Homework 1: Clustering and Regression

6530182121 Thananop Kullapan

January 21, 2026

Part 1: Metrics (T1–T4, OT1)

Model A Confusion Matrix

From the problem description:

- Total Population $= 30 + 20 + 10 + 40 = 100$
- Actual Dog: 50 (30 Pred Dog, 20 Pred Cat)
- Actual Cat: 50 (10 Pred Dog, 40 Pred Cat)

T1. Accuracy of Model A

$$\text{Accuracy} = \frac{TP + TN}{\text{Total}} = \frac{30 + 40}{100} = \frac{70}{100} = 0.7$$

T2. Cat as Positive Class (Class 1)

- TP (Actual Cat, Pred Cat) $= 40$
- FP (Actual Dog, Pred Cat) $= 20$
- FN (Actual Cat, Pred Dog) $= 10$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{40}{40 + 20} = \frac{40}{60} \approx 0.667$$
$$\text{Recall} = \frac{TP}{TP + FN} = \frac{40}{40 + 10} = \frac{40}{50} = 0.8$$
$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{0.667 \times 0.8}{0.667 + 0.8} \approx 0.727$$

T3. Dog as Positive Class (Class 1)

- TP (Actual Dog, Pred Dog) $= 30$
- FP (Actual Cat, Pred Dog) $= 10$
- FN (Actual Dog, Pred Cat) $= 20$

$$\text{Precision} = \frac{30}{30 + 10} = \frac{30}{40} = 0.75$$
$$\text{Recall} = \frac{30}{30 + 20} = \frac{30}{50} = 0.6$$
$$F1 = 2 \times \frac{0.75 \times 0.6}{0.75 + 0.6} = 2 \times \frac{0.45}{1.35} \approx 0.667$$

T4. Lopsided Population (80% Cats)

New Population Distribution: $P(\text{Cat}) = 0.8$, $P(\text{Dog}) = 0.2$. Model properties (Sensitivity/Recall) remain constant:

- TPR_{dog} (Recall for Dog) $= 0.6$
- TNR_{dog} (Recall for Cat) $= 0.8$
- FPR_{dog} $(1 - TNR) = 0.2$

Accuracy:

$$\text{Acc} = (TPR \times P(\text{Dog})) + (TNR \times P(\text{Cat})) = (0.6 \times 0.2) + (0.8 \times 0.8) = 0.12 + 0.64 = 0.76$$

Precision, Recall, F1 (Dog as Positive): Recall remains **0.6**.

$$\text{Precision} = \frac{TPR \times P(\text{Dog})}{TPR \times P(\text{Dog}) + FPR \times P(\text{Cat})} = \frac{0.6 \times 0.2}{(0.6 \times 0.2) + (0.2 \times 0.8)} = \frac{0.12}{0.12 + 0.16} = \frac{0.12}{0.28} \approx 0.429$$

Explanation: Accuracy increases because the model is better at predicting the majority class (Cats). However, Precision for Dogs drops significantly because the large number of Cats generates more False Positives relative to the small number of True Dog Positives.

OT1. Accuracy vs F1

Accuracy equals F1 when $TN = FP + FN - TP$. Generally, Accuracy can be misleadingly high in imbalanced datasets where TN is very large (easy negatives), whereas F1 focuses on the positive class performance.

Part 2: Clustering (T5–T7, OT2)

T5 & T6. K-Means Manual Trace Code

The following Python code simulates the K-Means steps (Assign and Update) for the given starting points.

```python
import numpy as np

# Data Points
X = np.array([
    [1, 2], [3, 3], [2, 2],
    [8, 8], [6, 6], [7, 7],
    [-3, -3], [-2, -4], [-7, -7]
])

def run_kmeans_step(data, centroids, label):
    print(f"--- {label} ---")
    iteration = 0
    while True:
        iteration += 1
        print(f"Iteration {iteration}")
        # 1. Assign Step
        clusters = [[] for _ in range(len(centroids))]
        for x in data:
            dists = [np.linalg.norm(x - c) for c in centroids]
            cluster_idx = np.argmin(dists)
            clusters[cluster_idx].append(x)

        # 2. Update Step
        new_centroids = []
        max_shift = 0
        for i, cluster in enumerate(clusters):
            if not cluster: new_c = centroids[i]
            else: new_c = np.mean(cluster, axis=0)
            shift = np.linalg.norm(new_c - centroids[i])
            max_shift = max(max_shift, shift)
            new_centroids.append(new_c)
            print(f"  Cluster {i}: New Mean {new_c}")

        if max_shift < 1e-4:
            break
        centroids = np.array(new_centroids)

# Run T5
start_t5 = np.array([[3.0, 3.0], [2.0, 2.0], [-3.0, -3.0]])
run_kmeans_step(X, start_t5, "T5")

# Run T6
start_t6 = np.array([[-3.0, -3.0], [2.0, 2.0], [-7.0, -7.0]])
run_kmeans_step(X, start_t6, "T6")
```

Listing 1: K-Means Simulation Code

T7. Comparison

The starting points in **T5 are better**. They are well-distributed across the natural clusters (small positive, large positive, and negative). T6 has two starting points in the negative region, which causes poor convergence where one natural cluster might be split or two merged incorrectly. **Measure of Goodness:** We can measure the quality using **Inertia (Sum of Squared Errors)**, which calculates the sum of squared distances from each point to its assigned centroid. Lower Inertia indicates better clustering.

OT2. Best K

By visual inspection, the data naturally forms **3 clusters**:

1. Negative values $((-3,-3),(-2,-4),(-7,-7))$
2. Small positive values $((1,2),(2,2),(3,3))$
3. Large positive values $((6,6),(7,7),(8,8))$

Therefore, $K = 3$ is optimal.

Part 3: Titanic Logistic Regression (T8–T13, OT3–OT4)

T8-T11. Full Implementation Code

The following code performs data preprocessing, implements Logistic Regression with Gradient Descent from scratch, and generates the Kaggle submission file.

```python
import numpy as np
import pandas as pd

# Load Data (Simulated paths for report)
train_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/train.csv
    "
test_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/test.csv"
train = pd.read_csv(train_url)
test = pd.read_csv(test_url)

# --- T8: Median Age ---
median_age = train["Age"].median()
train["Age"] = train["Age"].fillna(median_age)
test["Age"] = test["Age"].fillna(median_age)
print(f"Median Age: {median_age}")

# --- T9: Preprocessing ---
# Fill Embarked
mode_embarked = train["Embarked"].mode()[0]
train["Embarked"] = train["Embarked"].fillna(mode_embarked)
test["Embarked"] = test["Embarked"].fillna(mode_embarked)

# Mapping to Numbers
def preprocess(df):
    df.loc[df["Sex"] == "male", "Sex"] = 0
    df.loc[df["Sex"] == "female", "Sex"] = 1
    df.loc[df["Embarked"] == "S", "Embarked"] = 0
    df.loc[df["Embarked"] == "C", "Embarked"] = 1
    df.loc[df["Embarked"] == "Q", "Embarked"] = 2
    return df

train = preprocess(train)
test = preprocess(test)

# --- T10: Logistic Regression Class ---
class LogisticRegressionGD:
    def __init__(self, lr=0.01, iter=5000):
        self.lr = lr
        self.iter = iter

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-np.clip(z, -250, 250)))

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.iter):
            y_pred = self.sigmoid(np.dot(X, self.w) + self.b)
            dw = (1/n_samples) * np.dot(X.T, (y_pred - y))
            db = (1/n_samples) * np.sum(y_pred - y)
            self.w -= self.lr * dw
            self.b -= self.lr * db

    def predict(self, X):
        y_pred = self.sigmoid(np.dot(X, self.w) + self.b)
        return [1 if i >= 0.5 else 0 for i in y_pred]

# Training
features = ["Pclass", "Sex", "Age", "Embarked"]
X = np.array(train[features].values, dtype=float)
y = np.array(train["Survived"].values, dtype=float)

# Normalize (Crucial for GD)
mean_X, std_X = np.mean(X, axis=0), np.std(X, axis=0)
X_norm = (X - mean_X) / (std_X + 1e-8)

model = LogisticRegressionGD()
model.fit(X_norm, y)

# --- T11: Submission ---
X_test = np.array(test[features].values, dtype=float)
X_test_norm = (X_test - mean_X) / (std_X + 1e-8)
preds = model.predict(X_test_norm)

submission = pd.DataFrame({
    "PassengerId": test["PassengerId"],
    "Survived": preds
})
# submission.to_csv("submission.csv", index=False)
print("Submission created.")
```

Listing 2: Titanic Logistic Regression Implementation

T12. Higher Order Features

Adding higher-order features (e.g., Age^2, $Age \times Pclass$) generally allows the model to capture non-linear relationships.

- **Training Set:** Accuracy usually increases because the model fits the data more closely.
- **Test Set:** Accuracy might improve if the relationship is truly non-linear, but risk of **overfitting** increases. If too many features are added without regularization, test accuracy will drop.

T13. Reduced Features (Sex and Age only)

Reducing features simplifies the model.

- **Result:** Accuracy likely decreases slightly because 'Pclass' is a strong predictor of survival (higher class had priority).
- **Benefit:** The model becomes less prone to overfitting and easier to interpret.

OT3 & OT4. Matrix Inversion (Linear Regression)

Using Normal Equation $\theta = (X^TX)^{-1}X^Ty$:

- The weights obtained from Linear Regression (minimizing MSE) will **differ** from Logistic Regression (minimizing Cross-Entropy).
- Linear regression treats the classification problem as fitting a line through 0s and 1s, which is not theoretically ideal for probability estimation but often works as a rough approximation.

Part 4: Matrix Algebra Proofs (OT5–OT7)

OT5. Prove $\nabla_A \mathbf{tr}(AB) = B^T$

Let $Y = AB$. The trace is $\text{tr}(AB) = \sum_i (AB)_{ii} = \sum_i \sum_k A_{ik}B_{ki}$. Partial derivative w.r.t A_{mn}:

$$\frac{\partial}{\partial A_{mn}} \sum_i \sum_k A_{ik}B_{ki}$$

This is non-zero only when $i = m$ and $k = n$.

$$\frac{\partial}{\partial A_{mn}}(A_{mn}B_{nm}) = B_{nm}$$

Thus, the (m, n) entry of the gradient matrix is B_{nm}, which corresponds to the element (n, m) of B.

$$\therefore \nabla_A \text{tr}(AB) = B^T$$

OT6. Prove $\nabla_{A^T} f(A) = (\nabla_A f(A))^T$

Let $Y = \nabla_A f(A)$, so $Y_{ij} = \frac{\partial f}{\partial A_{ij}}$. Let $B = A^T$, implying $B_{ji} = A_{ij}$. We want $Z = \nabla_B f(B^T) = \nabla_{A^T} f(A)$.

$$Z_{ji} = \frac{\partial f}{\partial B_{ji}} = \frac{\partial f}{\partial A_{ij}} = Y_{ij}$$

The (j, i) element of the result is the (i, j) element of the gradient w.r.t A.

$$\therefore \nabla_{A^T} f(A) = (\nabla_A f(A))^T$$

OT7. Prove $\nabla_A \mathbf{tr}(ABA^TC) = CAB + C^TAB^T$

Using the product rule and cyclic property of trace:

$$d(\text{tr}(ABA^TC)) = \text{tr}(dA \cdot BA^TC) + \text{tr}(AB \cdot dA^T \cdot C)$$

Using $\text{tr}(XY) = \text{tr}(YX)$ and $\text{tr}(X) = \text{tr}(X^T)$:

- Term 1: $\text{tr}(BA^TC \cdot dA)$
- Term 2: $\text{tr}(AB(dA)^TC) = \text{tr}(CAB(dA)^T) = \text{tr}((CAB(dA)^T)^T) = \text{tr}(dA \cdot B^TA^TC^T)$

Combine:

$$\text{tr}((BA^TC + B^TA^TC^T)dA)$$

The gradient is the transpose of the term inside the parenthesis:

$$\nabla_A = (BA^TC + B^TA^TC^T)^T = C^TAB^T + CAB$$

Rearranging:

$$= CAB + C^TAB^T$$