

Project #1 Report

Team 8

Apoorv Thite - aat5564@psu.edu

www.kaggle.com/apoorvthite

Brian Nelson - brn5123@psu.edu

www.kaggle.com/bnelly

Nathan Kizlinski - [nj5644@psu.edu](mailto:njk5644@psu.edu),

www.kaggle.com/nathankizlinski4

Overview:

A medical data set is provided under the Data tab along with the target assignment.

The input variables correspond to measurements (physical, physiological, and blood related) for a given patient and the target variable corresponds to the level of diabetic condition in the patient. It contains:

- x train (242×64) and y train (242×1) for training.
- x test (200×64) for testing.

The goal is to use regression methods as outlined in the Project 1 documentation to come up with y test (200×2) i.e. it MUST CONTAIN 2 columns :

Col. 1: The Sl. No. column which corresponds to each row and








Col. 2: Output (which is the predicted output).

Project Description

In this Kaggle competition, the data set we will be using is a medical data set, where the input variables are physical, psychological, and blood related measurements for a patient. The target variable is the level of a patient's diabetic condition. We are instructed to test and implement different regression models in hopes of receiving the smallest Mean Squared Error to correctly predict the patient's diabetic condition. Our

scores are calculated on two different standards, the Kaggle competition score and the individual score.

Project Outcome

6	Team 21	  	3000.14291	11	10m
7	Team 8	  	3024.50336	43	1m
<div><div><p>Your Best Entry!</p><p>Your most recent submission scored 3024.50336, which is an improvement of your previous score of 3047.81066. Great job!</p></div><div>Tweet this</div></div>					

The MSE of our Final Code (Best Attempt) that we submitted on Kaggle was **3024.50** . This MSE was achieved after a lot of manipulations and exploring different regression models to try and maximize the accuracy of our model and minimizing the MSE as much as we could. By the time of submitting, this MSE got us to the 7th position on the public leaderboard. The detailed analysis below explains how our team was able to achieve this score and the pre-processing, hypertuning techniques that we used, in addition to selecting the best model for the given data.

Pre Processing

With this dataset, there wasn't much hands-on preprocessing that we had to do, for example there were no outliers, missing data values, etc. However, we were still able to examine the dataset before any manipulation. We first loaded the datasets, then viewed them using the head function and also returned the whole data. We were able to look at the columns, rows, what variables were being used, and more to further understand what we were working with. We noticed that the y_train data table had two columns, the serial number and the output (which was needed for final predictions). As mentioned before, there weren't any missing data points, so no such preprocessing was needed. With the data cleaned, we were then able to move on to calculations.

```
[3] # Preview the data
print(x_train.head())
```

	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	\
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	
2	0.005299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	
3	-0.009063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	


```
[5] print(y_train.head())
```

	Sl. No.	Output
0	1	151
1	2	75
2	3	141
3	4	206
4	5	135

```
[5 rows x 64 columns]
```

Feature Engineering

As a team, we employed feature engineering techniques to optimize the performance of our predictive model. One of the key techniques we used was normalization through the **MinMaxScaler**. This technique transformed all features in the dataset to a uniform scale, which is crucial when working with models like LassoLarsCV, where feature magnitude can impact the model's regularization behavior. By scaling the features to a consistent range, we ensured that no single feature would disproportionately influence the model, allowing for more accurate predictions. This approach helped us reduce the Mean Squared Error (MSE) and improve the overall performance of our regression model.

```
[13] # Initialize the scaler
scaler = MinMaxScaler()

# Fit and transform the data
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Model Building/Comparison

In our model building and comparison process, we explored multiple regression models to identify the one that would yield the best performance on our dataset. We initially employed traditional models such as **Linear Regression** and **Decision Tree Regressor** to establish baseline performances. We then progressed to more complex models, including **Random Forest Regressor**, to capture non-linear relationships and interactions between features. Furthermore, we incorporated **LassoLarsCV**, a variation of Lasso Regression with built-in cross-validation, which allowed us to automatically tune the regularization strength (alpha) to prevent overfitting. Each model's performance was evaluated using Mean Squared Error (MSE)

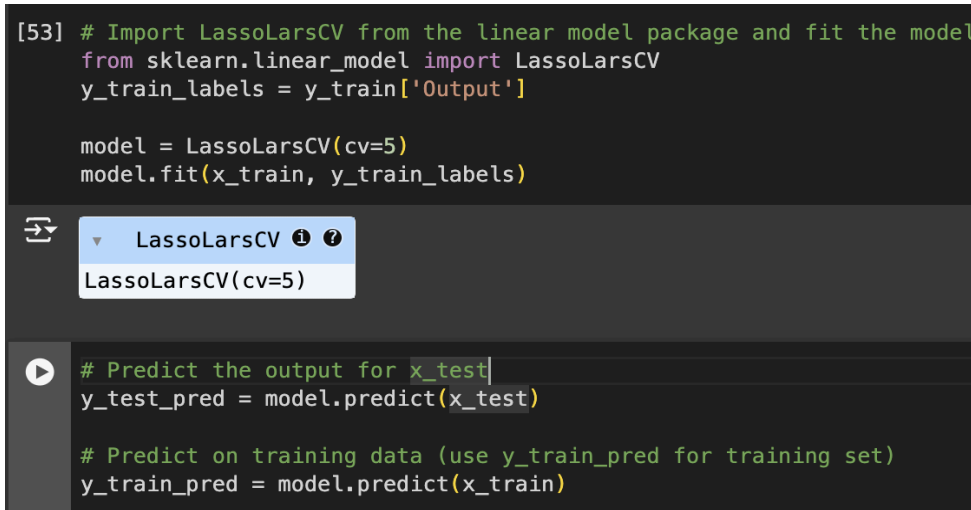
as the metric, and we compared results across training and test datasets. Through this comparison, we were able to identify the model that best balanced prediction accuracy and generalizability, ultimately selecting LassoLarsCV due to its superior performance after normalization and cross-validation.

```
[53] # Import LassoLarsCV from the linear model package and fit the model
      from sklearn.linear_model import LassoLarsCV
      y_train_labels = y_train['Output']

      model = LassoLarsCV(cv=5)
      model.fit(x_train, y_train_labels)

      # Predict the output for x_test
      y_test_pred = model.predict(x_test)

      # Predict on training data (use y_train_pred for training set)
      y_train_pred = model.predict(x_train)
```



Hyperparameter Setting/Tuning

Cross-Validation (cv=5): In the LassoLarsCV model, we utilized 5-fold cross-validation. This ensured that the model is evaluated on different subsets of the data to prevent overfitting and to find the best regularization parameter (alpha). Cross-validation divided the data into 5 parts, trained the model on 4 parts, and tested on the 5th part iteratively, helping to ensure the model generalizes well to unseen data.

Normalization: Before fitting the model, we used **MinMaxScaler** to normalize the feature set. This ensured that all features are on the same scale, which is crucial when using Lasso-based models since they are sensitive to feature magnitudes.

Performance Evaluation

For this project we used the MSE (mean squared error) to evaluate how well our models worked. We decided that it would be best to calculate the mse for the y_training set and the predicted y for the training set based on the model we used. We did this because we thought that the lower mean squared error that we got would tell us that our model is working well. This

would mean that our Y would be stronger compared to a y test with a high mse. Here are some examples of MSE's that we got:

```
mse=mean_squared_error(y_train['Output'],y_train_pred)
```

```
[117] mse
```

```
2187.1602949306084
```

```
# Calculate the MSE for the training set
mse_train = mean_squared_error(y_train_labels, y_train_pred)
print(f'MSE on Training Set: {mse_train}')
```

```
MSE on Training Set: 484.605388016529
```

```
[In [ ]]: # Calculate the MSE for the training set
mse_train = mean_squared_error(y_train_labels, y_train_pred)
print(f'MSE on Training Set: {mse_train}')
```

```
MSE on Training Set: 2634.7747015141335
```

Mean squared error is the sum of the distance from the predicted y value and the actual y value squared. A MSE of 0 would mean the model is perfect and higher mse's mean your predicted values are further away. In the process of this project we tried different models to try and find the smallest mse. We even got an MSE as low as 484 with the random forest model.

Group Performance: As a group we worked very well and efficiently. On the first day the project was introduced we made a group chat to communicate with each other and set up meeting times. Throughout the project we met multiple times to discuss and plan. Our overall plan was that we would kind of independently write our code (with help when needed) then come together and compare which of our models worked. This worked very well because it allowed us to try as many models as possible while tuning each other's code as we went. Overall we were a very efficient and in-the-loop group where we all gave solid effort and were willing to overcome problems. At the end we all came together to finish the project report as a group together.

Novel Ideas

The most interesting result that we did not expect in this project is how the MSE's of the training data did not fully correlate with the MSE's of the test data on Kaggle. This was surprising to us because we thought that if we got a really good MSE with the training data then we must have a very strong model. Therefore, we expected that our final MSE would also be very strong, which was not always the case. For example, the best model we scored on colab had a training set MSE of 2805, while the MSE on Kaggle was 3024. It is important to note that the lowest training MSE we got was 484, but this model did not perform the best on Kaggle.

Lessons Learnt

Overall, this project truly stretched our abilities as data scientists. We think that this project really taught us how much of data science is collaborative, and working by yourself would be much harder. We also learned that communication is key, we were constantly in communication with each other which expedited the project very much. In this project there were many regression models that we used, and each regression model was unique. This required us to keep in constant contact to keep track of what we have tried and what we still need to try. If there was ever a problem, the other two group members were there to help. Finally, we learned that sometimes things are not as simple as they may seem. We had to plan and test much more than we were originally expecting, however this experience was very good for us.