AnshDesai / **House-Price-Prediction**

Branch: **master** **House-Price-Prediction** / **AdvancedRegression.ipynb** Find file Copy path

AnshDesai Add files via upload d44ca2b 3 days ago

1 contributor

483 lines (482 sloc) 21.7 KB Raw Blame History

In [1]:
```
import pandas as pd
import numpy as np
import warnings
from sklearn.linear_model import Lasso
from scipy.stats import skew
import xgboost as xgb
from sklearn.metrics import mean_squared_error
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

train = pd.DataFrame(pd.read_csv('train.csv'))

test = pd.read_csv('test.csv')
train.head()

test.head()

train.drop(train[train['GrLivArea']>4000].index, inplace=True)
#Map for confirming outliers of GrLivArea

#sns.heatmap(train.isnull(), cbar=False);
combined = pd.DataFrame(index = train.index)

#For label encoding columns
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
def labelencode(df, factor_df, column, fill_na=None):
    factor_df[column] = df[column]
    if fill_na is not None:
        factor_df[column].fillna(fill_na, inplace=True)
    le.fit(factor_df[column].unique())
    factor_df[column] = le.transform(factor_df[column])
    return factor_df
def feat_eng(new_name,column,value):
    combined[new_name] = (temp[column] == value) *1
    return combined
#train['OverallCond'].head()

#Preprocessing and feature engineering on train and test data
def manipulate(temp):
    combined = pd.DataFrame(index = temp.index)
 #Fill empty or null values with 0 and add columns to new data frame.
    forfill_na = ['MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','GarageArea','Bsmt
FullBath','BsmtHalfBath','GarageCars','BsmtExposure',
             'PoolArea','GarageYrBlt','GarageCars']

    for i in forfill_na:
            combined[i] = temp[i]
            combined[i].fillna(0, inplace=True)

    #Directly add columns that have numerical features to new dataframe.
    forfill = ['LotArea','OverallQual','1stFlrSF','2ndFlrSF','GrLivArea','WoodDeckSF','OpenPorchSF',
'EnclosedPorch','3SsnPorch','ScreenPorch','FullBath','HalfBath','BedroomAbvGr','KitchenAbvGr','TotRms
AbvGrd','Fireplaces',
           'MiscVal','LowQualFinSF','OverallCond','MoSold','YrSold','YearRemodAdd','YearBuilt','Overa
llQual']
    for i in forfill:
        combined[i] = temp[i]
    combined["CentralAir"] = (temp["CentralAir"] == "Y") * 1.0

    #Factorize all categorical features through label encoding and add to new dataframe.
    CV = ['MSSubClass','LotConfig','Neighborhood','Condition1','BldgType','HouseStyle','RoofStyle','F
oundation','SaleCondition']
    for i in CV:
            labelencode(temp,combined,i)

    labelencode(temp, combined, "MSZoning", "RL")
    labelencode(temp, combined, "Exterior1st", "Other")
    labelencode(temp, combined, "Exterior2nd", "Other")
    labelencode(temp, combined, "SaleType", "Oth")

    #Map all the categorical features according to values(manually selected) as provided by data desc
ription about these values.
    quality = {None: 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5}
```

```
    forquality = ['ExterQual','ExterCond','BsmtQual','BsmtCond','HeatingQC','KitchenQual','FireplaceQ
u','GarageQual','GarageCond','PoolQC']

    for i in forquality:
            combined[i] = temp[i].map(quality).astype(int)


    combined["BsmtExposure"] = temp["BsmtExposure"].map(
         {None: 0, "No": 1, "Mn": 2, "Av": 3, "Gd": 4}).astype(int)
    combined["Functional"] = temp["Functional"].map(
        {None: 0, "Sal": 1, "Sev": 2, "Maj2": 3, "Maj1": 4,
         "Mod": 5, "Min2": 6, "Min1": 7, "Typ": 8}).astype(int)
    combined["GarageFinish"] = temp["GarageFinish"].map(
        {None: 0, "Unf": 1, "RFn": 2, "Fin": 3}).astype(int)
    combined["Fence"] = temp["Fence"].map(
        {None: 0, "MnWw": 1, "GdWo": 2, "MnPrv": 3, "GdPrv": 4}).astype(int)
    combined["BsmtFinType1"] = temp["BsmtFinType1"].map(

        {None: 0, "Unf": 1, "LwQ": 2, "Rec": 3, "BLQ": 4, "ALQ": 5, "GLQ": 6}).astype(int)
    combined["BsmtFinType2"] = temp["BsmtFinType2"].map(
        {None: 0, "Unf": 1, "LwQ": 2, "Rec": 3, "BLQ": 4, "ALQ": 5, "GLQ": 6}).astype(int)
    combined["SeasonSold"] = temp["MoSold"].map(
        {12:0, 1:0, 2:0, 3:1, 4:1, 5:1,6:2, 7:2, 8:2, 9:3, 10:3, 11:3}).astype(int)

    #Feature Engineering
    #Create new columns according to columns with most features having same value and removing outlie
rs in the process.

    combined["IsRegularLotShape"] = (temp["LotShape"] == "Reg") * 1
    combined["IsLandLevel"] = (temp["LandContour"] == "Lvl") * 1

    combined["IsLandSlopeGentle"] = (temp["LandSlope"] == "Gtl") * 1
    combined["IsElectricalSBrkr"] = (temp["Electrical"] == "SBrkr") * 1

    combined["IsGarageDetached"] = (temp["GarageType"] == "Detchd") * 1

    combined["IsPavedDrive"] = (temp["PavedDrive"] == "Y") * 1
    combined["HasShed"] = (temp["MiscFeature"] == "Shed") * 1.
    combined["RecentRemodel"] = (combined["YearRemodAdd"] == combined["YrSold"]) * 1
    combined["VeryNewHouse"] = (combined["YearBuilt"] == combined["YrSold"]) * 1
    combined["Has2ndFloor"] = (combined["2ndFlrSF"] == 0) * 1
    combined["HasMasVnr"] = (combined["MasVnrArea"] == 0) * 1
    combined["HasWoodDeck"] = (combined["WoodDeckSF"] == 0) * 1
    combined["HasOpenPorch"] = (combined["OpenPorchSF"] == 0) * 1
    combined["HasEnclosedPorch"] = (combined["EnclosedPorch"] == 0) * 1
    combined["Has3SsnPorch"] = (combined["3SsnPorch"] == 0) * 1
    combined["HasScreenPorch"] = (combined["ScreenPorch"] == 0) * 1


    combined["Remodeled"] = (combined["YearRemodAdd"] != combined["YearBuilt"]) * 1

    #Replace values of features according to data description.
    combined["NewerDwelling"] = temp["MSSubClass"].replace(
        {20: 1, 30: 0, 40: 0, 45: 0,50: 0, 60: 1, 70: 0, 75: 0, 80: 0, 85: 0,90: 0, 120: 1, 150: 0, 1
60: 0, 180: 0, 190: 0})

    combined.loc[temp.Neighborhood == 'NridgHt', "Neighborhood_Good"] = 1
    combined.loc[temp.Neighborhood == 'Crawfor', "Neighborhood_Good"] = 1
    combined.loc[temp.Neighborhood == 'StoneBr', "Neighborhood_Good"] = 1
    combined.loc[temp.Neighborhood == 'Somerst', "Neighborhood_Good"] = 1
    combined.loc[temp.Neighborhood == 'NoRidge', "Neighborhood_Good"] = 1
    combined["Neighborhood_Good"].fillna(0, inplace=True)
    lot_frontage_neig = temp["LotFrontage"].groupby(temp["Neighborhood"])
    combined["LotFrontage"] = temp["LotFrontage"]

    #Fill unique feature 'LotFrontage' values according to Neighborhood it corresponds and replace by
median of values.
    for key, group in lot_frontage_neig:
        idx = (temp["Neighborhood"] == key) & (temp["LotFrontage"].isnull())
        combined.loc[idx, "LotFrontage"] = group.median()

    #Replace values of features according to data description.
    combined.LotFrontage = combined.LotFrontage.astype(int)
    combined["SaleCondprice"] = temp.SaleCondition.replace(
    {'Abnorml': 1, 'Alloca': 1, 'AdjLand': 1, 'Family': 1, 'Normal': 0, 'Partial': 0})
    combined["BoughtOffPlan"] = temp.SaleCondition.replace(
    {"Abnorml" : 0, "Alloca" : 0, "AdjLand" : 0, "Family" : 0, "Normal" : 0, "Partial" : 1})
    combined["BadHeating"] = temp.HeatingQC.replace(
    {'Ex': 0, 'Gd': 0, 'TA': 0, 'Fa': 1, 'Po': 1})
```

```
    area = ['LotFrontage','LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
           'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'GarageArea', 'WoodDeckSF',
           'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'LowQualFinSF', 'PoolArea' ]
    combined["TotalArea"] = combined[area].sum(axis=1)

    combined["TotalArea1st2nd"] = combined["1stFlrSF"] + combined["2ndFlrSF"]

    combined["Age"] = 2010 - combined["YearBuilt"]
    combined["TimeSinceSold"] = 2010 - combined["YrSold"]
    combined["SeasonSold"] = combined["MoSold"].map({12:0, 1:0, 2:0, 3:1, 4:1, 5:1,
                                                     6:2, 7:2, 8:2, 9:3, 10:3, 11:3}).astype(int)

    combined["YearsSinceRemodel"] = combined["YrSold"] - combined["YearRemodAdd"]

    rating ={1:1,2:1,3:1,4:2,5:2,6:2,7:3,8:3,9:3,10:3}
    forreplace = ["OverallQual","OverallCond","PoolQC","GarageCond","GarageQual","FireplaceQu",
        "FireplaceQu","Functional","KitchenQual","HeatingQC","BsmtFinType1","BsmtFinType2","BsmtCond"
,"BsmtQual","ExterCond","ExterQual"]
    for i in forreplace:
            combined["Rated"+i] = combined[i].replace(rating)
    #Map according to below function which seperates feature values exactly by prices of houses in th
at region.
    #Neighborhood affects prices of houses so we need to replace the categorical data by mapping them
by numerical values.
    '''median_prices = temp["SalePrice"].groupby(temp["Neighborhood"]).median().sort_values()
    def split(a, n):
    k, m = divmod(len(a), n)
    return (a[i * k + min(i, m):(i + 1) * k + min(i + 1, m)] for i in range(n))
         list(split(median_prices.values, 4))'''
    neigh_map = {"MeadowV" : 0,"IDOTRR" : 1,"BrDale" : 1,"OldTown" : 1,"Edwards" : 1, "BrkSide" : 1,
"Sawyer" : 1,  "Blueste" : 2,
           "SWISU" : 2, "NAmes" : 2, "NPkVill" : 2,   "Mitchel" : 2, "SawyerW" : 2, "Gilbert" : 2,
"NWAmes" : 2, "Blmngtn" : 3,
           "CollgCr" : 3,"ClearCr" : 3,"Crawfor" : 3,"Veenker" : 3,"Somerst" : 3,"Timber" : 3,"Stone
Br" : 4,"NoRidge" : 4,"NridgHt" : 4}
    combined["NeighborhoodBin"] = temp["Neighborhood"].map(neigh_map)
    return combined
```

In [2]:
```
#Train data is processed
new_train = manipulate(train)
```

In [3]:
```
#Replace few feature values as they are in train data.
test.loc[666, "GarageQual"] = "TA"
test.loc[666, "GarageCond"] = "TA"
test.loc[666, "GarageFinish"] = "Unf"
test.loc[666, "GarageYrBlt"] = "1980"
#test.loc[1116, "GarageType"] = np.nan
#Test data is processed
new_test = manipulate(test)
neighborhood_train = pd.DataFrame(index = train.index)
neighborhood_train["NeighborhoodBin"] = new_train["NeighborhoodBin"]
neighborhood_test = pd.DataFrame(index = test.index)
neighborhood_test["NeighborhoodBin"] = new_test["NeighborhoodBin"]
```

In [5]:
```
#Select all columns with skewness more than 0.75.
numeric_features = new_train.dtypes[new_train.dtypes != "object"].index
allskew = new_train[numeric_features].apply(lambda x: skew(x.dropna().astype(float)))
allskew = allskew[allskew > 0.75]
#sns.distplot(allskew)
allskew = allskew.index
```

In [6]:
```
#Take log(1+numericalfeature) to remove skewness
new_train[allskew] = np.log1p(new_train[allskew])
new_test[allskew] = np.log1p(new_test[allskew])
```

In [7]:
```
#Scaling every numerical column so that it provides more efficiency to our model
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(new_train[numeric_features])
scaled = scaler.transform(new_train[numeric_features])

for i, col in enumerate(numeric_features):
    new_train[col] = scaled[:, i]
scaled = scaler.transform(new_test[numeric_features])
for i, col in enumerate(numeric_features):
    new_test[col] = scaled[:, i]
```

In [8]:
```
#Encoding using sklearn one-hot-encoder can also perform using pandas dummies variable method
#Need to drop one of the dummy variable column to avoid multicollinearity.
def onehot(onehot_en, df, column, fill_na, drop_nm):
    onehot_en[column] = df[column]
    if fill_na is not None:
        onehot_en[column].fillna(fill_na, inplace=True)

    dummies = pd.get_dummies(onehot_en[column], prefix="_" + column)

    onehot_en = onehot_en.join(dummies)
    onehot_en = onehot_en.drop([column], axis=1)
    return onehot_en
```

In [9]:
```
def ohe_df(arb):
    onehot_en = pd.DataFrame(index = arb.index)
    feature = ['MSSubClass','MSZoning','LotConfig','Neighborhood','Condition1','BldgType','HouseStyl
e','RoofStyle','Exterior1st',
               'Exterior2nd','Foundation','SaleType','SaleCondition','LotShape','LandContour','LandSlop
e','Electrical','GarageType',
                'PavedDrive','MiscFeature','Street','Alley','Condition2','RoofMatl','Heating']
    feat_value = [None,'RL', None, None, None, None, None, None,'VinylSd','VinylSd',None,'WD','Norma
l',None,None,None,'SBrkr'
                  'None',None,'None',None,'None',None,None,None]
    drop = ['40','RH','FR3','OldTown','RRNe','2fmCon','1.5Unf','Shed','CBlock','CBlock','Wood','Oth',
'AdjLand','IR3','Low','Sev'
                   'FuseP','CarPort','P','Othr','Grvl','Grvl','PosA','WdShake','Wall']
    for (i,j,k) in zip (feature,feat_value,drop):
        onehot_en = onehot(onehot_en,arb,i,j,k)
    numerical_fe = ['ExterQual','ExterCond','BsmtQual','BsmtCond','HeatingQC','KitchenQual','Fireplac
eQu','GarageQual',
                'GarageCond','PoolQC','BsmtExposure','BsmtFinType1','BsmtFinType2','Functional','Garage
Finish','Fence','MoSold']
    numerical_vl = ['None','None','None','None','None','TA','None','None','None','None','None','None'
,'None','Typ','None',
                    'None',None]
    drop_num_dumm = ['Ex','Ex','Ex','Ex','Ex','Ex','Ex','Ex','Ex','Ex','Gd','GLQ','GLQ','Typ','Fin',
'MnPrv',None]
    for (i,j,k) in zip(numerical_fe,numerical_vl,drop_num_dumm):
        onehot_en = onehot(onehot_en,arb,i,j,k)

    year_map = pd.concat(pd.Series("YearBin" + str(i+1), index=range(1871+i*20,1891+i*20)) for i in r
ange(0, 7))
    yearbin = pd.DataFrame(index = arb.index)
    yearbin["GarageYrBltBin"] = arb.GarageYrBlt.map(year_map)
    yearbin["GarageYrBltBin"].fillna("NoGarage", inplace=True)
    yearbin["YearBuiltBin"] = arb.YearBuilt.map(year_map)
    yearbin["YearRemodAddBin"] = arb.YearRemodAdd.map(year_map)
    onehot_en = onehot(onehot_en, yearbin, "GarageYrBltBin", None, None)
    onehot_en = onehot(onehot_en, yearbin, "YearBuiltBin", None, None)
    onehot_en = onehot(onehot_en, yearbin, "YearRemodAddBin", None, None)


    return onehot_en
```

In [10]:
```
#Joining columns of newtrain, newtest with onehot encoded columns
onehot_en = ohe_df(train)
onehot_en = onehot(onehot_en, neighborhood_train, "NeighborhoodBin", None, None)
new_train = new_train.join(onehot_en)
```

In [11]:
```
onehot_ent = ohe_df(test)
onehot_ent = onehot(onehot_en, neighborhood_test, "NeighborhoodBin", None, None)
new_test = new_test.join(onehot_ent)
```

In [12]:
```
#Drop columns that are not in either train or test to curb overfitting our model.
drop_train = ["_Exterior1st_ImStucc", "_Exterior1st_Stone","_Exterior2nd_Other","_HouseStyle_2.5Fin",
              "_RoofMatl_Membran", "_RoofMatl_Metal", "_RoofMatl_Roll","_Condition2_RRAe", "_Condition2
_RRAn", "_Condition2_RRNn",
              "_Electrical_Mix", "_MiscFeature_TenC","_Condition2_PosN", "_MSZoning_C (all)","_MSSubCla
ss_160"]
new_train.drop(drop_train, axis=1, inplace=True)

drop_test = [ "_Condition2_PosN", "_MSZoning_C (all)","_MSSubClass_160", "_Exterior1st_ImStucc", "_Ne
ighborhoodBin_2.0",
              "_NeighborhoodBin_3.0","_Condition2_RRNn", "_HouseStyle_2.5Fin", "_NeighborhoodBin_0.0",
"_Electrical_Mix",
              "_Condition2_RRAn", "_Exterior2nd_Other", "_Condition2_RRAe", "_RoofMatl_Metal", "_RoofM
```

```
atl_Membran",
            "_MiscFeature_TenC", "_RoofMatl_Roll", "_NeighborhoodBin_4.0", "_Exterior1st_Stone","_Ne
ighborhoodBin_1.0"
]
new_test.drop(drop_test,axis=1,inplace=True)
new_test = new_test.fillna(new_test.mean())
```

In [13]:
```
#As mentioned in description the sale price will be calculated as log of SalePrice
label = pd.DataFrame(columns=["SalePrice"])
label["SalePrice"] = np.log(train["SalePrice"])
```

In [14]:
```
print("Training set size:", new_train.shape)
print("Test set size:", new_test.shape)
```

```
Training set size: (1456, 393)
Test set size: (1459, 393)
```

In [129]:
```
#Xgboost model parameters are scaled using GridSearch
regr = xgb.XGBRegressor(
                 colsample_bytree=0.2,
                 gamma=0,
                 learning_rate=0.6,
                 max_depth=5,
                 min_child_weight=1.5,
                 n_estimators=7200,
                 reg_alpha=0.9,
                 reg_lambda=0.6,
                 subsample=0.2,
                 seed=42,
                 silent=1)
xg_model = regr.fit(new_train,label)
pred_xgboost = regr.predict(new_train)
print('XGBOOST score:',np.sqrt(mean_squared_error(label,pred_xgboost)))
```

```
XGBOOST score: 0.024377922839727097
```

In [124]:
```
#Lasso Model to support XGboost
lasso_mod = Lasso(alpha=.000009, max_iter=50000)
lasso_mod.fit(new_train,label)
pred_lasso =lasso_mod.predict(new_train)
```

In [130]:
```
pred_comb = (pred_xgboost + pred_lasso)/2
```

In [127]:
```
pred_combined = (regr.predict(new_test) + lasso_mod.predict(new_test))/2
pred_combined = np.exp(pred_combined)
prediction_final = pd.DataFrame(pred_combined, index = test["Id"], columns=["SalePrice"])
prediction_final.to_csv('prediction.csv', header=True, index_label='Id')
```