

PA 1 Report

Soo Jin Jung, Yeji Park (Team 6)

szj5489@psu.edu, ykp5105@psu.edu

Kaggle Account: Team 6: YejiPark0514, Soo_Jin_Jung_

DS 310


Pennsylvania State University

Description of the competition:

In this competition, we leveraged regression methods to predict diabetic condition levels in patients based on three medical datasets provided. The training data included measurements of 242 patients, and we applied this information to generate predictions for a 200-patient test set. Our goal was to generate a 200 x 2 matrix CSV file with one column of Sl. No. (serial numbers) and another column with the predictions (Output) that we needed to obtain. This project gave students a hands-on chance to use machine learning ideas, emphasizing skills in feature selection, model training, and result evaluation in a real-world setting.

Final Rank in Leaderboard:

Our final Rank on the public leaderboard:

11	Team 6	 	3058.35462	86	9m
----	--------	---	------------	----	----

Data Pre-Processing

We were provided with three datasets: `x_train.csv`, `x_test.csv`, and `y_train.csv`. Additionally, a dataset named `samples.csv` was provided for saving our predicted values. Initially, we performed data preprocessing by loading the data into Jupyter Notebook using the `read_csv` function from the pandas package. As the provided datasets were already divided into training and testing sets, we omitted the train/test split step. The `y_train` dataset was provided with two columns: 'Sl. No.' and 'Output'. We wanted our training data to consist only of the target variable, so we extracted only the 'Output' column from the given dataset and saved it as our new `y_train`.

```
import pandas as pd
import numpy as np

X_train = pd.read_csv('x_train.csv')
X_test = pd.read_csv('x_test.csv')
y_train = pd.read_csv('y_train.csv')
samples = pd.read_csv('sampleSubmission.csv')

y_train = y_train['Output']
```

Linear Regression

We started with Linear Regression as our initial model. Using the `LinearRegression` function from the *sklearn.linear_model* package, we created `LR_model` to perform linear regression analysis. Next, we trained `LR_model` by fitting it with *X_train* and *y_train*. Afterward, we evaluated the performance of our linear regression model on the test dataset by making predictions and storing the results in the *y_pred* variable. Upon submitting the predicted *y_pred* values from linear regression to Kaggle, we obtained an MSE value of **4133.78094**.

```
from sklearn.linear_model import LinearRegression

# Create Linear Regression Model
LR_model = LinearRegression()
LR_model.fit(X_train, y_train)

# Make Predictions
y_pred = LR_model.predict(X_test)
```

Lasso and Ridge regression techniques, categorized as regularization methods, incorporate penalty terms into linear regression models to mitigate variance in estimates and enhance predictive accuracy. As a result, Lasso and Ridge regression usually produces lower MSE values compared to conventional linear regression. Unlike these methods, linear regression does not have built-in methods to identify overfitting, like in our case with *y_pred*. Since our `LR_model` was not fine-tuned through feature engineering or hyperparameter adjustment, it was not the best choice for achieving the lowest MSE. Therefore, we moved forward with Lasso and Ridge regression analyses.

Ridge Regression

We tried Ridge Regression for our second model. We used the `RidgeRegression` function from the *sklearn.linear_model* package to establish a model for conducting Ridge regression analysis. Following this, we trained the model by fitting it with *X_train* and *y_train*. Before evaluating the model, we decided to do a hyperparameter tuning.

A key emphasis of using Ridge regression was to refine the model's performance through hyperparameter tuning, specifically targeting the regularization parameter, alpha. Alpha determines the weight of a penalty and finding the best alpha helps prevent models from becoming complex and adapting to the data rather than the patterns in the data given. We systematically tested a range of

alpha values, spanning from 0.001 to 1000, by leveraging the logarithmic scale to strike a balance in model complexity. The logarithmic scale was employed to thoroughly explore a broad range of values, with a particular emphasis on smaller values to identify optimal performance. The grid search was implemented using `np.logspace(-3, 3, 100)`, which allowed us to generate 100 evenly spaced numbers on the logarithmic scale between 0.001 and 1000. Logarithmic spacing is commonly utilized with hyperparameters as it provides a comprehensive exploration across a wide variable of numbers.

We then conducted hyperparameter tuning using the `GridSearchCV` module from the `sklearn.model_selection` package to identify the optimal alpha for our model. With `GridSearchCV`, we specified a `param_grid` dictionary containing a range of alpha values to test. By setting the cross-validation parameter to 5, the function split the data into 5 folds and trained the model multiple times to determine the best parameter value. Afterward, we evaluated the performance of our ridge regression model on the test dataset by making predictions and storing the results in the `y_pred` variable. Upon submitting the predicted `y_pred` values from Ridge regression to Kaggle, we obtained an MSE value of **3180**. The score we obtained from Ridge regression was lower than that from Linear regression. However, we decided to try using Lasso regression to see if we can achieve better results.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

model = Ridge(alpha = 1.0)
model.fit(X_train, y_train)

# Define hyperparameters for grid search
params = {'alpha': np.logspace(-3, 3, 100)}

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=model, param_grid=params, cv=5)
grid_search.fit(X_train, y_train)

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions
y_pred = best_model.predict(X_test)
```

Lasso Regression

Lasso regression was the last regression model we used. We utilized the `LassoRegression` function from the `sklearn.linear_model` package to build a model for conducting Lasso regression analysis. Subsequently, we trained the model by fitting it with X_{train} and y_{train} . Unlike Ridge regression, we decided to use a simple Lasso algorithm without any hyperparameters or tuning. If the result returns unlikely, we planned to use hyperparameter tuning. We tried running the model with different alpha values, including 0.1, 0.05, 0.01, and 0.005. Afterward, we evaluated the performance of our Lasso regression model on the test dataset by making predictions and storing the results in the y_{pred} variable. Upon submitting the predicted y_{pred} values from Lasso regression to Kaggle, we obtained an MSE value of **3058.35462**. This result occurred when we used an alpha value equal to 0.1. Since we got a significantly low MSE value with this model, we decided not to proceed further with hyperparameter tuning.

```
from sklearn.linear_model import Lasso

# Create Lasso Regression Model
model = Lasso(alpha = 0.1)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

Conclusion

We tried running Ridge regression without employing any hyperparameter tuning. However, we found that Lasso regression with no hyperparameter tuning outperformed Ridge regression with no hyperparameter tuning in our analyses. Upon research, we learned that this can be attributed to Lasso regression's inherent feature selection mechanism within the regularization process. Unlike Ridge regression, Lasso has the capability to shrink the coefficients of less important features all the way to zero. As a result, Lasso selects only the most relevant predictors for the model, leading to a more streamlined and interpretable model. In contrast, while Ridge regression may shrink the coefficients, it does not force them to zero. This can result in less sparse solutions, potentially retaining some less important features in the model.

Additionally, we were interested in understanding why Lasso regression performed better than Ridge regression even when we attempted hyperparameter tuning with Ridge regression. When deciding on a feature engineering method, we opted for `GridSearchCV` over `kNN`, as we believed `kNN` might not be suitable for our regularization regression models, Lasso and Ridge. However, upon

closer examination of the GridSearchCV function, we discovered that the recommended optimal alpha value might not be as effective as manually searched alpha values. This discrepancy could be due to the complexity of our dataset, which contains many detailed features. During our analysis, we realized that GridSearchCV might be sensitive to the large number of features in our dataset, which consists of 64 columns. This sensitivity could have caused GridSearchCV to overlook the best hyperparameter values for our model.

In conclusion, the model that performed the best was Lasso regression with an alpha parameter value of 0.1, which resulted in the lowest MSE value of 3058.35462.