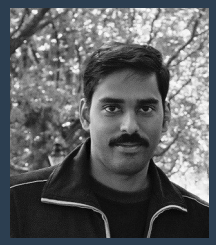
Mercedes-Benz Greener Manufacturing

Predicting Vehicle Testing Time



Justin Jose

https://github.com/justinpolackal/VehicleTestingTime-machine-learning

www.linkedin.com/in/justinpolackal

UPX Academy: Course Project : Machine Learning

Python Code and other project resources

https://github.com/justinpolackal/VehicleTestingTime-machine-learning

Objective

Daimler’s 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 each and every unique car configuration before they hit the road, Daimler’s engineers have developed a robust testing system. But, optimizing the speed of their testing system for so many possible feature combinations are complex and time-consuming without a powerful algorithmic approach.

The objective of this exercise is to :

- **Device a Machine Learning Approach to predict the vehicle testing time for a given feature combination.**
- **Tackle the curse of dimensionality (Use PCA for dimensionality reduction) while devising the prediction system.**
-

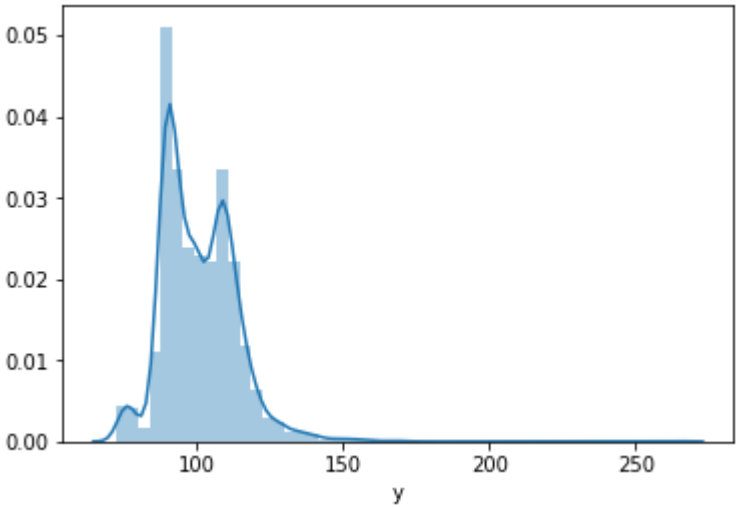
Dataset

- **The dataset is a CSV file containing an anonymized set of variables (X0 to X385), each representing a custom feature in a Mercedes car.**
- **Since this is confidential data, all features are de-identified.**
- **The ground truth is available as “y” column in the dataset**
- **Variables with letters are categorical. Variables with 0/1 are binary values**

Data Analysis

- **The Y variable (dependent variable) is continuous.**
- **Dataset contains variables X0,X1,X2,X3,X4,X5,X6 and X8 that are character type and are categorical.**
- **Remaining variables X10 to X385 are binary variables (1 represents the presence of a feature in the car, while 0 shows the absence of the feature)**

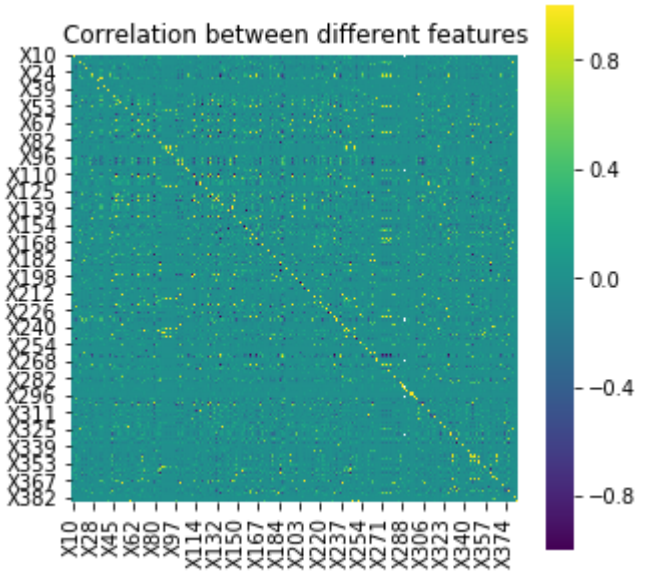
The distribution of dependent variable y shows two peaks.



Data Preparation

- **Character values in X0,X1,X2,X3,X4,X5,X6 and X8 variables are dummy coded to convert them into binary values. (One-hot encoding)**
- **There are no null values**
- **After one-hot encoding, the data set has 563 variables. The number of independent variables has to be reduced using PCA (Principal Component Analysis)**

Correlation plot of independent variables does not really show any significant trend



One-hot encoding

Encoding Categorical variables

Transform string type categorical variables to numerical and binary types

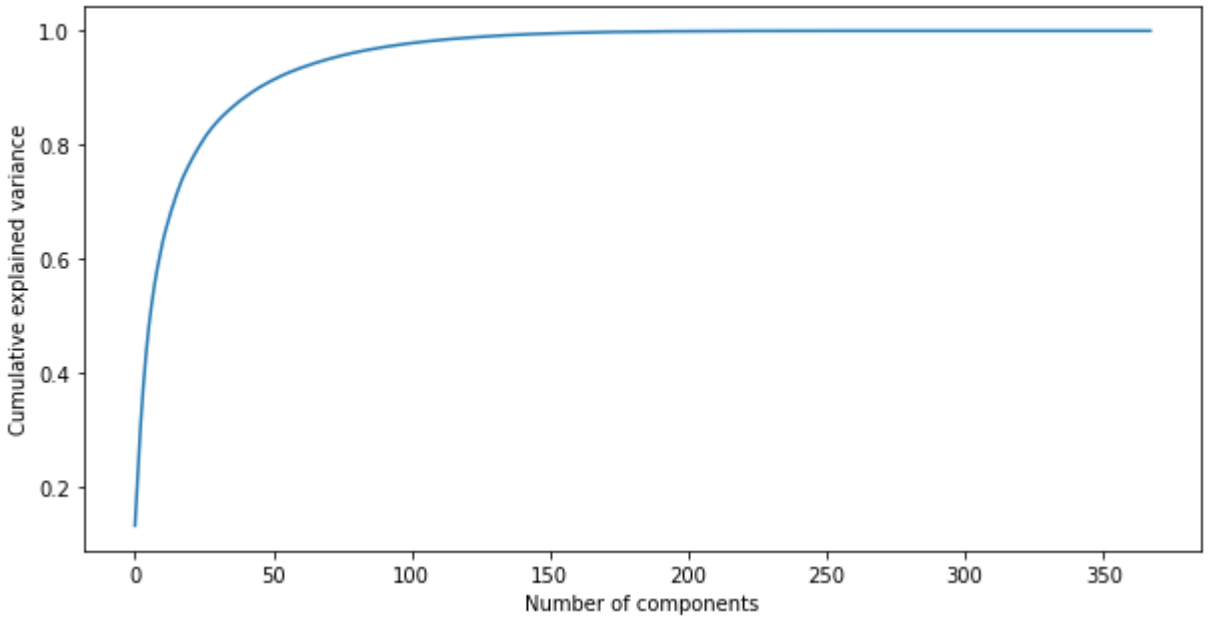
- Use Pandas get_dummies() function to one-hot-encode character values into binary
- Append theose dataframes into the original dataset
- Drop original string categorical variables

```
dfx0 = pd.get_dummies(rawdata['X0'],prefix='X0_')
dfx1 = pd.get_dummies(rawdata['X1'],prefix='X1_')
dfx2 = pd.get_dummies(rawdata['X2'],prefix='X2_')
dfx3 = pd.get_dummies(rawdata['X3'],prefix='X3_')
dfx4 = pd.get_dummies(rawdata['X4'],prefix='X4_')
dfx5 = pd.get_dummies(rawdata['X5'],prefix='X5_')
dfx6 = pd.get_dummies(rawdata['X6'],prefix='X6_')
dfx8 = pd.get_dummies(rawdata['X8'],prefix='X8_')
dfx = pd.concat([dfx0, dfx1, dfx2, dfx3, dfx4, dfx5, dfx6, dfx8], axis=1)
rawdata = pd.concat([rawdata,dfx], axis=1)
```

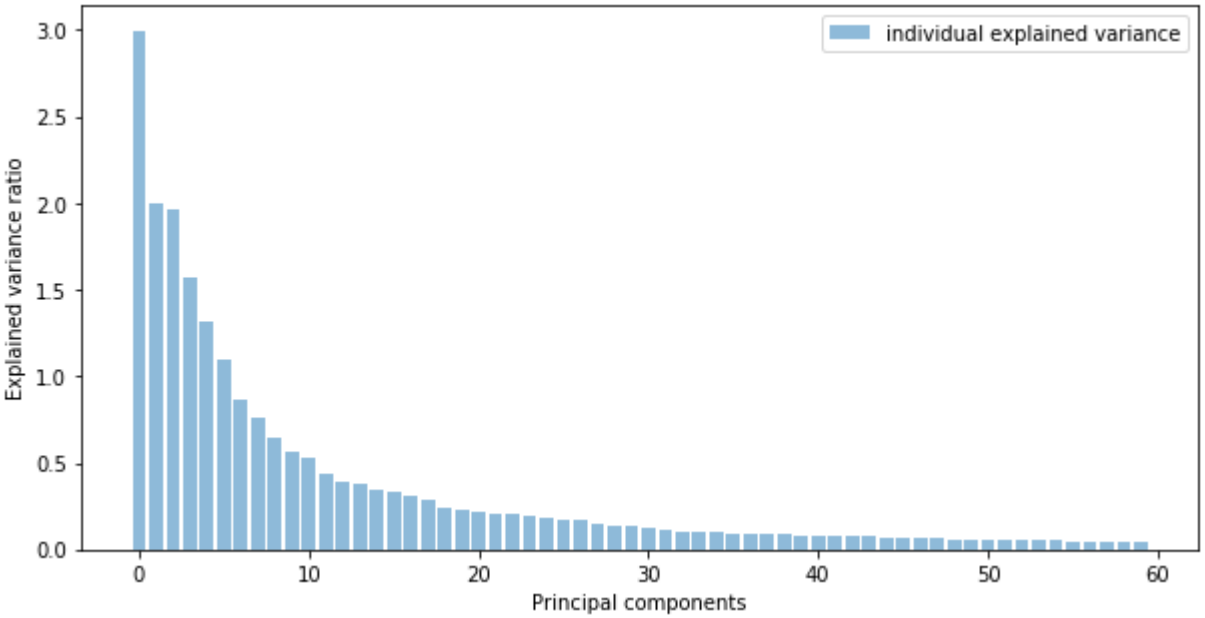
Principal Component Analysis (PCA)

- **The cumulative explained variance after fitting the PCA.**
- **The kink on the graph happens somewhere near 50 and the cumulative explained variance approaches around 90% this point.**

```
pca = PCA().fit(X_train)
plt.figure(figsize=(10,5))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```



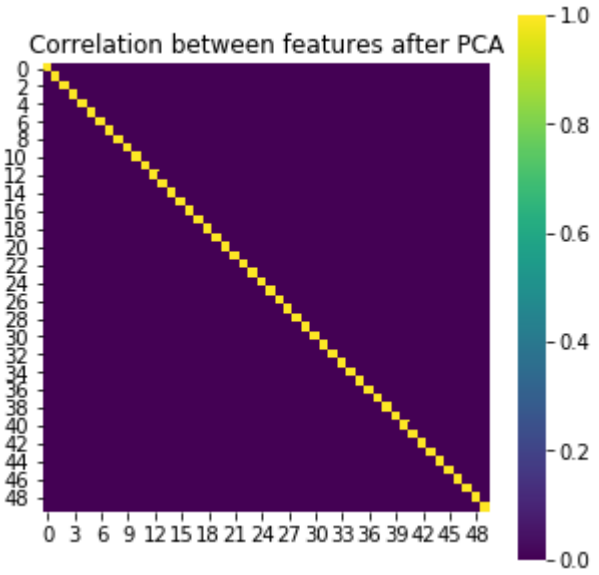
PCA – Top 50 components



- **Individual explained variance is plotted**
- **As per the graph, approximately 50 components can explain about 90% of the variance.**

The right-hand side chart shows the correlation between top 50 PCA components.

There is no correlation seen, which is expected as PCA components will be orthogonal and will not have multi-colliniarity.



Python code snippet for PCA

```
pca = PCA().fit(X_train)
plt.figure(figsize=(10,5))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

```
#consider first 50 components as they are explaining more than 90% of variation in the data
x_pca = PCA(n_components=num_pca_components)
#transform training set X values using 15 component PCA
x_train_pca = x_pca.fit_transform(X_train)
```

Build Regression Models

- **Following regression models were implemented and compared for this exercise.**
  - Multiple Linear Regression
  - Lasso, Ridge and ElasticNet Regression
  - Bayesian Ridge Regression
  - Support Vector Machine Regression
  - Random Forest Regression
  - XGBoost
  - Stacked Pipeline Model using ExtraTrees Regressor and Random Forest

ScikitLearn Library

The Scikitlearn library for machine learning provides a generalized syntax for model instantiation, training and testing

Create Train and Test datasets

```
X_train, X_test, y_train, y_test = train_test_split(rawdata, y_labels, random_state=123)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

Use x_train_pca and y_train as X and Y variables

```
# transform the test data set using the same PCA object
x_test_pca = x_pca.fit_transform(X_test)
```

Sample Python code for Linear Regression

Linear Regression

```
linreg = LinearRegression()
linreg.fit(x_train_pca, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
# Predict for the test data set
y_pred_lr = linreg.predict(x_test_pca)
```

Additional Libraries used

Xgboost – ExtremeGradient Boosting – provides gradient boosted decision tree model

Tpot – is a library to help automate machine learning. StackingEstimator from tpot library is used to stack models.

Models and Scores

- **Almost all basic regression models were implemented and compared**
- **The predictions from stacked models were included to calculate weighted average with results from Random Forest regressor.**

Evaluating all the models built

- **All the models built for the project were evaluated based on error terms such as RMSE (Root Mean Square Error) and MAE (Mean Absolute Error)**
- **RSquared values are also compared for all models, which is the percentage variation in Y explained by all the X variables together.**

Various Models Compared

Combination of stacked models and random Forest classifier shows improvement in RMSE as well as Rsquared values

	Model	Null-RMSE	RMSE	MAE	RSquared
0	Linear Model	12.603966	10.977665	8.021208	24.14
1	Lasso	12.603966	10.807772	7.934880	26.47
2	Ridge	12.603966	10.975871	8.019529	24.17
3	ElasticNet	12.603966	10.932992	8.087122	24.76
4	Random Forest	12.603966	9.826961	6.977092	39.21
5	Support Vector Regression	12.603966	10.604455	7.370170	29.21
6	XGBoost	12.603966	9.852228	7.019464	38.90
7	Stacked Model RF	12.603966	11.668967	8.069630	14.29
8	Stacked Model ET	12.603966	10.195976	7.352537	34.56
9	Combination1 - RF,ElasticNet	12.603966	9.630098	6.739024	41.62
10	Combination2 - RF, ExtraTrees	12.603966	9.553413	6.722272	42.55
11	Bayesian Ridge	12.603966	10.929101	7.975903	24.81

Thank You