Mercedes-Benz Greener Manufacturing

March 15, 2021

0.0.1 Project 1 - Mercedes-Benz Greener Manufacturing

DESCRIPTION Reduce the time a Mercedes-Benz spends on the test bench.

0.0.2 Problem Statement Scenario:

Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include the passenger safety cell with the crumple zone, the airbag, and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium carmakers. Mercedes-Benz cars are leaders in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams.

To ensure the safety and reliability of every unique car configuration before they hit the road, Daimler's engineers have developed a robust testing system. As one of the world's biggest manufacturers of premium cars, safety and efficiency are paramount on Daimler's production lines. However, optimizing the speed of their testing system for many possible feature combinations is complex and time-consuming without a powerful algorithmic approach.

You are required to reduce the time that cars spend on the test bench. Others will work with a dataset representing different permutations of features in a Mercedes-Benz car to predict the time it takes to pass testing. Optimal algorithms will contribute to faster testing, resulting in lower carbon dioxide emissions without reducing Daimler's standards.

0.0.3 Requirements Analysis

0.0.4 Problem Summary

1. **GIVEN** we need to reduce the time that cars spend on the test bench
2. **WHEN** we apply Singular Value Decomposition to the dataset representing different permutations of features in a Mercedes-Benz car
3. **THEN** we can contribute to faster testing, resulting in lower carbon dioxide emissions without reducing Daimler's standards.

Business Objectives and Constraints

- Predicting accurate time a car spends on the test bench
- No strict latency constraints, few seconds to a few minutes prediction time is okay, but not hours.

0.1 Step 1 : Load the Datasets

0.1.1 DataSet Overview

This dataset contains an anonymized set of variables, each representing a custom feature in a Mercedes car. For example, a variable could be 4WD, added air suspension, or a head-up display. The ground truth is labeled 'y' and represents the time (in seconds) that the car took to pass testing for each variable.

We have two comma-separated files: - **train.csv** — Contains the training set with 4209 rows (data points) and 378 columns (features) with labels - Here we can see there are 4209 datapoints indexing from 0 to 4208 and 378 columns/features. - We have three types of data in the train.csv dataset: > - float64(1): Dependent feature, testing time in seconds > - int64(369): Independent Binary features > - object(8): Independent Categorical features - **test.csv** — Contains the test set with 4209 rows (data points) and 377 columns (features) with no labels - Here we can see there are 4209 datapoints indexing from 0 to 4208 and 378 columns/features. - We have two types of data in the test.csv dataset: > - int64(369): Independent Binary features > - object(8): Independent Categorical features

We can see here we have the same number of data points in the train and test dataset.

[1]:
```python
import numpy as np
import pandas as pd
df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
```

[2]:
```python
df_train.head()
```

[2]:
```
   ID       y  X0 X1  X2 X3 X4 X5 X6 X8  …  X375  X376  X377  X378  X379  \
0   0  130.81   k  v  at  a  d  u  j  o  …     0     0     1     0     0
1   6   88.53   k  t  av  e  d  y  l  o  …     1     0     0     0     0
2   7   76.26  az  w   n  c  d  x  j  x  …     0     0     0     0     0
3   9   80.62  az  t   n  f  d  x  l  e  …     0     0     0     0     0
4  13   78.02  az  v   n  f  d  h  d  n  …     0     0     0     0     0

   X380  X382  X383  X384  X385
0     0     0     0     0     0
1     0     0     0     0     0
2     0     1     0     0     0
3     0     0     0     0     0
4     0     0     0     0     0

[5 rows x 378 columns]
```

[3]:
```python
df_test.head()
```

[3]:
```
   ID  X0 X1  X2 X3 X4 X5 X6 X8  X10  …  X375  X376  X377  X378  X379  X380  \
0   1  az  v   n  f  d  t  a  w    0  …     0     0     0     1     0     0
```

```
1   2   t  b  ai  a  d  b  g  y    0  ...     0     0     1     0     0     0
2   3  az  v  as  f  d  a  j  j    0  ...     0     0     0     1     0     0
3   4  az  l   n  f  d  z  l  n    0  ...     0     0     0     1     0     0
4   5   w  s  as  c  d  y  i  m    0  ...     1     0     0     0     0     0

   X382  X383  X384  X385
0     0     0     0     0
1     0     0     0     0
2     0     0     0     0
3     0     0     0     0
4     0     0     0     0

[5 rows x 377 columns]
```

[4]:
```python
df_train.shape, df_test.shape
```

[4]: ((4209, 378), (4209, 377))

0.2 Step 2: Examine the data (for possible outliers)

- Before we do anything else, determine if there are any NaN values Or other irregularities

[5]:
```python
hasNans = df_train.isnull().sum().any()
hasDups = df_train.duplicated().sum().any()
hasDupIDs = df_train['ID'].duplicated().sum()>0
hasNans, hasDups, hasDupIDs
```

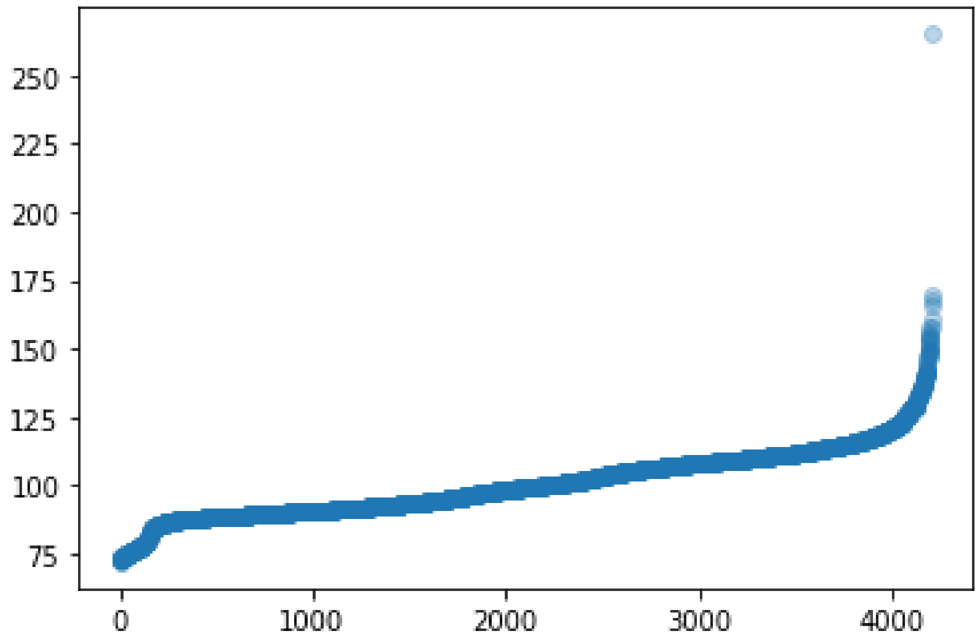
[5]: (False, False, False)

[6]:
```python
hasNans = df_test.isnull().sum().any()
hasDups = df_test.duplicated().sum().any()
hasDupIDs = df_test['ID'].duplicated().sum()>0
hasNans, hasDups, hasDupIDs
```

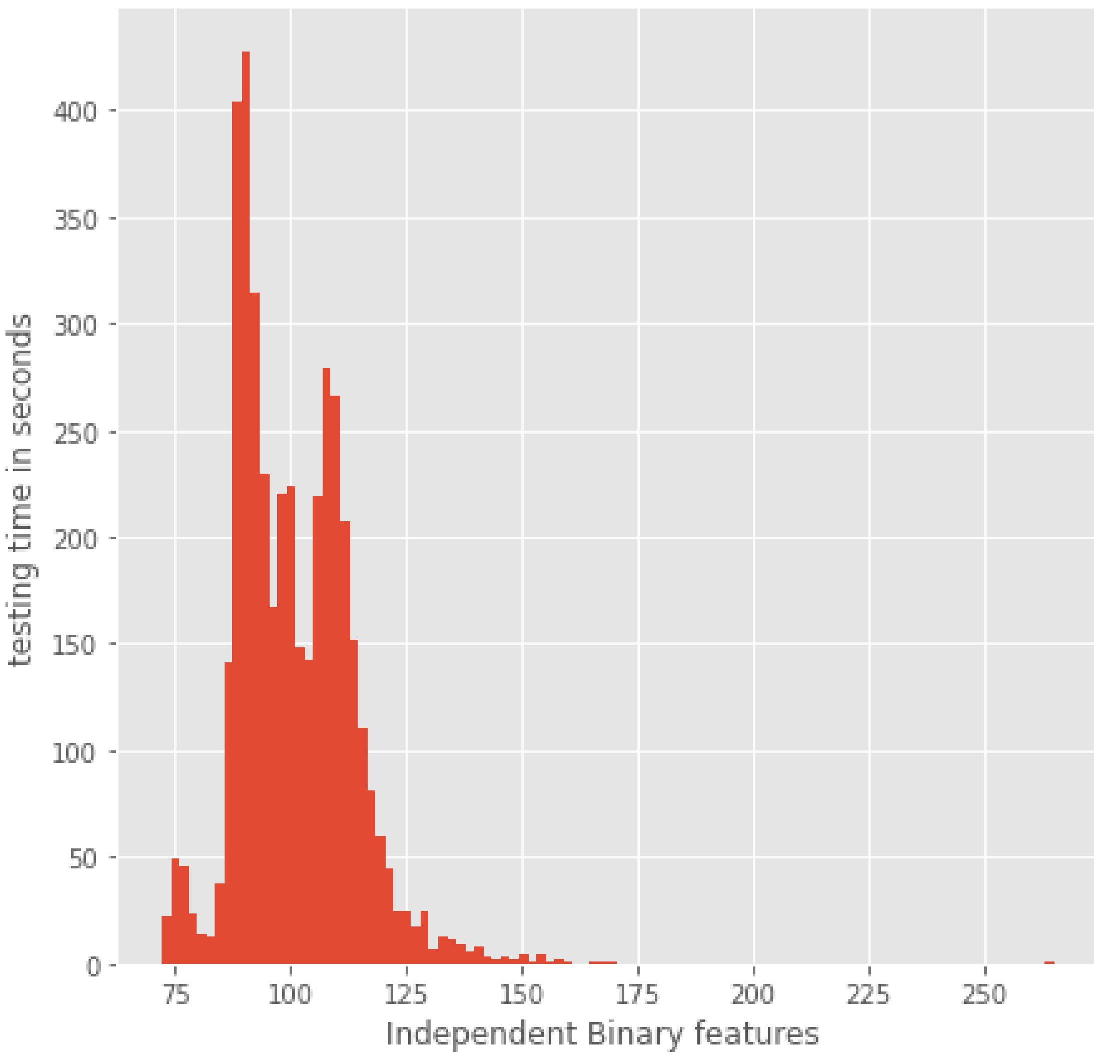
[6]: (False, False, False)

[7]:
```python
import matplotlib.pyplot as plt
from matplotlib import style
%matplotlib inline
plt.scatter(range(len(df_train)), np.sort(df_train.y.values), alpha=0.3)
```

[7]: <matplotlib.collections.PathCollection at 0x7fad884cbfa0>



[8]:
```
#plot histogram
style.use('ggplot')
plt.figure(figsize=(7,7))
plt.hist(df_train.y,bins=100)
plt.xlabel('Independent Binary features ')
plt.ylabel('testing time in seconds')
plt.show()
```

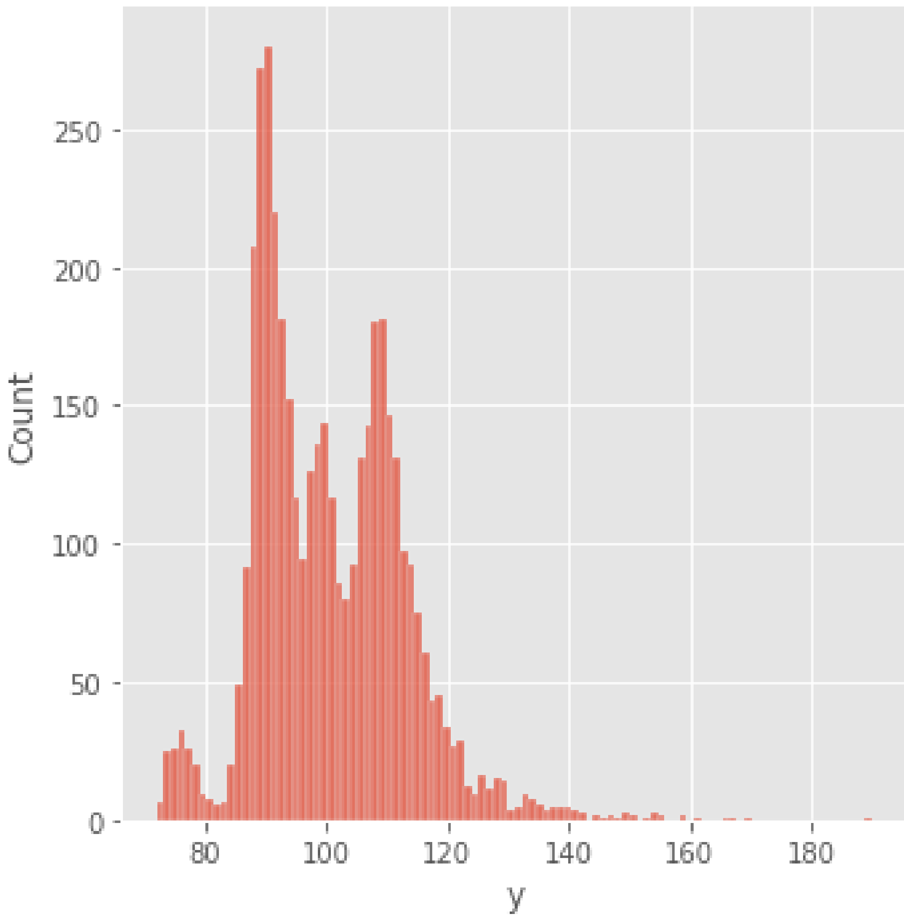


Now remove the outliers values for the purposes of our model.

[9]:
```
ulim = 190
df_train.loc[df_train.y>ulim, 'y'] = ulim
```

[10]:
```
import seaborn as sns
sns.displot(df_train.y, bins=100, kde=False)
```

[10]: <seaborn.axisgrid.FacetGrid at 0x7fad78faeb20>



0.3 Step 3: Setup the X_train, X_test, y_train, y_test parameters for the models.

Identify any features from the train dataframe that are not part of the test dataframe

[11]:
```
list(set(df_train.columns) - set(df_test.columns))
```

[11]: ['y']

- We conclude that the 'y' column of the train dataframe must be the item to be predicted.
- In effect, the 'y' column is the time, (in seconds) that is required to test a feature of a car on the assembly line.

[12]:
```
y_train = df_train.y
y_train.head()
```

[12]:
```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

Now remove the 'y' column from the train dataframe so that the train dataframe can be used to train the models.

[13]:
```
X_train = df_train.drop(['y', 'ID'], axis = 1)
X_train.head()
```

[13]:
```
   X0 X1  X2 X3 X4 X5 X6 X8  X10  X11  …  X375  X376  X377  X378  X379  \
0   k  v  at  a  d  u  j  o    0    0  …     0     0     1     0     0
1   k  t  av  e  d  y  l  o    0    0  …     1     0     0     0     0
2  az  w   n  c  d  x  j  x    0    0  …     0     0     0     0     0
3  az  t   n  f  d  x  l  e    0    0  …     0     0     0     0     0
4  az  v   n  f  d  h  d  n    0    0  …     0     0     0     0     0

   X380  X382  X383  X384  X385
0     0     0     0     0     0
1     0     0     0     0     0
2     0     1     0     0     0
3     0     0     0     0     0
4     0     0     0     0     0

[5 rows x 376 columns]
```

- We should use X_train and y_train from here
- Now what types of data are in the dataset?

[14]:
```
print(X_train.dtypes.value_counts())
print(df_test.dtypes.value_counts())
```

```
int64     368
object      8
dtype: int64
int64     369
object      8
dtype: int64
```

- Now remove any variables that just have a single unique value
- (as these do not add value to the model)

[15]:
```
# check how many columns have a single unique value
X_train_column_unique_values = X_train.apply(lambda x: pd.Series.nunique(x))

# get the names of the columns to be removed
```

```
X_train_columns_remove =␣
↪X_train_column_unique_values[X_train_column_unique_values == 1].index.
↪tolist()

# drop these columns from the dataframe
X_train = X_train.drop(X_train_columns_remove, axis = 1)
X_test = df_test.drop(X_train_columns_remove + ['ID'], axis = 1)

print("New shape of the train dataframe:", X_train.shape)
print("New shape of the train dataframe:", X_test.shape)
```

```
New shape of the train dataframe: (4209, 364)
New shape of the train dataframe: (4209, 364)
```

- Now remove any correlated variables

[16]:
```
correlations = X_train.corr()
```

[17]:
```
def remove_correlated(correlation_df):
    # list of all columns
    all_columns = correlation_df.columns
    chosen_columns = []
    removed_columns = []

    while len(all_columns) > 0:

        # choose the first column in the list
        col = all_columns[0]

        # add it to the chosen columns list
        chosen_columns.append(col)

        # set criteria to filter variables
        criteria = abs(correlation_df[col]) >= 0.6

        # get correlated variables except for the variable itself
        correlated_columns = list(set(correlation_df.loc[criteria, col].index)␣
↪- set([col]))

        # reduce the overall columns to check
        all_columns = list(set(all_columns) - set(correlated_columns + [col]))

        # add columns to be removed in removed columns
        removed_columns.append(correlated_columns)

        # filter out removed variable from the correlation_df
        correlations_df = correlation_df[all_columns]
```

```
    return chosen_columns
```

[18]:
```
chosen_columns = remove_correlated(correlations)
```

[19]:
```
X_train_final = X_train[chosen_columns]
X_train_final, X_test_final = X_train_final.align(X_test, join = 'inner', axis␣
↪= 1)

print("Shape of train data:", X_train_final.shape)
print("Shape of test data:", X_test_final.shape)
```

```
Shape of train data: (4209, 185)
Shape of test data: (4209, 185)
```

[20]:
```
# store feature names
feature_names = X_train_final.columns.tolist()

# convert to arrays
X_train_array = np.array(X_train_final)
y_train_array = np.array(y_train)
test_array = np.array(X_test_final)

from sklearn.model_selection import train_test_split
X_train_train, X_train_val, y_train_train, y_train_val =␣
↪train_test_split(X_train_array, y_train_array, test_size = 0.2,                                                    ␣
↪random_state = 42)

print(X_train_train.shape)
print(X_train_val.shape)
print(y_train_train.shape)
print(y_train_val.shape)
```

```
(3367, 185)
(842, 185)
(3367,)
(842,)
```

0.4 Step 4: Model Building

0.4.1 `sklearn.linear_model`: Lasso

- Linear Model trained with L1 prior as regularizer (aka the Lasso)
- The optimization objective for Lasso is:
- `(1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1`
- Technically the Lasso model is optimizing the same objective function as the Elastic Net with `l1_ratio=1.0` (no L2 penalty).

[21]:
```
from sklearn.linear_model import LassoCV
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score

# define the lasso cv model
cv_model = LassoCV(alphas = None, cv = 5, max_iter = 10000, random_state = 23)
cv_model.fit(X_train_train, y_train_train)
best_alpha = cv_model.alpha_

lasso_model = Lasso(alpha = best_alpha)
lasso_model.fit(X_train_train, y_train_train)

prediction = lasso_model.predict(X_train_val)
Lasso_score = r2_score(y_train_val, prediction)
Lasso_score
```

[21]:
```
0.5680120978533502
```

[22]:
```
lasso_model = Lasso(alpha = best_alpha)
lasso_model.fit(X_train_array, y_train_array)
prediction = lasso_model.predict(test_array)
output = pd.DataFrame({'ID': df_test['ID'],'y': prediction})
output.to_csv('sub_lasso_final.csv', index = False)
```

0.5 Numerical Data Selection

- **GIVEN** That only numerical data can be used by the models
- **WHEN** we select only the numerical columns
- **THEN** we can use the linear models with the datasets

[23]:
```
X_train.head()
```

[23]:
```
   X0 X1  X2 X3 X4 X5 X6 X8  X10  X12  …  X375  X376  X377  X378  X379  \
0   k  v  at  a  d  u  j  o    0    0  …     0     0     1     0     0
1   k  t  av  e  d  y  l  o    0    0  …     1     0     0     0     0
2  az  w   n  c  d  x  j  x    0    0  …     0     0     0     0     0
3  az  t   n  f  d  x  l  e    0    0  …     0     0     0     0     0
4  az  v   n  f  d  h  d  n    0    0  …     0     0     0     0     0

   X380  X382  X383  X384  X385
0     0     0     0     0     0
1     0     0     0     0     0
2     0     1     0     0     0
3     0     0     0     0     0
4     0     0     0     0     0

[5 rows x 364 columns]
```

[24]:
```
y_train
```

[24]:
```
0       130.81
1        88.53
2        76.26
3        80.62
4        78.02
         …
4204    107.39
4205    108.77
4206    109.22
4207     87.48
4208    110.85
Name: y, Length: 4209, dtype: float64
```

[25]:
```
usable_columns = list(set(df_train.columns) - set(['ID', 'y','X0', 'X1', 'X2',␣
↪'X3', 'X4', 'X5', 'X6', 'X8' ]))
_y_train = df_train['y'].values
_id_test = df_test['ID'].values
_x_train = df_train[usable_columns]
_x_test = df_test[usable_columns]
```

[26]:
```
usable_columns = list(set(df_train.columns) - set(['ID', 'y','X0', 'X1', 'X2',␣
↪'X3', 'X4', 'X5', 'X6', 'X8' ]))
_y_train = y_train.values
_id_test = df_test['ID'].values
_x_train = _x_train[usable_columns]
_x_test = df_test[usable_columns]
```

[27]:
```
pd.set_option('mode.chained_assignment', None)
for column in usable_columns:
    cardinality = len(np.unique(_x_train[column]))
    if cardinality == 1:
        _x_train.drop(column, axis=1)
        _x_test.drop(column, axis=1)
    if cardinality > 2:
        mapper = lambda x: sum([ord(digit) for digit in x])
        _x_train[column] = _x_train[column].apply(mapper)
        _x_test[column] = _x_test[column].apply(mapper)
```

[28]:
```
X_train = _x_train
y_train = _y_train
```

0.5.1 sklearn.model_selection: train_test_split

- Split arrays or matrices into random train and test subsets
- Quick utility that wraps input validation and `next(ShuffleSplit().split(X,`

y)) and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.
- Read more in the User Guide.

[29]:
```
# store feature names
feature_names = X_train.columns.tolist()

# convert to arrays
X_train_array = np.array(X_train)
y_train_array = np.array(y_train)
test_array = np.array(X_test)

from sklearn.model_selection import train_test_split
X_train_train, X_train_val, y_train_train, y_train_val = train_test_split(
    X_train_array, y_train_array, test_size = 0.2,
        random_state = 42)

print(X_train_train.shape)
print(X_train_val.shape)
print(y_train_train.shape)
print(y_train_val.shape)
```

```
(3367, 368)
(842, 368)
(3367,)
(842,)
```

[30]:
```
test_array = _x_train.values
```

0.5.2 `sklearn.linear_model`: Ridge

- Linear least squares with l2 regularization.
- Minimizes the objective function:
- `||y - Xw||^2_2 + alpha * ||w||^2_2`
- This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape (n_samples, n_targets)).

[31]:
```
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import Ridge

# define the lasso cv model
cv_model = RidgeCV(alphas = 10**np.linspace(10,-6,100)*0.5, cv = 5)
cv_model.fit(X_train_train, y_train_train)
best_alpha = cv_model.alpha_
```

```python
# fit on entire training data
ridge_model = Ridge(alpha = best_alpha)
ridge_model.fit(X_train_train, y_train_train)

# get estimate on X_val
prediction = ridge_model.predict(X_train_val)
Ridge_score = r2_score(y_train_val, prediction)
Ridge_score
```

[31]: 0.5731494510022381

[48]:
```python
ridge_model = Ridge(alpha = best_alpha)
ridge_model.fit(X_train_array, y_train_array)
prediction = ridge_model.predict(test_array)
output = pd.DataFrame({'ID': df_test['ID'], 'y': prediction})
output.to_csv('sub_ridge_final.csv', index = False)
```

0.5.3 sklearn.model_selection: KFold

- LK-Folds cross-validator
- Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).
- Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

[49]:
```python
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline

# hidden units = (100,), batch_size = 25, learning_rate = 0.00001, max_iter =␣
↪1000, score = 0.504

X_train_train, X_train_val, y_train_train, y_train_val = train_test_split(
    X_train_array, y_train_array, test_size = 0.2,
    random_state = 42)

mlp_model = MLPRegressor(activation = 'identity', solver = 'sgd', learning_rate␣
↪= 'constant',
                         random_state = 42, learning_rate_init = 0.00001,
                         hidden_layer_sizes = (100,), max_iter = 1000,␣
↪batch_size = 25)
```

[50]:
```python
mlp_model.fit(X_train_train, y_train_train)
predictions = mlp_model.predict(X_train_val)
```

```
KFold_score = r2_score(y_train_val, predictions)
KFold_score
```

[50]: 0.5734967738018436

[35]:
```
mlp_model.fit(X_train_array, y_train_array)
predictions = mlp_model.predict(test_array)
output = pd.DataFrame({'ID': df_test['ID'],'y': predictions})
output.to_csv('sub_mlp.csv', index = False)
```

0.5.4 sklearn.linear_model: ElasticNet

- Linear regression with combined L1 and L2 priors as regularizer.
- Minimizes the objective function:
- `1 / (2 * n_samples) * ||y - Xw||^2_2`
- `alpha * l1_ratio * ||w||_1`
- `0.5 * alpha * (1 - l1_ratio) * ||w||^2_2`
- If you are interested in controlling the L1 and L2 penalty separately, keep in mind that this is equivalent to:
- `a * L1 + b * L2`
- where:
- `alpha = a + b and l1_ratio = a / (a + b)`
- The parameter l1_ratio corresponds to alpha in the glmnet R package while alpha corresponds to the lambda parameter in glmnet. Specifically, l1_ratio = 1 is the lasso penalty. Currently, l1_ratio <= 0.01 is not reliable, unless you supply your own sequence of alpha.

[36]:
```
from sklearn.linear_model import ElasticNet
elas_model = ElasticNet(alpha=0.001,normalize=True)
elas_model.fit(X_train_train, y_train_train)
```

[36]: ElasticNet(alpha=0.001, normalize=True)

[37]:
```
from math import sqrt
predictions1 = elas_model.predict(X_train_val)
Elas_score = elas_model.score(X_train_val,y_train_val)
Elas_score
```

[37]: 0.5278351276203694

[38]:
```
elas_model.fit(X_train_array, y_train_array)
predictions2 = elas_model.predict(test_array)
Elas_score2 = elas_model.score(X_train_array, y_train_array)
output = pd.DataFrame({'ID': df_test['ID'],'y': predictions2})
output.to_csv('sub_elas.csv', index = False)
Elas_score2
```

[38]: 0.5239550849260941

[39]:
```
print(sqrt(mean_squared_error(y_train_val,predictions1)))
print(sqrt(mean_squared_error(y_train_array,predictions2)))
```

```
8.572772823779191
8.622657266500235
```

0.6 Step 7: Training using xgboost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

Parameters to be used for the xgboost model

[40]:
```
X_train.head()
```

[40]:
```
   X60  X77  X339  X75  X36  X376  X18  X190  X103  X245  …  X359  X150  \
0    0    0     0    0    0     0    1     0     0     0  …     0     1
1    0    0     0    0    0     0    1     0     0     0  …     0     1
2    0    0     0    1    0     0    0     0     0     0  …     0     1
3    0    0     0    0    0     0    0     0     0     0  …     0     1
4    0    0     0    0    0     0    0     0     0     0  …     0     1

   X232  X86  X123  X250  X45  X80  X53  X65
0     0    0     0     0    0    0    0    0
1     0    0     0     1    0    1    0    0
2     1    0     0     1    0    1    0    0
3     1    0     0     1    0    1    0    0
4     1    0     0     1    0    1    0    0

[5 rows x 368 columns]
```

[41]:
```
X_test = _x_test
y_mean = y_train.mean()
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
```

```
(4209, 368)
(4209, 368)
(4209,)
```

[42]:

```python
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

xgb_params = {
    'eta': 0.005,
    'max_depth': 3,
    'subsample': 0.95,
    'colsample_bytree': 0.6,
    'objective': 'reg:squarederror',
    'eval_metric': 'rmse',
    'base_score': np.log(y_mean)
}

# form DMatrices for Xgboost training
#X_train_train, X_train_val, y_train_train, y_train_val
#dtrain = xgb.DMatrix(X_train_train, np.array(np.log(y_train_train)))
dtrain = xgb.DMatrix(X_train, np.array(np.log(y_train)))
dtest = xgb.DMatrix(X_test)

xgb_score = -1

# evaluation metric
def the_metric(y_pred, y):
    y_true = y.get_label()
    global xgb_score
    xgb_score = r2_score(y_true, y_pred)
    return 'r2', xgb_score

# xgboost, cross-validation
cv_result = xgb.cv(xgb_params,
                   dtrain,
                   num_boost_round=2000,
                   nfold = 3,
                   early_stopping_rounds=50,
                   feval=the_metric,
                   verbose_eval=True,
                   show_stdv=False
                  )

num_boost_rounds = len(cv_result)
print('num_boost_rounds=' + str(num_boost_rounds))

# train model
model = xgb.train(dict(xgb_params), dtrain, num_boost_round=num_boost_rounds)
```

```
[0]     train-rmse:0.12099      train-r2:0.00161        test-rmse:0.12099
test-r2:0.00037
[1]     train-rmse:0.12062      train-r2:0.00770        test-rmse:0.12062
test-r2:0.00648
[2]     train-rmse:0.12025      train-r2:0.01393        test-rmse:0.12024
test-r2:0.01269
[3]     train-rmse:0.11988      train-r2:0.01987        test-rmse:0.11988
test-r2:0.01859
[4]     train-rmse:0.11951      train-r2:0.02593        test-rmse:0.11951
test-r2:0.02462
[5]     train-rmse:0.11914      train-r2:0.03198        test-rmse:0.11914
test-r2:0.03067
[6]     train-rmse:0.11877      train-r2:0.03795        test-rmse:0.11877
test-r2:0.03664
[7]     train-rmse:0.11841      train-r2:0.04378        test-rmse:0.11841
test-r2:0.04246
[8]     train-rmse:0.11806      train-r2:0.04948        test-rmse:0.11806
test-r2:0.04810
[9]     train-rmse:0.11772      train-r2:0.05486        test-rmse:0.11773
test-r2:0.05349
[10]    train-rmse:0.11736      train-r2:0.06063        test-rmse:0.11737
test-r2:0.05922
[11]    train-rmse:0.11701      train-r2:0.06627        test-rmse:0.11702
test-r2:0.06485
[12]    train-rmse:0.11667      train-r2:0.07173        test-rmse:0.11668
test-r2:0.07037
[13]    train-rmse:0.11632      train-r2:0.07724        test-rmse:0.11633
test-r2:0.07585
[14]    train-rmse:0.11599      train-r2:0.08245        test-rmse:0.11600
test-r2:0.08105
[15]    train-rmse:0.11566      train-r2:0.08777        test-rmse:0.11566
test-r2:0.08642
[16]    train-rmse:0.11532      train-r2:0.09305        test-rmse:0.11533
test-r2:0.09164
[17]    train-rmse:0.11499      train-r2:0.09829        test-rmse:0.11500
test-r2:0.09681
[18]    train-rmse:0.11465      train-r2:0.10363        test-rmse:0.11467
test-r2:0.10212
[19]    train-rmse:0.11432      train-r2:0.10876        test-rmse:0.11434
test-r2:0.10723
[20]    train-rmse:0.11398      train-r2:0.11395        test-rmse:0.11401
test-r2:0.11239
[21]    train-rmse:0.11366      train-r2:0.11898        test-rmse:0.11369
test-r2:0.11735
[22]    train-rmse:0.11333      train-r2:0.12410        test-rmse:0.11336
test-r2:0.12244
[23]    train-rmse:0.11301      train-r2:0.12905        test-rmse:0.11304
test-r2:0.12733
```

```
[24]    train-rmse:0.11270      train-r2:0.13379        test-rmse:0.11274
test-r2:0.13203
[25]    train-rmse:0.11239      train-r2:0.13861        test-rmse:0.11243
test-r2:0.13678
[26]    train-rmse:0.11206      train-r2:0.14353        test-rmse:0.11211
test-r2:0.14170
[27]    train-rmse:0.11174      train-r2:0.14843        test-rmse:0.11179
test-r2:0.14657
[28]    train-rmse:0.11143      train-r2:0.15319        test-rmse:0.11148
test-r2:0.15132
[29]    train-rmse:0.11112      train-r2:0.15797        test-rmse:0.11117
test-r2:0.15607
[30]    train-rmse:0.11081      train-r2:0.16267        test-rmse:0.11086
test-r2:0.16076
[31]    train-rmse:0.11049      train-r2:0.16738        test-rmse:0.11055
test-r2:0.16546
[32]    train-rmse:0.11018      train-r2:0.17202        test-rmse:0.11024
test-r2:0.17009
[33]    train-rmse:0.10989      train-r2:0.17644        test-rmse:0.10995
test-r2:0.17442
[34]    train-rmse:0.10959      train-r2:0.18101        test-rmse:0.10965
test-r2:0.17898
[35]    train-rmse:0.10929      train-r2:0.18545        test-rmse:0.10935
test-r2:0.18341
[36]    train-rmse:0.10902      train-r2:0.18945        test-rmse:0.10908
test-r2:0.18743
[37]    train-rmse:0.10872      train-r2:0.19388        test-rmse:0.10878
test-r2:0.19184
[38]    train-rmse:0.10843      train-r2:0.19814        test-rmse:0.10850
test-r2:0.19605
[39]    train-rmse:0.10814      train-r2:0.20250        test-rmse:0.10821
test-r2:0.20039
[40]    train-rmse:0.10785      train-r2:0.20679        test-rmse:0.10792
test-r2:0.20466
[41]    train-rmse:0.10756      train-r2:0.21106        test-rmse:0.10763
test-r2:0.20893
[42]    train-rmse:0.10729      train-r2:0.21502        test-rmse:0.10736
test-r2:0.21289
[43]    train-rmse:0.10701      train-r2:0.21908        test-rmse:0.10709
test-r2:0.21689
[44]    train-rmse:0.10673      train-r2:0.22320        test-rmse:0.10680
test-r2:0.22100
[45]    train-rmse:0.10648      train-r2:0.22676        test-rmse:0.10656
test-r2:0.22455
[46]    train-rmse:0.10620      train-r2:0.23082        test-rmse:0.10628
test-r2:0.22860
[47]    train-rmse:0.10594      train-r2:0.23465        test-rmse:0.10602
test-r2:0.23238
```

```
[48]	train-rmse:0.10566	train-r2:0.23865	test-rmse:0.10574
test-r2:0.23637
[49]	train-rmse:0.10539	train-r2:0.24258	test-rmse:0.10547
test-r2:0.24030
[50]	train-rmse:0.10511	train-r2:0.24651	test-rmse:0.10520
test-r2:0.24420
num_boost_rounds=1
```

[43]:
```
# Predict on trian and test
y_train_pred = np.exp(model.predict(dtrain))
y_train_pred
```

[43]:
```
array([100.62033 , 100.62033 , 100.52241 , …, 100.705864, 100.62033 ,
       100.62033 ], dtype=float32)
```

[44]:
```
# Predict on trian and test
y_pred = np.exp(model.predict(dtest))
y_pred
```

[44]:
```
array([100.52241 , 100.62033 , 100.52241 , …, 100.62033 , 100.705864,
       100.62033 ], dtype=float32)
```

[45]:
```
output = pd.DataFrame({'id': df_test['ID'].astype(np.int32), 'y': y_pred})
output.to_csv('sub_15_encoded.csv', index=False)
```

1 Summary

[46]:
```
# X_train_train, X_train_val, y_train_train, y_train_val
```

[47]:
```
print('R2 = {} with Elas'.format(Elas_score))
print('R2 = {} with KFold'.format(KFold_score))
print('R2 = {} with Ridge'.format(Ridge_score))
print('R2 = {} with Lasso'.format(Lasso_score))
print('R2 = {} with XGBoost'.format(xgb_score))
```

```
R2 = 0.5278351276203694 with Elas
R2 = 0.5734967738018436 with KFold
R2 = 0.5731494510022381 with Ridge
R2 = 0.5680120978533502 with Lasso
R2 = 0.2452701073023824 with XGBoost
```

1.1 Results

Out of the five models evaluated the Lasso model score showed it to be the most accurate to be used.

[ ]:
```

```

[ ]:

```

```