EXPERIMENT 2

Implementation of Multiple Linear Regression, Ridge Regression, and Lasso Regression on Insurance Premium Dataset

1 Dataset Source

Dataset Name: Insurance Premium Prediction Dataset
Source: Kaggle
Link:
https://www.kaggle.com/datasets/noordeen/insurance-premium-prediction/data

The dataset contains medical and demographic information used to predict insurance premium charges.

2 Dataset Description

The Insurance dataset contains personal and medical attributes of individuals and their corresponding insurance charges.

Dataset Characteristics

- Number of Instances: ~1300+ records
- Number of Features: 6 input features + 1 target variable
- Data Type: Mixed (Categorical + Numerical)
- No missing values (in most versions)

Input Features

Feature	Description
age	Age of the individual
sex	Gender (male/female)
bmi	Body Mass Index
children	Number of dependents
smoker	Whether person is a smoker
region	Residential region

Target Variable

Variable	Description
Charges	Insurance premium amount (continuous value)

3 Mathematical Formulation of the Algorithms

A. Multiple Linear Regression

Multiple Linear Regression models the relationship between one dependent variable and multiple independent variables.

Mathematical Equation:

y=β0+β1x1+β2x2+...+βnxn+ϵy = \beta_0 + \beta_1x_1 + \beta_2x_2 + ... + \beta_nx_n + \epsilony=β0+β1x1+β2x2+...+βnxn+ϵ

Where:

- yyy = predicted insurance charge

- β0\beta_0β0 = intercept

- βi\beta_iβi = coefficients

- xix_ixi = independent variables

- ϵ\epsilonϵ = error term

The model minimizes the cost function:

- MSE=1n∑(y−y^)2MSE = \frac{1}{n} \sum (y - \hat{y})^2MSE=n1∑(y−y^)2
- xixi = independent variables
- ϵϵ = error term

The model minimizes:

MSE=1n∑(y−y^)2MSE=n1∑(y−y^)2

B. Ridge Regression

Ridge Regression adds a penalty term to reduce overfitting.

Cost Function:

J(β)=∑(y−y^)2+λ∑β2J(\beta) = \sum (y - \hat{y})^2 + \lambda \sum
\beta^2J(β)=∑(y−y^)2+λ∑β2

Where:

- λ\lambdaλ = regularization parameter

- Penalizes large coefficients

Effect:

- Shrinks coefficients

- Reduces variance

- Handles multicollinearity

C.Lasso Regression

Lasso adds absolute penalty.

Cost Function:
J(β)=∑(y−y^)2+λ∑∣β∣J(\beta) = \sum (y - \hat{y})^2 + \lambda \sum |\beta|J(β)=∑(y−y^)2+λ∑∣β∣

Effect:

- Shrinks coefficients
- Can make some coefficients exactly zero
- Performs feature selection

4 Algorithm Limitations

Multiple Linear Regression Limitations

- Assumes linear relationship
- Sensitive to outliers
- Requires low multicollinearity
- Cannot capture nonlinear relationships

Ridge Regression Limitations

- Does not perform feature selection
- All features remain in model

- Choosing λ value is critical

Lasso Regression Limitations

- Can remove important features if λ is large
- Unstable when features are highly correlated
- Sensitive to scaling

5 Methodology / Workflow

Step 1: Data Loading

- Dataset uploaded to Google Colab
- Loaded using Pandas

Step 2: Data Inspection

- Checked dataset shape
- Verified missing values
- Identified categorical and numerical features

Step 3: Data Preprocessing

- Applied One-Hot Encoding to categorical variables
- Applied StandardScaler to numerical features
- Created preprocessing pipeline

Step 4: Train-Test Split

- 80% Training data
- 20% Testing data
- Used random_state = 42 for reproducibility

Step 5: Model Training

- Trained Multiple Linear Regression
- Trained Ridge Regression
- Trained Lasso Regression

Step 6: Model Evaluation

Evaluation Metrics Used:

- RMSE (Root Mean Squared Error)
- R^2 Score

Workflow Diagram

Dataset
↓
Data Cleaning
↓
Encoding & Scaling
↓
Train-Test Split

↓
Model Training
↓
Prediction
↓
Performance Evaluation

6 Performance Analysis

1)Multiple Linear Regression

Code

```
X = df.drop('expenses', axis=1)

y = df['expenses']

cat_cols = X.select_dtypes(include=['object']).columns

num_cols = X.select_dtypes(exclude=['object']).columns

numeric_transformer = Pipeline(steps=[

    ('scaler', StandardScaler())

])


categorical_transformer = Pipeline(steps=[

    ('encoder', OneHotEncoder(drop='first'))

])


preprocessor = ColumnTransformer(

    transformers=[

        ('num', numeric_transformer, num_cols),
```

```
        ('cat', categorical_transformer, cat_cols)
    ])
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)


linear_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])


linear_model.fit(X_train, y_train)
y_pred_lr = linear_model.predict(X_test)


rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_lr = r2_score(y_test, y_pred_lr)


print("Multiple Linear Regression")
print("RMSE:", rmse_lr)
print("R2 Score:", r2_lr)
```

```
Multiple Linear Regression
RMSE: 5796.556335884077
R2 Score: 0.7835726930039905
```

2)Ridge Regression

Code

```
ridge_model = Pipeline(steps=[
```

```
    ('preprocessor', preprocessor),

    ('regressor', Ridge())

])


ridge_model.fit(X_train, y_train)

y_pred_ridge = ridge_model.predict(X_test)


rmse_ridge = np.sqrt(mean_squared_error(y_test, y_pred_ridge))

r2_ridge = r2_score(y_test, y_pred_ridge)


print("\nRidge Regression")

print("RMSE:", rmse_ridge)

print("R2 Score:", r2_ridge)
```

```
...
    Ridge Regression
    RMSE: 5800.731196221169
    R2 Score: 0.7832608253669844
```

3)Lasso Regression

Code

```
lasso_model = Pipeline(steps=[

    ('preprocessor', preprocessor),

    ('regressor', Lasso())

])


lasso_model.fit(X_train, y_train)

y_pred_lasso = lasso_model.predict(X_test)
```

```
rmse_lasso = np.sqrt(mean_squared_error(y_test, y_pred_lasso))

r2_lasso = r2_score(y_test, y_pred_lasso)


print("\nLasso Regression")

print("RMSE:", rmse_lasso)

print("R2 Score:", r2_lasso)
```

```
...
    Lasso Regression
    RMSE: 5797.315513119226
    R2 Score: 0.7835159981546111
```

7 Hyperparameter Tuning

Ridge Hyperparameter

Parameter tuned:

- `alpha` (λ)

Grid values tested:

- [0.01, 0.1, 1, 10, 100]

Effect

- Optimal alpha improved model generalization
- Reduced overfitting

Lasso Hyperparameter

Parameter tuned:

- `alpha` (λ)

Grid values tested:

- [0.001, 0.01, 0.1, 1, 10]

Best alpha obtained:

Effect:

- Selected optimal number of features
- Balanced bias and variance

(Leave space for tuning output screenshot)

Conclusion

This experiment successfully demonstrated:

- Implementation of Multiple Linear Regression
- Application of Ridge (L2) Regularization
- Application of Lasso (L1) Regularization
- Use of pipelines for preprocessing
- Hyperparameter tuning using GridSearchCV
- Performance comparison of regularized and non-regularized models

Regularization techniques help improve model performance by controlling overfitting and reducing variance.