Final.ipynb ☆

File Edit View Insert Runtime Tools Help All ...

Comment Share

+ Code + Text

RAM Disk

Editing

[1]

```
import numpy as np
np.random.seed(123)
import pandas as pd
import pandas.testing as tm
import pandas.util.testing as tm
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from tqdm import tqdm_notebook
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import PCA
from sklearn.decomposition import FastICA
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
import xgboost as xgb
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn import linear_model
from sklearn import metrics
from sklearn.ensemble import AdaBoostRegressor
from sklearn.svm import SVR
import json
from tqdm import tqdm_notebook
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: FutureWarning: pandas.
  """
```

[2]

```
# loading the train and test dataset
df_1=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Case Study 1/mercedes-b
df_2=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Case Study 1/mercedes-b
```

[3]

```
def final_pred(df_1, df_2):
    # converting categorical features to numerical features
```

```
x_dummies_1=pd.get_dummies(df_1)
x_dummies_2=pd.get_dummies(df_2)

missing_cols = set( x_dummies_1.columns ) - set( x_dummies_2.columns )
y=x_dummies_1['y']
# align dataframes
x_dummies_1, x_dummies_2= x_dummies_1.align(x_dummies_2, join='inner', axis=1)

# creating a list of columns which have only zeros
zeros=[]
for i,j in x_dummies_1.any().items():#https://pandas.pydata.org/pandas-docs/st
    if j==False:
        zeros.append(i)

# dropping the columns containing only zeros
x_dummies_1 = x_dummies_1.drop(zeros, axis=1)
x_dummies_2 = x_dummies_2.drop(zeros, axis=1)
# saving the dataframes as a backup for further computations
x_1_safe = x_dummies_1
x_2_safe = x_dummies_2
x_dummies_1['y']=y
# Dropping Outliers
x_filtered= x_dummies_1[x_dummies_1['y']>70]#https://www.geeksforgeeks.org/dr
x_filtered= x_filtered[x_filtered['y']<150]
y=x_filtered['y']
x_filtered=x_filtered.drop(['y'], axis=1)
top_20_features= ['ID', 'X14', 'X29', 'X54', 'X76', 'X118', 'X119', 'X127', '
   'X136', 'X189', 'X222', 'X232', 'X263', 'X279', 'X311', 'X314', 'X315',
   'X6_g', 'X6_j']
def calc_vif(X):

    # Calculationg VIF
    vif=pd.DataFrame()
    vif['variables']=X.columns
    vif['VIF']= [variance_inflation_factor(X.values, i) for i in tqdm_notebool

    return (vif)
'''vif_values=calc_vif(x_filtered)
vif_values=vif_values.sort_values(by='VIF', ascending=True)
lst_variables= list(vif_values['variables'])
lst_vif= list(vif_values['VIF'])
vif_dict = dict(zip(lst_variables, lst_vif))'''
'''inf_indices=[]
count=0
for k,v in vif_dict.items():
    if np.isinf(v):
        inf_indices.append(k)
        count+=1'''
'''# removing top_20_features from the features that need to be dropped from
inf_indices=list(set(inf_indices)-set(top_20_features))'''
with open('/content/drive/My Drive/Colab Notebooks/Case Study 1/inf_indices_c
    checc=json.loads(f.read())
inf_indices=checc
```

```
    # dropping the features
    x_filtered = x_filtered.drop(inf_indices, axis=1)
    x_dummies_2= x_dummies_2.drop(inf_indices, axis=1)
    # generating SVD features
    tsvd= TruncatedSVD(n_components=2, random_state=42)
    tsvd_train= tsvd.fit_transform(x_filtered)
    tsvd_test=  tsvd.transform(x_dummies_2)
    # generating PCA features
    pca = PCA(n_components=2, random_state=42)
    pca_train= pca.fit_transform(x_filtered)
    pca_test= pca.transform(x_dummies_2)
    # generating ICA features
    ica=FastICA(n_components=2, random_state=42)
    ica_train= ica.fit_transform(x_filtered)
    ica_test= ica.transform(x_dummies_2)
    # adding these dimensionality reduction features to the dataset
    for i in range(0, tsvd_train.shape[1]):
        x_filtered['tsvd_'+str(i)]= tsvd_train[:, i]
        x_dummies_2['tsvd_'+str(i)]= tsvd_test[:, i]
        x_filtered['pca_'+str(i)]= pca_train[:, i]
        x_dummies_2['pca_'+str(i)]= pca_test[:, i]
        x_filtered['ica_'+str(i)]= ica_train[:, i]
        x_dummies_2['ica_'+str(i)]= ica_test[:, i]

    # adding new features using feature interaction of important features
    x_filtered['X64 + X218']=x_filtered['X64']+x_filtered['X218']
    x_dummies_2['X64 + X218']=x_dummies_2['X64']+x_dummies_2['X218']

    x_filtered['X218 + X224 + X273']=x_filtered['X218']+x_filtered['X224'] + x_fi
    x_dummies_2['X218 + X224 + X273']=x_dummies_2['X218']+x_dummies_2['X224'] + x_

    x_filtered['X64 + X224 + X273']=x_filtered['X64']+x_filtered['X224'] + x_filt
    x_dummies_2['X64 + X224 + X273']=x_dummies_2['X64']+x_dummies_2['X224'] + x_d

    x_filtered['X314 + X315']=x_filtered['X314']+x_filtered['X315']
    x_dummies_2['X314 + X315']=x_dummies_2['X314']+x_dummies_2['X315']

    x_filtered['X314 + X315 + X29']=x_filtered['X314']+x_filtered['X315'] + x_fil
    x_dummies_2['X314 + X315 + X29']=x_dummies_2['X314']+x_dummies_2['X315'] + x_(

    x_filtered['X314 + X315 + X118']=x_filtered['X314']+x_filtered['X315'] + x_fi
    x_dummies_2['X314 + X315 + X118']=x_dummies_2['X314']+x_dummies_2['X315'] + x_

    x_filtered['X127 + X189']=x_filtered['X127']+x_filtered['X189']
    x_dummies_2['X127 + X189']=x_dummies_2['X127']+x_dummies_2['X189']

    x_filtered['X118 + X54']=x_filtered['X118']+x_filtered['X54']
    x_dummies_2['X118 + X54']=x_dummies_2['X118']+x_dummies_2['X54']

    x_train, x_test, y_train, y_test = train_test_split(x_filtered, y, test_size=(

    # applying RandomForestRegressor model
    # Number of trees in a Random forest
```

```
    n_estimators=[int(x) for x in np.linspace (start=100, stop=1000, num=10)]

    # Number of features to consider at every split
    max_features=['auto', 'sqrt']

    # Maximun no of levels in a tree
    max_depth=[int(x) for x in np.linspace(10, 110, num = 11)]
    max_depth.append(None)

    # minimum number of samples required to split a node
    min_samples_split = np.arange(50,250,20)

    # Minimum number of samples required at each leaf node
    min_samples_leaf = np.arange(5,50,5)
    # Method of selecting samples for training each tree
    bootstrap = [True, False]

    # Create the random grid
    random_grid = {'n_estimators': n_estimators,
                   'max_features': max_features,
                   'max_depth': max_depth,
                   'min_samples_split': min_samples_split,
                   'min_samples_leaf': min_samples_leaf,
                   'bootstrap': bootstrap}
    # First create the base model to tune
    rf = RandomForestRegressor()
    # Random search of parameters, using 3 fold cross validation,
    # search across 100 different combinations, and use all available cores
    '''rf_random = RandomizedSearchCV(estimator = rf, param_distributions = rando
    # Fit the random search model
    rf_random.fit(x_filtered, y)
    best_rf=rf_random.best_estimator_'''
    best_rf= RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse'
                          max_depth=70, max_features='auto', max_leaf_nodes=None,
                          max_samples=None, min_impurity_decrease=0.0,
                          min_impurity_split=None, min_samples_leaf=40,
                          min_samples_split=110, min_weight_fraction_leaf=0.0,
                          n_estimators=500, n_jobs=None, oob_score=False,
                          random_state=None, verbose=0, warm_start=False)
    # fitting the best model on to the dataset
    best_rf.fit(x_filtered, y)
    y_train_pred=best_rf.predict(x_filtered)
    y_test_pred=best_rf.predict(x_dummies_2)
    return y_train_pred, y_test_pred

[4]  1 y_train_pred, y_test_pred = final_pred(df_1, df_2)

[ ]  1

[5]  1 # loading the train and test dataset
```

```
df_1=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Case Study 1/mercedes-b
df_2=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Case Study 1/mercedes-b
def final_metric(df_1, df_2):
    # converting categorical features to numerical features
    x_dummies_1=pd.get_dummies(df_1)
    x_dummies_2=pd.get_dummies(df_2)

    missing_cols = set( x_dummies_1.columns ) - set( x_dummies_2.columns )
    y=x_dummies_1['y']
    # align dataframes
    x_dummies_1, x_dummies_2= x_dummies_1.align(x_dummies_2, join='inner', axis=1

    # creating a list of columns which have only zeros
    zeros=[]
    for i,j in x_dummies_1.any().items():#https://pandas.pydata.org/pandas-docs/s
        if j==False:
            zeros.append(i)

    # dropping the columns containing only zeros
    x_dummies_1 = x_dummies_1.drop(zeros, axis=1)
    x_dummies_2 = x_dummies_2.drop(zeros, axis=1)
    # saving the dataframes as a backup for further computations
    x_1_safe = x_dummies_1
    x_2_safe = x_dummies_2
    x_dummies_1['y']=y
    # Dropping Outliers
    x_filtered= x_dummies_1[x_dummies_1['y']>70]#https://www.geeksforgeeks.org/dr
    x_filtered= x_filtered[x_filtered['y']<150]
    y=x_filtered['y']
    x_filtered=x_filtered.drop(['y'], axis=1)
    top_20_features= ['ID', 'X14', 'X29', 'X54', 'X76', 'X118', 'X119', 'X127', '
       'X136', 'X189', 'X222', 'X232', 'X263', 'X279', 'X311', 'X314', 'X315',
       'X6_g', 'X6_j']
    def calc_vif(X):

        # Calculationg VIF
        vif=pd.DataFrame()
        vif['variables']=X.columns
        vif['VIF']= [variance_inflation_factor(X.values, i) for i in tqdm_notebool

        return (vif)
    '''vif_values=calc_vif(x_filtered)
    vif_values=vif_values.sort_values(by='VIF', ascending=True)
    lst_variables= list(vif_values['variables'])
    lst_vif= list(vif_values['VIF'])
    vif_dict = dict(zip(lst_variables, lst_vif))'''
    '''inf_indices=[]
    count=0
    for k,v in vif_dict.items():
        if np.isinf(v):
            inf_indices.append(k)
            count+=1'''
    '''# removing top_20_features from the features that need to be dropped from
```

```
    inf_indices=list(set(inf_indices)-set(top_20_features))'''
    with open('/content/drive/My Drive/Colab Notebooks/Case Study 1/inf_indices_c
        checc=json.loads(f.read())
    inf_indices=checc
    # dropping the features
    x_filtered = x_filtered.drop(inf_indices, axis=1)
    x_dummies_2= x_dummies_2.drop(inf_indices, axis=1)
    # generating SVD features
    tsvd= TruncatedSVD(n_components=2, random_state=42)
    tsvd_train= tsvd.fit_transform(x_filtered)
    tsvd_test=  tsvd.transform(x_dummies_2)
    # generating PCA features
    pca = PCA(n_components=2, random_state=42)
    pca_train= pca.fit_transform(x_filtered)
    pca_test= pca.transform(x_dummies_2)
    # generating ICA features
    ica=FastICA(n_components=2, random_state=42)
    ica_train= ica.fit_transform(x_filtered)
    ica_test= ica.transform(x_dummies_2)
    # adding these dimensionality reduction features to the dataset
    for i in range(0, tsvd_train.shape[1]):
        x_filtered['tsvd_'+str(i)]= tsvd_train[:, i]
        x_dummies_2['tsvd_'+str(i)]= tsvd_test[:, i]
        x_filtered['pca_'+str(i)]= pca_train[:, i]
        x_dummies_2['pca_'+str(i)]= pca_test[:, i]
        x_filtered['ica_'+str(i)]= ica_train[:, i]
        x_dummies_2['ica_'+str(i)]= ica_test[:, i]

    # adding new features using feature interaction of important features
    x_filtered['X64 + X218']=x_filtered['X64']+x_filtered['X218']
    x_dummies_2['X64 + X218']=x_dummies_2['X64']+x_dummies_2['X218']

    x_filtered['X218 + X224 + X273']=x_filtered['X218']+x_filtered['X224'] + x_fi
    x_dummies_2['X218 + X224 + X273']=x_dummies_2['X218']+x_dummies_2['X224'] + x_

    x_filtered['X64 + X224 + X273']=x_filtered['X64']+x_filtered['X224'] + x_filt
    x_dummies_2['X64 + X224 + X273']=x_dummies_2['X64']+x_dummies_2['X224'] + x_d

    x_filtered['X314 + X315']=x_filtered['X314']+x_filtered['X315']
    x_dummies_2['X314 + X315']=x_dummies_2['X314']+x_dummies_2['X315']

    x_filtered['X314 + X315 + X29']=x_filtered['X314']+x_filtered['X315'] + x_fil
    x_dummies_2['X314 + X315 + X29']=x_dummies_2['X314']+x_dummies_2['X315'] + x_

    x_filtered['X314 + X315 + X118']=x_filtered['X314']+x_filtered['X315'] + x_fi
    x_dummies_2['X314 + X315 + X118']=x_dummies_2['X314']+x_dummies_2['X315'] + x_

    x_filtered['X127 + X189']=x_filtered['X127']+x_filtered['X189']
    x_dummies_2['X127 + X189']=x_dummies_2['X127']+x_dummies_2['X189']

    x_filtered['X118 + X54']=x_filtered['X118']+x_filtered['X54']
    x_dummies_2['X118 + X54']=x_dummies_2['X118']+x_dummies_2['X54']

```

```
    x_train, x_test, y_train, y_test = train_test_split(x_filtered, y, test_size=(

    # applying RandomForestRegressor model
    # Number of trees in a Random forest
    n_estimators=[int(x) for x in np.linspace (start=100, stop=1000, num=10)]

    # Number of features to consider at every split
    max_features=['auto', 'sqrt']

    # Maximun no of levels in a tree
    max_depth=[int(x) for x in np.linspace(10, 110, num = 11)]
    max_depth.append(None)

    # minimum number of samples required to split a node
    min_samples_split = np.arange(50,250,20)

    # Minimum number of samples required at each leaf node
    min_samples_leaf = np.arange(5,50,5)
    # Method of selecting samples for training each tree
    bootstrap = [True, False]

    # Create the random grid
    random_grid = {'n_estimators': n_estimators,
                   'max_features': max_features,
                   'max_depth': max_depth,
                   'min_samples_split': min_samples_split,
                   'min_samples_leaf': min_samples_leaf,
                   'bootstrap': bootstrap}
    # First create the base model to tune
    rf = RandomForestRegressor()
    # Random search of parameters, using 3 fold cross validation,
    # search across 100 different combinations, and use all available cores
    '''rf_random = RandomizedSearchCV(estimator = rf, param_distributions = rando
    # Fit the random search model
    rf_random.fit(x_filtered, y)
    best_rf=rf_random.best_estimator_'''
    best_rf= RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse'
                           max_depth=70, max_features='auto', max_leaf_nodes=None,
                           max_samples=None, min_impurity_decrease=0.0,
                           min_impurity_split=None, min_samples_leaf=40,
                           min_samples_split=110, min_weight_fraction_leaf=0.0,
                           n_estimators=500, n_jobs=None, oob_score=False,
                           random_state=None, verbose=0, warm_start=False)
    # fitting the best model on to the dataset
    best_rf.fit(x_filtered, y)
    y_train_pred=best_rf.predict(x_filtered)
    y_test_pred=best_rf.predict(x_dummies_2)

    return r2_score(y, y_train_pred)

```

```
[6]   1 final_metric(df_1, df_2)
```

```
0.6535470835738704
```

```
[7]  1 final_df= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID',
```

```
[8]  1 final_df.head()
```

	ID	y
0	1	78.505489
1	2	94.308220
2	3	78.433665
3	4	78.505489
4	5	113.414816

```
1 #final_df.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/RandomFo
```