739236175496964
Final sklearn loss: 0.0002954851332951214
First 5 GD coefficients: [-0.48241728 -0.55337154 -0.46661102 -0.50266762 -
0.23216018]
First 5 sklearn coefficients: [ 22.91087147  -6.13887295  45.08367648   7.97471529 -
23.28746162]
GD intercept: 0.5012271610638136
Sklearn intercept: -36.98822516136943
```

Both models minimize the same cross-entropy objective, but they behave differently during optimization. The breast cancer data are close to linearly separable, so an unregularized logistic regression (sklearn) fit can keep increasing the magnitude of the coefficients to push the training loss closer to zero. **In contrast, the gradient descent implementation stops once the improvement in loss becomes negligible, as enforced by the stopping rule** `abs(losses[-2] - losses[-1]) < tol`.

As a result, the gradient descent solution has finite, moderate coefficients and a slightly higher loss, while sklearn continues toward very large coefficient values. Despite these differences, both models yield similar predictions and decision boundaries.

1(i)

Assume the coefficients are independent and satisfy the prior

$$w_j \sim \mathcal{N}(0, \tau^2).$$

The joint prior over $w = (w_1, \dots, w_l)$ is

$$p(w) = \prod_{j=1}^{l} \frac{1}{\sqrt{2\pi\tau^2}} \exp\left(-\frac{w_j^2}{2\tau^2}\right).$$

Taking logs gives the log-prior:

$$\log p(w) = \sum_{j=1}^{l} \left[-\frac{1}{2} \log(2\pi\tau^2) - \frac{w_j^2}{2\tau^2} \right] = -\frac{l}{2} \log(2\pi\tau^2) - \frac{1}{2\tau^2} \sum_{j=1}^{l} w_j^2.$$

Up to an additive constant:

$$\boxed{\log p(w) = -\frac{1}{2\tau^2} \|w\|_2^2 + C}$$

1(j)

The posterior distribution satisfies

$$p(w \mid X, y) \propto p(y \mid X, w)\, p(w).$$

Taking logs,

$$\log p(w \mid X, y) = \log p(y \mid X, w) + \log p(w) + C.$$

Maximizing the posterior equals minimizing the negative log-posterior:

$$-\log p(w \mid X, y) = -\log p(y \mid X, w) - \log p(w) + C.$$

For logistic regression, $-\log p(y \mid X, w)$ is the cross-entropy loss (up to averaging). Using the result from part (i),

$$-\log p(w) = \frac{1}{2\tau^2} \|w\|_2^2 + C.$$

Therefore, the MAP estimation is:

$$\min_w \quad \text{CrossEntropy}(w) + \frac{1}{2\tau^2} \|w\|_2^2.$$

The second term is exactly an **L2 regularization (ridge) penalty**, with regularization strength that is inversely proportional to τ^2.

1(k)

The prior controls how large the model coefficients are allowed to be. A strong prior (small τ^2) implies a strong penalty, shrinking the coefficients toward zero and making the model simpler and less sensitive to noise. This **reduces variance but can increase bias** if the model becomes too constrained. A weak prior (large τ^2) allows larger coefficients and a more flexible model, which can **reduce bias but increase variance and the risk of overfitting**. Thus, the prior variance determines the bias–variance tradeoff.