Project 1 Summary

Dataset 1: Auto MPG

The AutoMPG data set was a given dataset in this project. It was taken from the ScalaTion code base. The shape of the original dataset was 392x8. However, the origin column was dropped because it was categorical. The data was also checked for any columns that had perfect collinearity. While cylinders, displacement, horsepower, and weight had fairly high collinearity, no columns in the dataset were perfectly collinear so the final input features were the following: cylinders, displacement, horsepower, weight, acceleration, and model_year. The target value in this case was mpg.

After running all required models on the AutoMPG dataset, we found that the symbolic ridge regression yielded the best in Scala with 5 input features. It produced an R^2 bar of 0.860607, a R^2 cv of 0.858435, AIC of -179.69, and BIC of -163.19. However, the best result in Python was with symbolic regression. We found that the best number of features was 2 (horsepower and cylinders) that produced an R2 bar of 0.847902, a R2 cv of 0.847512, an AIC of 173.73, and a BIC of 178.46. Overall, based on our results the best model for this dataset is symbolic ridge regression.

Dataset 2: Forest Fires

The Forest Fires dataset was a given data set in this project. It was taken from the UCI Machine Learning Repository. The shape of the original dataset was 517x13. This dataset had a large number of 0 values in the target column area. In an effort to smooth the data and make it more linear, we performed a log transform on the area column making a new column called log_area. We then dropped the original area column so the data retained its original shape of 517x13. The data was also checked for any columns that had perfect collinearity. No columns in the dataset were perfectly collinear and there was low collinearity in general between the columns. The final input features were the following: X, Y, month, day, FFMC, DMC, DC, ISI, temp, RH, wind, and rain. The target value in this case was log_area.

After running all required models on the Forest Fires dataset, we found that none of the models really performed well at all. However, cubic X regression yielded the best in Scala with 4 input features. It produced an R2 bar of 0.078917, a R2 cv of 0.071721, AIC of -167.65, and BIC of -154.94. The best result in Python was with linear regression. We found that the best number of features was 3 (DMC, day, and DC) that produced an R^2 bar of 0.008740, a R^2 cv of 0.004883, an AIC of 73.27, and a BIC of 81.18. Overall, based on our results no model performed well, but best results came from cubic X regression.

Dataset 3: Air Quality

The Air Quality dataset was a given data set in this project. It was taken from the UCI Machine Learning Repository. The shape of the original dataset was 9357x15. First, we dropped the Date and Time columns and replaced them with an index value to correspond to the date and time. This allowed us to represent Date and Time numerically instead of categorically. Upon inspection of the data, we found that there were a large number of values missing (denoted as -200). We found the percentage of data missing by column and any columns that had over 10% missing values were discarded. For the remaining columns, all rows with missing values were dropped. The final shape of the data after cleaning was 8991x10. The data was also checked for any columns that had perfect collinearity. While there was high collinearity between several of the features, none were perfectly collinear.The final input features were the following: Time, PT08.S1(CO), C6H6(GT), PT08.S2(NMHC), PT08.S3(NOx), PT08.S5(O3), T, RH, and AH. The target value in this case was PT08.S4(NO2).

After running all required models on the Air Quality dataset cubic X regression yielded the best in Scala with 5 input features. It produced an R^2 bar of 0.975015, a R^2 cv of 0.975001, AIC of -9728, and BIC of -9695.03. The best result in Python was with quadratic X regression. We found that the best number of features was 12 and that produced an R^2 bar of 0.968809, a R^2 cv of 0.968848, an AIC of 14800.27, and a BIC of 14800.27. Overall, all of the models performed well, but our best results came from cubic X regression.

Dataset 4: CCPP

The CCPP dataset was a retrieved data set in this project. It was taken from the UCI Machine Learning Repository. The shape of the original dataset was 9568x5. There was no preprocessing necessary for this because there were no missing/null values, nor any non-ordinal/numeric values. The data was also checked for any columns that had perfect collinearity. No columns in the dataset were perfectly collinear. The input features for the dataset were AT (temperature), V (exhaust vacuum), AP (ambient pressure), and RH(relative humidity). The target variable was PE (power plant electrical output).

After running all required models on the CCPP dataset, we found that most of the models performed fairly well with a minimum R^2 cv of 0.8036. Ultimately, Symbolic regression yielded the best in Scala with all 4 input features included. It produced an R^2 bar of 0.9397, a R^2 cv of 0.9396, AIC of -5399.97, and BIC of -5277.73. The best result in Python was with Quadratic X regression. We found that the best number of features was 10 (which included several altered variables from the original data) that produced an R^2 bar of 0.9374, a R^2 cv of 0.9373, an AIC of 5571.97, and a BIC of 5627.54. Overall, based on our results all of the models performed well, but best results came from Symbolic regression through Scala.

Dataset 5: Bike Sharing

The Bike Sharing dataset was a chosen data set in this project. It was taken from the UCI Machine Learning Repository. The shape of the original dataset was 17379x15. Due to no

missing or null values, nor any non-ordinal/numeric values, no preprocessing was necessary. There were 2 variables with relatively high collinearity, such as month and season, and atemp (perceived temperature) and temp (actual temperature) were almost perfectly collinear. As such, atemp was dropped. The target value in this case was cnt, or the count of total rental bikes that hour.

After running all required models on the bike sharing dataset, we found that the symbolic regression yielded the best in Scala with 4 input features. It produced an R^2 bar of 0.287, a R^2 cv of 0.286, AIC of -1097.22, and BIC of -1072.78. However, the best result in Python was with cubic X regression. We found that the best number of features was 220 that produced an R^2 bar of 0.629211, a R^2 cv of 0.633884, an AIC of 33093.65, and a BIC of 34447.44. Overall, based on our results the best model for this dataset is cubic X regression.

Dataset 6: Wine Quality

Within this wine quality data set, there are two datasets included, related to red and white vinho verde wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests using a variety of regression models. The dataset is pulled from UCI Machine Learning Repository. There are 11 inputs (fixed acidity, volatile acidity, citric acid, etc.) and the output is based on sensory data made by wine experts . The quality ranges from 0 (very bad) and 10 (very excellent). Due to the fact that white wine had more instances than red wine, we decided to go with the white wine data set (4898 instances).There are no missing attribute values.

After running all the required models on the wine quality dataset, we found that the Symbolic Regression yielded the best results in Scala with 4 input features. The models produced an R^2 bar value of 0.287, a R^2 cross-validation value of 0.287, an Akaike information criterion value of -1097.22, and a Bayesian information criterion value of -1072.78. On the other hand, the best model produced by Python is quadratic x regression. The best number of features was 15 features that produced an R^2 bar value of 0.266, an R^2 cv value of 0.268, an AIC value of -515.90, and a BIC value of -442.59. Overall, we found that symbolic regression is the best model.

Model Parameters Explained

sklearn.linear_model:

- LinearRegression
  - **fit_intercept:*bool, default=True***
    - Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).
    - Since our data is not expected to be centered, we keep this value True
  - **normalize:bool** - DEPRECIATED, ignored by model
  - **copy_X:*bool, default=True***
    - If True, X will be copied; else, it may be overwritten.
    - Kept True because we want to preserve X to be used for other models
  - **n_jobs:*int, default=None***
    - The number of jobs to use for the computation. This will only provide speedup in case of sufficiently large problems, that is if firstly `n_targets > 1` and secondly `X` is sparse or if `positive` is set to `True`. `None` means 1 unless in a **`joblib.parallel_backend`** context. `-1` means using all processors.
    - Left at None due to CPU processor power varying per group member as well as a general lack of need of it due to fast run times
  - **positive:*bool, default=False***
    - When set to `True`, forces the coefficients to be positive. This option is only supported for dense arrays.
    - There is no reason to force this with this dataset, so it is kept default
- Ridge
  - **alpha:*{float, ndarray of shape (n_targets,)}, default=1.0***
    - Regularization strength. Larger values specify stronger regularization.
    - We tested with a small range of alphas and found that for the most part 1.0 offered the best results, so we kept it universally
  - **fit_intercept: *bool, default=True***
    - Same meaning, value, and reasoning as LinearRegression's fit_interecept
  - **normalize:*bool, default=False***
    - DEPRECIATED, ignored by model
  - **copy_X:*bool, default=True***

        - Same meaning, value, and reasoning as LinearRegression's copy_X
    - **max_iter:*int, default=None***
        - Maximum number of iterations for conjugate gradient solver.
        - Since we aren't using a cg solver, has no use, so is ignored
    - **tol:*float, default=1e-3***
        - Precision of the solution.
        - Through a brief test we determined this is the best value generally.
    - **solver*{'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga', 'lbfgs'}, default='auto'***
        - Solver to use in the computational routines. 'auto' chooses the solver automatically based on the type of data.
        - Given that the default is capable of determining the best solver for the type of data passed, we thought it would be best to keep this default.
    - **positive:*bool, default=False***
        - Same meaning, value, and reasoning as LinearRegression's positive
    - **random_state:*int, RandomState instance, default=None***
        - Used when `solver` == 'sag' or 'saga' to shuffle the data
        - Since we aren't using the sag or saga solver, this is ignored
- Lasso
    - **alpha:*float, default=1.0***
        - Constant that multiplies the L1 term. `alpha = 0` is equivalent to an ordinary least square, solved by the **`LinearRegression`** object.
        - We tested with a small range of alphas and found that for the most part 1.0 offered the best results, so we kept it universally
    - **fit_intercept: *bool, default=True***
        - Same meaning, value, and reasoning as LinearRegression's fit_interecept
    - **normalize:*bool, default=False***
        - DEPRECIATED, ignored by model
    - **precompute:*bool or array-like of shape (n_features, n_features), default=False***
        - Whether to use a precomputed Gram matrix to speed up calculations.
        - Since speeding up calculations is not needed here this is left as False
    - **copy_X:*bool, default=True***
        - Same meaning, value, and reasoning as LinearRegression's copy_X
    - **max_iter*int, default=1000***

        - The maximum number of iterations.
        - Didn't seem to have much of an impact on results so it was left alone
    - **tol:*float, default=1e-4***
        - The tolerance for the optimization: if the updates are smaller than `tol`, the optimization code checks the dual gap for optimality and continues until it is smaller than `tol`.
        - Through a brief test we determined this is the best value generally.
    - **warm_start:*bool, default=False***
        - When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.
        - Since reusing previous calls to fit would ruin cross validation, this is kept False
    - **positive:*bool, default=False***
        - Same meaning, value, and reasoning as LinearRegression's positive
    - **random_state:*int, RandomState instance, default=None***
        - The seed of the pseudo random number generator that selects a random feature to update. Used when `selection` == 'random'.
        - Since selection = cyclic for us this does not apply and is ignored
    - **selection:*{'cyclic', 'random'}, default='cyclic'***
        - If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default. This (setting to 'random') often leads to significantly faster convergence especially when tol is higher than 1e-4.
        - There is no need for a faster convergence time, so it is kept as cyclic

gplearn

- SymbolicRegressor
    - Everything was kept default because the defaults are closest to how it is implemented in scala