**import** numpy **as** np

**import** pandas **as** pd

**from** sklearn.metrics **import** r2\_score

**from** sklearn **import** preprocessing, metrics

**import** gc;gc.enable()

**from** sklearn.neural\_network **import** MLPRegressor

**from** sklearn.ensemble **import** RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor

*#Transforming the datasets attributes according to what has been asked in the competition*

**def** DataFrameTransform(DataFrame):

DataFrame[**'date'**] = DataFrame[**'date'**].dt.dayofweek

DataFrame[**'day'**] = DataFrame[**'date'**].dt.day

DataFrame[**'date'**] = pd.to\_datetime(DataFrame[**'date'**])

DataFrame[**'month'**] = DataFrame[**'date'**].dt.month

DataFrame[**'year'**] = DataFrame[**'date'**].dt.year

*#Competition problem had asked to count perishable attribute of the dataset to be mapped to 1 and 1.25 according to what it contains*

DataFrame[**'perishable'**] = DataFrame[**'perishable'**].map({0: 1.0, 1: 1.25})

*#We had to map the on promotion attribute of the training dataset to include the onpromotion attribute in the training state*

DataFrame[**'onpromotion'**] = DataFrame[**'onpromotion'**].map({**'False'**: 0, **'True'**: 1})

*#changing the NaN values to -1 so that the data is scaled, which won't create a problem or an error in the training process*

DataFrame = DataFrame.fillna(-1)

**return** DataFrame

*#encoding the datasets*

**def** DataFrameLabelEncoder(DataFrame):

**for** column **in** DataFrame.columns:

**if** DataFrame[column].dtype == **'object'**:

LabelEncoder = preprocessing.LabelEncoder()

DataFrame[column] = LabelEncoder.fit\_transform(DataFrame[column])

**return** DataFrame

*#random forest regressor, so far this has been producing the best results so far*

**def** RandomForest(Train1, Train2, Target1, Target2, columns):

print(**'\n-> Random Forest'**)

rf = RandomForestRegressor(n\_estimators=100, max\_depth=20, min\_samples\_split=3, min\_samples\_leaf=1, n\_jobs=-1, max\_features=**"auto"**)

rf.fit(Train1[columns], Target1)

*# predicitng for cross-validation so that we can find out the errors produced in the training and predicting*

pred = rf.predict(Train2[columns])

*# finding the errors for this prediction*

Error11 = ErrorFunction(Target2, pred, Train2[**'perishable'**])

Error21 = ErrorFunction2(Target2, pred, Train2[**'perishable'**])

*# predicting the transactions for the test data which then again will be predicted for unit\_sales*

test[**'transactions'**] = rf.predict(test[columns])

test[**'transactions'**] = test[**'transactions'**].clip(lower=0. + 1e-15)

columns = [col **for** col **in** Train1 **if** col **not in** [**'id'**, **'unit\_sales'**, **'perishable'**]]

Target1 = Train1[**'unit\_sales'**].values

Target2 = Train2[**'unit\_sales'**].values

rf = RandomForestRegressor(n\_estimators=100, max\_depth=20, min\_samples\_split=3, min\_samples\_leaf=1, n\_jobs=-1, max\_features=**"auto"**)

rf.fit(Train1[columns], Target1)

*# predicting the second portion of the training set for cross-valdiation purpose*

pred = rf.predict(Train2[columns])

*# finding the error for the same*

Error12 = ErrorFunction(Target2, pred, Train2[**'perishable'**])

Error22 = ErrorFunction2(Target2, pred, Train2[**'perishable'**])

print(**'\n-> Mean Squared Error(1): '**, Error21, **'Error(2): '**, Error22)

print(**'\n-> Mean Squared Log Error(1): '**, Error11, **'Error(2): '**, Error12)

*# finally predicting the unit\_sales of the testing data*

test[**'unit\_sales'**] = rf.predict(test[columns])

cut = 0. + 1e-12 *# 0.+1e-15*

test[**'unit\_sales'**] = (np.exp(test[**'unit\_sales'**]) - 1).clip(lower=cut)

*# printing the results, this line can be commented in case you want to see the results in the form of csv*

*# print(test[['id', 'unit\_sales']])*

*# setting up the output csv file with values of the errors obtained earlier*

*# Output = 'RandomForest'+'MQLE'+str(Error11)+'and'+str(Error12)+'MSE'+str(Error21)+'and'+str(Error22)+'.csv'*

*# Exporting the data from the program to the csv file*

*# test[['id','unit\_sales']].to\_csv(Output, index=False, float\_format='%.2f')*

*#Deep Learning, so far this has been producting the worst results*

**def** DeepLearning(Train1, Train2, Target1, Target2, columns):

print(**'\n-> Deep Learning'**)

mlp = MLPRegressor(hidden\_layer\_sizes=(5,5), activation=**'relu'**, solver=**'lbfgs'**, alpha=0.01,

learning\_rate=**'adaptive'**, learning\_rate\_init=1)

mlp.fit(Train1[columns], Target1)

*#predicitng for cross-validation so that we can find out the errors produced in the training and predicting*

pred = mlp.predict(Train2[columns])

*#finding the errors for this prediction*

Error11 = ErrorFunction(Target2, pred, Train2[**'perishable'**])

Error21 = ErrorFunction2(Target2, pred, Train2[**'perishable'**])

*#predicting the transactions for the test data which then again will be predicted for unit\_sales*

test[**'transactions'**] = mlp.predict(test[columns])

test[**'transactions'**] = test[**'transactions'**].clip(lower=0. + 1e-15)

columns = [col **for** col **in** Train1 **if** col **not in** [**'id'**, **'unit\_sales'**, **'perishable'**]]

Target1 = Train1[**'unit\_sales'**].values

Target2 = Train2[**'unit\_sales'**].values

mlp = MLPRegressor(hidden\_layer\_sizes=(5,5), activation=**'relu'**, solver=**'lbfgs'**, alpha=0.01, learning\_rate=**'adaptive'**, learning\_rate\_init=1)

mlp.fit(Train1[columns], Target1)

*#predicting the second portion of the training set for cross-valdiation purpose*

pred = mlp.predict(Train2[columns])

*#finding the error for the same*

Error12 = ErrorFunction(Target2, pred, Train2[**'perishable'**])

Error22 = ErrorFunction2(Target2, pred, Train2[**'perishable'**])

print(**'\n-> Mean Squared Error(1): '**, Error21, **'Error(2)'**, Error22)

print(**'\n-> Mean Squared Log Error(1): '**, Error11, **'Error(2): '**, Error12)

*#finally predicting the unit\_sales of the testing data*

test[**'unit\_sales'**] = mlp.predict(test[columns])

cut = 0. + 1e-12 *# 0.+1e-15*

test[**'unit\_sales'**] = (np.exp(test[**'unit\_sales'**]) - 1).clip(lower=cut)

*# printing the results, this line can be commented in case you want to see the results in the form of csv*

*# print(test[['id', 'unit\_sales']])*

*# setting up the output csv file with values of the errors obtained earlier*

*# Output = 'DeepLearning'+'MQLE'+str(Error11)+'and'+str(Error12)+'MSE'+str(Error21)+'and'+str(Error22)+'.csv'*

*# Exporting the data from the program to the csv file*

*# test[['id','unit\_sales']].to\_csv(Output, index=False, float\_format='%.2f')*

*#Gradient Boosting so far this has been producing good results but not as good as random forest but at the same not bad as deep learning*

**def** GradientBoosting(Train1, Train2, Target1, Target2, columns):

print(**'\n-> Gradient Boosting'**)

gbr = GradientBoostingRegressor(learning\_rate=1, n\_estimators=100, criterion=**'friedman\_mse'**)

gbr.fit(Train1[columns], Target1)

*#predicitng for cross-validation so that we can find out the errors produced in the training and predicting*

pred = gbr.predict(Train2[columns])

Error11 = ErrorFunction(Target2, pred, Train2[**'perishable'**])

Error21 = ErrorFunction2(Target2, pred, Train2[**'perishable'**])

test[**'transactions'**] = gbr.predict(test[columns])

test[**'transactions'**] = test[**'transactions'**].clip(lower=0. + 1e-15)

columns = [col **for** col **in** Train1 **if** col **not in** [**'id'**, **'unit\_sales'**, **'perishable'**]]

Target1 = Train1[**'unit\_sales'**].values

Target2 = Train2[**'unit\_sales'**].values

gbr = GradientBoostingRegressor(learning\_rate=1, n\_estimators=100, criterion=**'friedman\_mse'**)

gbr.fit(Train1[columns], Target1)

*# predicting the second portion of the training set for cross-valdiation purpose*

pred = gbr.predict(Train2[columns])

*#finding the error for the same*

Error12 = ErrorFunction(Target2, pred, Train2[**'perishable'**])

Error22 = ErrorFunction2(Target2, pred, Train2[**'perishable'**])

print(**'\n-> Mean Squared Error(1): '**, Error21, **'Error(2): '**, Error22)

print(**'\n-> Mean Squared Log Error(1): '**, Error11, **'Error(2): '**, Error12)

*#finally predicting the unit\_sales of the testing data*

test[**'unit\_sales'**] = gbr.predict(test[columns])

cut = 0. + 1e-12 *# 0.+1e-15*

test[**'unit\_sales'**] = (np.exp(test[**'unit\_sales'**]) - 1).clip(lower=cut)

*# printing the results, this line can be commented in case you want to see the results in the form of csv*

*# print(test[['id', 'unit\_sales']])*

*# setting up the output csv file with values of the errors obtained earlier*

*# Output = 'GradientBoosting'+'MQLE'+str(Error11)+'and'+str(Error12)+'MSE'+str(Error21)+'and'+str(Error22)+'.csv'*

*# Exporting the data from the program to the csv file*

*# test[['id','unit\_sales']].to\_csv(Output, index=False, float\_format='%.2f')*

*#AdaBoost so far this has been producing good results but not as good as random forest but at the same not bad as deep learning*

**def** Adaboost(Train1, Train2, Target1, Target2, columns):

print(**'\n-> AdaBoost'**)

abr = AdaBoostRegressor(n\_estimators=50, learning\_rate=1, loss=**'linear'**, random\_state=100)

abr.fit(Train1[columns], Target1)

*#predicitng for cross-validation so that we can find out the errors produced in the training and predicting*

pred = abr.predict(Train2[columns])

Error11 = ErrorFunction(Target2, pred, Train2[**'perishable'**])

Error21 = ErrorFunction2(Target2, pred, Train2[**'perishable'**])

test[**'transactions'**] = abr.predict(test[columns])

test[**'transactions'**] = test[**'transactions'**].clip(lower=0. + 1e-15)

columns = [col **for** col **in** Train1 **if** col **not in** [**'id'**, **'unit\_sales'**, **'perishable'**]]

Target1 = Train1[**'unit\_sales'**].values

Target2 = Train2[**'unit\_sales'**].values

abr = AdaBoostRegressor(n\_estimators=100, learning\_rate=1, loss=**'linear'**, random\_state=100)

abr.fit(Train1[columns], Target1)

*# predicting the second portion of the training set for cross-valdiation purpose*

pred = abr.predict(Train2[columns])

*#finding the error for the same*

Error12 = ErrorFunction(Target2, pred, Train2[**'perishable'**])

Error22 = ErrorFunction2(Target2, pred, Train2[**'perishable'**])

print(**'\n-> Mean Squared Error(1): '**, Error21, **'Error(2): '**, Error22)

print(**'\n-> Mean Squared Log Error(1): '**, Error11, **'Error(2): '**, Error12)

*#finally predicting the unit\_sales of the testing data*

test[**'unit\_sales'**] = abr.predict(test[columns])

cut = 0. + 1e-12 *# 0.+1e-15*

test[**'unit\_sales'**] = (np.exp(test[**'unit\_sales'**]) - 1).clip(lower=cut)

*# printing the results, this line can be commented in case you want to see the results in the form of csv*

*# print(test[['id', 'unit\_sales']])*

*# setting up the output csv file with values of the errors obtained earlier*

*# Output = 'Adaboost'+'MQLE'+str(Error11)+'and'+str(Error12)+'MSE'+str(Error21)+'and'+str(Error22)+'.csv'*

*# Exporting the data from the program to the csv file*

*# test[['id','unit\_sales']].to\_csv(Output, index=False, float\_format='%.2f')*

*#This is the mean squared log error function which takes the real value, predicted value and weights as parameters*

**def** ErrorFunction(y, pred, w):

**return** metrics.mean\_squared\_log\_error(y, pred, sample\_weight=w) \*\* 0.5

*#This is the mean squared error function which takes the real value, predicted value and weights as parameters*

**def** ErrorFunction2(y, pred, w):

**return** metrics.mean\_squared\_error(y, pred, sample\_weight=w) \*\* 0.5

*#main method*

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

*# Keep this as 1 for Deep Learing,2 for Random Forest, 3 for Gradient Boosting , 4 for Adaboost*

RegressorId = 1

*# setting the data types for different fields*

DataTypes = {**'id'**: **'int64'**, **'item\_nbr'**: **'int32'**, **'store\_nbr'**: **'int8'**, **'onpromotion'**: str}

print(**'\n-> Reading Datasets...'**)

TrainingSet = pd.DataFrame

ItemSet = pd.DataFrame

TestingSet = pd.DataFrame

TransactionSet = pd.DataFrame

StoreSet = pd.DataFrame

HolidaySet = pd.DataFrame

OilSet = pd.DataFrame

*#reading the training dataset*

TrainingSet = pd.read\_csv(**'../input/train.csv'**, dtype=DataTypes, parse\_dates=[**'date'**])

*#reading the item dataset*

ItemSet = pd.read\_csv(**'../input/items.csv'**)

*#reading the testing dataset*

TestingSet = pd.read\_csv(**'../input/test.csv'**, dtype=DataTypes, parse\_dates=[**'date'**])

*#reading the transaction dataset*

TransactionSet = pd.read\_csv(**'../input/transactions.csv'**, parse\_dates=[**'date'**])

*#reading the stores dataset*

StoreSet = pd.read\_csv(**'../input/stores.csv'**)

*#reading the holiday dataset*

HolidaySet = pd.read\_csv(**'../input/holidays\_events.csv'**, dtype={**'transferred'**: str}, parse\_dates=[**'date'**])

*#reading the oil dataset*

OilSet = pd.read\_csv(**'../input/oil.csv'**, parse\_dates=[**'date'**])

print(**'\n-> Processing Datasets...'**)

*#limiting the amount of training data*

train = TrainingSet[(TrainingSet[**'date'**].dt.month == 8) & (TrainingSet[**'date'**].dt.day > 15)]

**del** TrainingSet;

gc.collect();

*#storing the real values into TargetValues*

TargetValues = train[**'unit\_sales'**].values

*#replacing the negative values*

TargetValues[TargetValues < 0.] = 0.

train[**'unit\_sales'**] = np.log1p(TargetValues)

print(**'\n-> Merging Items and Training Datasets...'**)

ItemSet = DataFrameLabelEncoder(ItemSet)

*#merging the training and item datasets*

train = pd.merge(train, ItemSet, how=**'left'**, on=[**'item\_nbr'**])

*#merging the testing and item datasets*

test = pd.merge(TestingSet, ItemSet, how=**'left'**, on=[**'item\_nbr'**])

**del** TestingSet;

gc.collect();

**del** ItemSet;

gc.collect();

print(**'\n-> Merging Transaction and Training Datasets...'**)

*#merging the training and transaction datasets*

train = pd.merge(train, TransactionSet, how=**'left'**, on=[**'date'**, **'store\_nbr'**])

*#merging the testing and transaction datasets*

test = pd.merge(test, TransactionSet, how=**'left'**, on=[**'date'**, **'store\_nbr'**])

**del** TransactionSet;

gc.collect();

TargetValues = train[**'transactions'**].values

TargetValues[TargetValues < 0.] = 0.

train[**'transactions'**] = np.log1p(TargetValues)

*#Encoding the dataset*

print(**'\n-> Encoding Store Dataset...'**)

StoreSet = DataFrameLabelEncoder(StoreSet)

print(**'\n-> Merging Store and Training Datasets '**)

*#merging the training and the store dataset*

train = pd.merge(train, StoreSet, how=**'left'**, on=[**'store\_nbr'**])

*#merging the testing and store dataset*

test = pd.merge(test, StoreSet, how=**'left'**, on=[**'store\_nbr'**])

**del** StoreSet;

gc.collect();

HolidaySet = HolidaySet[HolidaySet[**'locale'**] == **'National'**][[**'date'**, **'transferred'**]]

*#mapping values to 0/1 on the basis of the holiday being transferred or not*

HolidaySet[**'transferred'**] = HolidaySet[**'transferred'**].map({**'False'**: 0, **'True'**: 1})

print(**'\n-> Merging Holiday and Training Datasets... '**)

*#merging the training and the holiday datasets*

train = pd.merge(train, HolidaySet, how=**'left'**, on=[**'date'**])

*#merging the testing and holiday dataset*

test = pd.merge(test, HolidaySet, how=**'left'**, on=[**'date'**])

**del** HolidaySet;

gc.collect();

print(**'\n-> Merging Oil and Training Datasets...'**)

*#merging the training and oil datasets*

train = pd.merge(train, OilSet, how=**'left'**, on=[**'date'**])

*#merging the testing and oil datasets*

test = pd.merge(test, OilSet, how=**'left'**, on=[**'date'**])

**del** OilSet;

gc.collect();

print(**'\n-> Transforming Train and Test Datasets...'**)

*#transforing the dates and some more attributes of entire train dataset after merging with all datasets*

train = DataFrameTransform(train)

*#transforming the dates and some more attributes of entire test dataset after merging with all datasets*

test = DataFrameTransform(test)

columns = [col **for** col **in** train **if** col **not in** [**'id'**, **'unit\_sales'**, **'perishable'**, **'transactions'**]]

*#splitting the train datasets into 2 parts i.e one of instances that have dates other that year 2016 and rest in the other part*

Train1 = train[(train[**'year'**] != 2016)]

Train2 = train[(train[**'year'**] == 2016)]

**del** train;

gc.collect();

*#storing the transaction values of each splitted part into different variables*

Target1 = Train1[**'transactions'**].values

Target2 = Train2[**'transactions'**].values

**if** (RegressorId == 1):

*#implementing the deep learning refressor*

*# DeepLearning(Train1,Train2,Target1,Target2,columns)*

RegressorId += 1

**if** (RegressorId == 2):

*#implementing the random forest regressor*

*# RandomForest(Train1, Train2, Target1, Target2, columns)*

RegressorId += 1

**if** (RegressorId == 3):

*#implementing the gradient boosting regressor*

GradientBoosting(Train1,Train2,Target1,Target2,columns)

RegressorId += 1

**if** (RegressorId == 4):

*#implementing the adaboost regressor*

Adaboost(Train1,Train2,Target1,Target2,columns)

RegressorId += 1

print(**'\_\_The End\_\_'**)

*#program ends...*