**LAKSHYA GURUNG**

Worksheet 7: Regularization and Logistic Regression

```
import numpy as np
import pandas as pd
import os

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, accuracy_score, classification_report
from sklearn.linear_model import LinearRegression, Ridge, Lasso, LogisticRegression
from sklearn.datasets import load_breast_cancer

import kagglehub
```

```
path = kagglehub.dataset_download("camnugent/california-housing-prices")
housing_data_path = os.path.join(path, "housing.csv")

df = pd.read_csv(housing_data_path)
df.head()
```

```
Using Colab cache for faster access to the 'california-housing-prices' dataset.
```

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | median_income | med |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -122.23 | 37.88 | 41.0 | 880.0 | 129.0 | 322.0 | 126.0 | 8.3252 | |
| **1** | -122.22 | 37.86 | 21.0 | 7099.0 | 1106.0 | 2401.0 | 1138.0 | 8.3014 | |
| **2** | -122.24 | 37.85 | 52.0 | 1467.0 | 190.0 | 496.0 | 177.0 | 7.2574 | |
| **3** | -122.25 | 37.85 | 52.0 | 1274.0 | 235.0 | 558.0 | 219.0 | 5.6431 | |
| **4** | -122.25 | 37.85 | 52.0 | 1627.0 | 280.0 | 565.0 | 259.0 | 3.8462 | |

Next steps: Generate code with df | New interactive sheet

```
df = pd.get_dummies(df, drop_first=True)
df = df.dropna()

X = df.drop("median_house_value", axis=1)
Y = df["median_house_value"]
```

```
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42
)

print(X_train.shape, X_test.shape)
```

```
(16346, 12) (4087, 12)
```

```
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

scaler_Y = StandardScaler()
Y_train_scaled = scaler_Y.fit_transform(Y_train.values.reshape(-1,1)).ravel()
Y_test_scaled = scaler_Y.transform(Y_test.values.reshape(-1,1)).ravel()
```

```
linear_model = LinearRegression()
linear_model.fit(X_train_scaled, Y_train_scaled)

y_train_pred = linear_model.predict(X_train_scaled)
y_test_pred = linear_model.predict(X_test_scaled)

print("Train MSE:", mean_squared_error(Y_train_scaled, y_train_pred))
print("Test MSE:", mean_squared_error(Y_test_scaled, y_test_pred))
```

```
Train MSE: 0.3543517602353898
Test MSE: 0.36278746237466286
```

```
coef_df = pd.DataFrame({
    "Feature": X.columns,
    "Coefficient": linear_model.coef_
}).sort_values("Coefficient", key=abs, ascending=False)

coef_df
```

| | Feature | Coefficient |
|---|---|---|
| 7 | median_income | 0.647872 |
| 1 | latitude | -0.476378 |
| 0 | longitude | -0.472620 |
| 4 | total_bedrooms | 0.373744 |
| 5 | population | -0.357397 |
| 8 | ocean_proximity_INLAND | -0.158489 |
| 6 | households | 0.141739 |
| 3 | total_rooms | -0.118322 |
| 2 | housing_median_age | 0.118209 |
| 9 | ocean_proximity_ISLAND | 0.025155 |
| 10 | ocean_proximity_NEAR BAY | -0.017129 |
| 11 | ocean_proximity_NEAR OCEAN | 0.009235 |

Next steps: Generate code with coef_df | New interactive sheet

```
alpha_grid = {"alpha": [0.001, 0.01, 0.1, 10, 1000, 10000]}

ridge = Ridge()
ridge_grid = GridSearchCV(
    ridge,
    alpha_grid,
    cv=5,
    scoring="neg_mean_squared_error",
    n_jobs=-1,
    verbose=1
)

ridge_grid.fit(X_train_scaled, Y_train_scaled)
ridge_grid.best_params_
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
{'alpha': 10}
```

```
best_ridge = Ridge(alpha=ridge_grid.best_params_["alpha"])
best_ridge.fit(X_train_scaled, Y_train_scaled)

ridge_train_pred = best_ridge.predict(X_train_scaled)
ridge_test_pred = best_ridge.predict(X_test_scaled)

print("Ridge Train MSE:", mean_squared_error(Y_train_scaled, ridge_train_pred))
print("Ridge Test MSE:", mean_squared_error(Y_test_scaled, ridge_test_pred))
```

```
Ridge Train MSE: 0.3543593234096068
Ridge Test MSE: 0.3627983942738068
```

```
lasso = Lasso(max_iter=10000)

lasso_grid = GridSearchCV(
    lasso,
    alpha_grid,
    cv=5,
    scoring="neg_mean_squared_error",
    n_jobs=-1,
    verbose=1
)

lasso_grid.fit(X_train_scaled, Y_train_scaled)
lasso_grid.best_params_
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
{'alpha': 0.001}
```

```
best_lasso = Lasso(alpha=lasso_grid.best_params_["alpha"], max_iter=10000)
best_lasso.fit(X_train_scaled, Y_train_scaled)

lasso_train_pred = best_lasso.predict(X_train_scaled)
lasso_test_pred = best_lasso.predict(X_test_scaled)

print("Lasso Train MSE:", mean_squared_error(Y_train_scaled, lasso_train_pred))
print("Lasso Test MSE:", mean_squared_error(Y_test_scaled, lasso_test_pred))
```

```
Lasso Train MSE: 0.3544639569212034
Lasso Test MSE: 0.3629258203721613
```

```
cancer = load_breast_cancer()
X_cancer = cancer.data
y_cancer = cancer.target
feature_names = cancer.feature_names
target_names = cancer.target_names
```

```
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(
    X_cancer, y_cancer, test_size=0.2, random_state=42, stratify=y_cancer
)
```

```
scaler_c = StandardScaler()
X_train_c_scaled = scaler_c.fit_transform(X_train_c)
X_test_c_scaled = scaler_c.transform(X_test_c)
```

```
baseline_logistic = LogisticRegression(max_iter=1000, random_state=42)
baseline_logistic.fit(X_train_c_scaled, y_train_c)

y_train_pred_base = baseline_logistic.predict(X_train_c_scaled)
y_test_pred_base = baseline_logistic.predict(X_test_c_scaled)

print("Train Accuracy:", accuracy_score(y_train_c, y_train_pred_base))
print("Test Accuracy:", accuracy_score(y_test_c, y_test_pred_base))
```

```
Train Accuracy: 0.989010989010989
Test Accuracy: 0.9824561403508771
```

```
print(classification_report(y_test_c, y_test_pred_base, target_names=target_names))
```

```
              precision    recall  f1-score   support

   malignant       0.98      0.98      0.98        42
      benign       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

```
param_grid = {
    "C": [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    "penalty": ["l1", "l2"],
    "solver": ["liblinear", "saga"]
}

logistic_grid = GridSearchCV(
    LogisticRegression(max_iter=5000, random_state=42),
    param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1,
    verbose=1
)

logistic_grid.fit(X_train_c_scaled, y_train_c)
logistic_grid.best_params_
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

```
{'C': 0.1, 'penalty': 'l2', 'solver': 'saga'}
```

```
l1_model = LogisticRegression(
    penalty="l1",
    C=1.0,
    solver="liblinear",
    max_iter=5000,
    random_state=42
)

l2_model = LogisticRegression(
    penalty="l2",
    C=0.1,
    solver="lbfgs",
    max_iter=5000,
    random_state=42
)

l1_model.fit(X_train_c_scaled, y_train_c)
l2_model.fit(X_train_c_scaled, y_train_c)
```

```
LogisticRegression
LogisticRegression(C=0.1, max_iter=5000, random_state=42)
```

```
comparison_df = pd.DataFrame({
    "Model": ["Baseline", "L1", "L2"],
    "Train Accuracy": [
        accuracy_score(y_train_c, baseline_logistic.predict(X_train_c_scaled)),
        accuracy_score(y_train_c, l1_model.predict(X_train_c_scaled)),
        accuracy_score(y_train_c, l2_model.predict(X_train_c_scaled))
    ],
    "Test Accuracy": [
        accuracy_score(y_test_c, baseline_logistic.predict(X_test_c_scaled)),
        accuracy_score(y_test_c, l1_model.predict(X_test_c_scaled)),
        accuracy_score(y_test_c, l2_model.predict(X_test_c_scaled))
    ],
    "Non-Zero Coefficients": [
        np.sum(baseline_logistic.coef_[0] != 0),
        np.sum(l1_model.coef_[0] != 0),
        np.sum(l2_model.coef_[0] != 0)
    ]
})

comparison_df
```

| | Model | Train Accuracy | Test Accuracy | Non-Zero Coefficients |
|---|---|---|---|---|
| 0 | Baseline | 0.989011 | 0.982456 | 30 |
| 1 | L1 | 0.989011 | 0.991228 | 16 |
| 2 | L2 | 0.986813 | 0.973684 | 30 |

Next steps: Generate code with comparison_df | New interactive sheet

```
Start coding or generate with AI.
```