House Price Prediction-using one-hot

March 4, 2019

```
In [1]: import pandas as pd
        import re
        from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
        from IPython.display import display
        import numpy as np
        import math
        from sklearn import metrics
        from pandas.api.types import is_string_dtype, is_numeric_dtype
        import matplotlib.pyplot as plt
        from sklearn.ensemble import forest
        import scipy
        from scipy.cluster import hierarchy as hc
```

```
In [52]: def rmse(x,y):
             return math.sqrt(((x-y)**2).mean())

         def print_score(m):
             res = [rmse(m.predict(X_train), y_train), rmse(m.predict(X_valid), y_valid),
                         m.score(X_train, y_train), m.score(X_valid, y_valid)]
             if hasattr(m, 'oob_score_'): res.append(m.oob_score_)
             print(res)

         def split_vals(a,n):
             return a[:n].copy(), a[n:].copy()

         def get_oob(df):
             m = RandomForestRegressor(n_estimators=40, min_samples_leaf=5, max_features=0.5, r
             x, _ = split_vals(df, n_trn)
             m.fit(x, y_train)
             return m.oob_score_

         def add_datepart(df, fldname, drop=True, time=False):
             fld = df[fldname]
             fld_dtype = fld.dtype
             if isinstance(fld_dtype, pd.core.dtypes.dtypes.DatetimeTZDtype):
                 fld_dtype = np.datetime64
```

```python
    if not np.issubdtype(fld_dtype, np.datetime64):
        df[fldname] = fld = pd.to_datetime(fld, infer_datetime_format=True)
    targ_pre = re.sub('[Dd]ate$', '', fldname)
    attr = ['Year', 'Month', 'Week', 'Day', 'Dayofweek', 'Dayofyear',
            'Is_month_end', 'Is_month_start', 'Is_quarter_end', 'Is_quarter_start', 'I
    if time: attr = attr + ['Hour', 'Minute', 'Second']
    for n in attr: df[targ_pre + n] = getattr(fld.dt, n.lower())
    df[targ_pre + 'Elapsed'] = fld.astype(np.int64) // 10 ** 9
    if drop: df.drop(fldname, axis=1, inplace=True)

def train_cats(df):
    for n,c in df.items():
        if is_string_dtype(c): df[n] = c.astype('category').cat.as_ordered()

def fix_missing(df, col, name, na_dict):
    if is_numeric_dtype(col):
        if pd.isnull(col).sum() or (name in na_dict):
            df[name+'_na'] = pd.isnull(col)
            filler = na_dict[name] if name in na_dict else col.median()
            df[name] = col.fillna(filler)
            na_dict[name] = filler
    return na_dict

def proc_df(df, y_fld=None, skip_flds=None, ignore_flds=None, do_scale=False, na_dict=
            preproc_fn=None, max_n_cat=None, subset=None, mapper=None):
    if not ignore_flds: ignore_flds=[]
    if not skip_flds: skip_flds=[]
    if subset: df = get_sample(df,subset)
    else: df = df.copy()
    ignored_flds = df.loc[:, ignore_flds]
    df.drop(ignore_flds, axis=1, inplace=True)
    if preproc_fn: preproc_fn(df)
    if y_fld is None: y = None
    else:
        if not is_numeric_dtype(df[y_fld]): df[y_fld] = df[y_fld].cat.codes
        y = df[y_fld].values
        skip_flds += [y_fld]
    df.drop(skip_flds, axis=1, inplace=True)

    if na_dict is None: na_dict = {}
    else: na_dict = na_dict.copy()
    na_dict_initial = na_dict.copy()
    for n,c in df.items(): na_dict = fix_missing(df, c, n, na_dict)
    if len(na_dict_initial.keys()) > 0:
        df.drop([a + '_na' for a in list(set(na_dict.keys()) - set(na_dict_initial.key
    if do_scale: mapper = scale_vars(df, mapper)
    for n,c in df.items(): numericalize(df, c, n, max_n_cat)
    df = pd.get_dummies(df, dummy_na=True)
```

```
        df = pd.concat([ignored_flds, df], axis=1)
        res = [df, y, na_dict]
        if do_scale: res = res + [mapper]
        return res

    def numericalize(df, col, name, max_n_cat):
        if not is_numeric_dtype(col) and ( max_n_cat is None or col.nunique()>max_n_cat):
            df[name] = col.cat.codes+1
```

In [15]:
```
#Import training dataset
df_raw = pd.read_csv('train.csv', low_memory=False)
df_test = pd.read_csv('test.csv', low_memory=False)
df_training = df_raw.copy()
```

In [16]:
```
#Change all object variable type to category
train_cats(df_training)
```

In [17]:
```
# Add age column and age of renovation
df_training['house_age'] = df_training['YrSold'] - df_training['YearBuilt']
df_training['renovation_age'] = df_training['YrSold'] - df_training['YearRemodAdd']
```

In [14]:
```
cat_cols = list(df_training.select_dtypes(include=['category']).columns)
for c in cat_cols:
    print(c, len(df_training[c].value_counts()))
```

```
MSZoning 5
Street 2
Alley 2
LotShape 4
LandContour 4
Utilities 2
LotConfig 5
LandSlope 3
Neighborhood 25
Condition1 9
Condition2 8
BldgType 5
HouseStyle 8
RoofStyle 6
RoofMatl 8
Exterior1st 15
Exterior2nd 16
MasVnrType 4
ExterQual 4
ExterCond 5
Foundation 6
BsmtQual 4
BsmtCond 4
BsmtExposure 4
```

```
BsmtFinType1 6
BsmtFinType2 6
Heating 6
HeatingQC 5
CentralAir 2
Electrical 5
KitchenQual 4
Functional 7
FireplaceQu 5
GarageType 6
GarageFinish 3
GarageQual 5
GarageCond 5
PavedDrive 3
PoolQC 3
Fence 4
MiscFeature 4
SaleType 9
SaleCondition 6
```

In [18]:
```
print(df_training.shape)
df, y, _ = proc_df(df_training, 'SalePrice', max_n_cat=16)
print(df.shape, y.shape)
```

```
(1460, 83)
(1460, 312) (1460,)
```

In [19]:
```
df.head()
```

Out[19]:
```
   Id  MSSubClass  LotFrontage  LotArea  Neighborhood  OverallQual  \
0   1          60         65.0     8450             6            7
1   2          20         80.0     9600            25            6
2   3          60         68.0    11250             6            7
3   4          70         60.0     9550             7            7
4   5          60         84.0    14260            16            8

   OverallCond  YearBuilt  YearRemodAdd  MasVnrArea          ...           \
0            5       2003          2003       196.0          ...
1            8       1976          1976         0.0          ...
2            5       2001          2002       162.0          ...
3            5       1915          1970         0.0          ...
4            5       2000          2000       350.0          ...

   SaleType_Oth  SaleType_WD  SaleType_nan  SaleCondition_Abnorml  \
0             0            1             0                      0
1             0            1             0                      0
2             0            1             0                      0
```

```
3              0            1             0                      1
4              0            1             0                      0

   SaleCondition_AdjLand  SaleCondition_Alloca  SaleCondition_Family  \
0                      0                     0                     0
1                      0                     0                     0
2                      0                     0                     0
3                      0                     0                     0
4                      0                     0                     0

   SaleCondition_Normal  SaleCondition_Partial  SaleCondition_nan
0                     1                      0                  0
1                     1                      0                  0
2                     1                      0                  0
3                     0                      0                  0
4                     1                      0                  0

[5 rows x 312 columns]
```

In [24]:
```
df.columns
```

Out[24]:
```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'Neighborhood',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea',
       ...
       'SaleType_Oth', 'SaleType_WD', 'SaleType_nan', 'SaleCondition_Abnorml',
       'SaleCondition_AdjLand', 'SaleCondition_Alloca', 'SaleCondition_Family',
       'SaleCondition_Normal', 'SaleCondition_Partial', 'SaleCondition_nan'],
      dtype='object', length=312)
```

In [22]:
```
y = np.log(y)
```

In [20]:
```
#Check for nulls
missing = pd.DataFrame({'missing' : df.isnull().sum()})
missing[missing['missing'] > 0]
```

Out[20]:
```
Empty DataFrame
Columns: [missing]
Index: []
```

In [25]:
```
# Split training set into training and validation with validation set = 450 rows (30%
n_trn = len(df) - 450
X_train, X_valid = split_vals(df, n_trn)
y_train, y_valid = split_vals(y, n_trn)
```

In [27]:
```
X_train.shape, X_valid.shape
```

Out[27]:
```
((1010, 312), (450, 312))
```

In [32]:
```
#Run Randomforest and print score for various values of max_features
for i in ['auto', 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]:
```

```
            m = RandomForestRegressor(n_estimators=40, max_features=i, n_jobs=-1)
            m.fit(X_train, y_train)
            print(i)
            print_score(m)
```

```
auto
[0.057264107396910756, 0.14595146151734822, 0.9801132170184275, 0.8552438890099305]
0.1
[0.05670356764389828, 0.14700253010301412, 0.9805006420256878, 0.8531514611865809]
0.2
[0.0557545106711805, 0.14281397518269767, 0.9811479075650249, 0.8614005758410758]
0.3
[0.05714857301985468, 0.1357240990947264, 0.9801933820431021, 0.8748202887201635]
0.4
[0.05777141897791424, 0.13780891055881828, 0.9797592960415762, 0.8709450671318838]
0.5
[0.05629982495900144, 0.1411070327871255, 0.9807773334457079, 0.864693914587451]
0.6
[0.057703837067260524, 0.13919972735439026, 0.9798066241277498, 0.8683269851079916]
0.7
[0.056224303292726624, 0.1423688505158308, 0.9808288701643995, 0.8622632067988136]
0.8
[0.05549891367359145, 0.13992479068123698, 0.981320359741019, 0.8669516962138775]
0.9
[0.05685607582535505, 0.14539225228152958, 0.9803956112177733, 0.8563510225248484]
```

In [33]:
```
# Score for 0.3 seems to be the best
m = RandomForestRegressor(n_estimators=40, max_features=0.3, n_jobs=-1)
m.fit(X_train, y_train)
print_score(m)
```

```
[0.05806519282153965, 0.13897303371465622, 0.9795529203627539, 0.8687555079208872]
```

In [60]:
```
#Get base OOB score
print(get_oob(df))
```

```
0.8582961292551443
```

In [34]:
```
#Feature importance
feature_importance = pd.DataFrame({'Feature' : X_train.columns, 'Importance' : m.featu
feature_importance.sort_values('Importance', ascending=False, inplace=True)
feature_importance.head(30)
```

Out[34]:

	Feature	Importance
5	OverallQual	0.209184
17	GrLivArea	0.161286

38	house_age	0.119836
27	GarageCars	0.041943
14	1stFlrSF	0.041552
13	TotalBsmtSF	0.036160
7	YearBuilt	0.031264
28	GarageArea	0.024823
20	FullBath	0.023187
169	ExterQual_TA	0.022397
3	LotArea	0.021524
10	BsmtFinSF1	0.018325
179	Foundation_PConc	0.015738
39	renovation_age	0.012547
2	LotFrontage	0.012495
25	Fireplaces	0.011905
253	FireplaceQu_nan	0.011549
8	YearRemodAdd	0.010975
15	2ndFlrSF	0.010809
24	TotRmsAbvGrd	0.008016
184	BsmtQual_Ex	0.007911
4	Neighborhood	0.005973
226	CentralAir_N	0.005942
9	MasVnrArea	0.005892
6	OverallCond	0.005346
43	MSZoning_C (all)	0.005060
12	BsmtUnfSF	0.005057
26	GarageYrBlt	0.004871
255	GarageType_Attchd	0.003917
227	CentralAir_Y	0.003913

In [57]:
```
#Try different cut off points of the importance and check the score
for i in [0.1, 0.04, 0.03, 0.02, 0.01, 0.008, 0.006, 0.005, 0.004, 0.003, 0.002]:
    important_features = feature_importance[feature_importance['Importance'] > i]
    df_important = df[important_features['Feature']]

    print(i, '-', get_oob(df_important))
```

```
0.1 - 0.7934727649696957
0.04 - 0.8212366933118291
0.03 - 0.8240322501602082
0.02 - 0.8328022143079075
0.01 - 0.8565203403272628
0.008 - 0.8514816870340083
0.006 - 0.8599841015524083
0.005 - 0.8642630065229542
0.004 - 0.8604090745475325
0.003 - 0.8580309161888017
0.002 - 0.8590092788234875
```

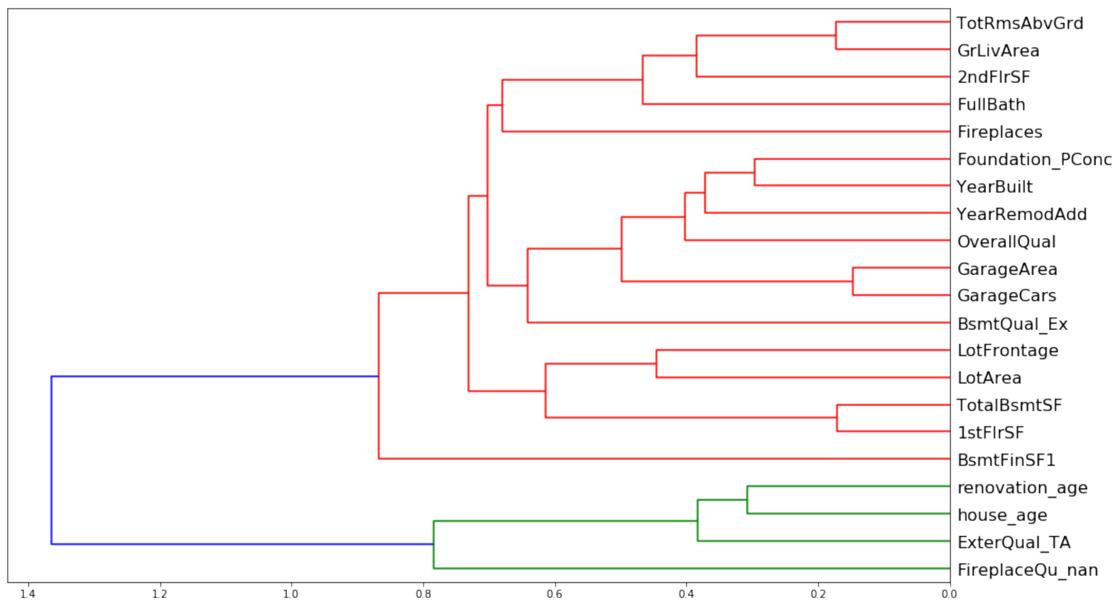
```
In [61]: #Best seems to be 0.005
         important_features = feature_importance[feature_importance['Importance'] > 0.005]
         df_important = df[important_features['Feature']]

         #Once again create training and validation sets and re-run random forest with other p
         X_train, X_valid = split_vals(df_important, n_trn)
         y_train, y_valid = split_vals(y, n_trn)

         m = RandomForestRegressor(n_estimators=40, max_features=0.3, n_jobs=-1)
         m.fit(X_train, y_train)
         print(i)
         print_score(m)
```

```
0.002
[0.054876910172810454, 0.1385870581450863, 0.9817367171819165, 0.8694835170907944]
```

```
In [40]: #Find correlated features
         #Detect and remove redundant features
         #Draw dendogram of feature clusters
         corr = np.round(scipy.stats.spearmanr(df_important).correlation, 4)
         corr_condensed = hc.distance.squareform(1-corr)
         z = hc.linkage(corr_condensed, method='average')
         fig = plt.figure(figsize=(16,10))
         dendrogram = hc.dendrogram(z, labels=df_important.columns, orientation='left', leaf_fo
         plt.show()
```




```
In [43]: cluster_pairs = ['TotRmsAbvGrd', 'GrLivArea', 'Foundation_PConc', 'YearBuilt', 'Garage
         #'LotFrontage', 'LotArea', '1stFlrSF', 'TotalBsmtSF', 'Fireplaces', 'FireplaceQu', 'G
```

In [59]:
```
#Run model after dropping each of these columns
for c in cluster_pairs:
    print(c, '-', get_oob(df_important))
```

```
TotRmsAbvGrd - 0.8583331928429265
GrLivArea - 0.8591550193121151
Foundation_PConc - 0.8559710667867992
YearBuilt - 0.8641688717764191
GarageArea - 0.8551094571006468
GarageCars - 0.8571219276534079
LotFrontage - 0.8598826033035669
LotArea - 0.8531096063024851
1stFlrSF - 0.8609756996598041
TotalBsmtSF - 0.853666174278166
renovation_age - 0.853231850719235
house_age - 0.8624916473627491
```

In [62]:
```
to_drop = ['GrLivArea', 'YearBuilt', 'GarageCars', 'LotFrontage', 'LotFrontage']
```

In [51]:
```
X_train, X_valid = split_vals(df_important.drop(to_drop, axis=1), n_trn)
y_train, y_valid = split_vals(y, n_trn)

m = RandomForestRegressor(n_estimators=40, max_features=0.3, n_jobs=-1)
m.fit(X_train, y_train)
print_score(m)
```

```
[0.05881662304477783, 0.14363903282749935, 0.9790202786770198, 0.859794530913184]
```

In [63]:
```
df_keep = df_important.drop(to_drop, axis=1)
```

In [64]:
```
df_keep.shape
```

Out[64]:
```
(1460, 21)
```

In [66]:
```
X_train, X_valid = split_vals(df_keep, n_trn)
y_train, y_valid = split_vals(y, n_trn)

m = RandomForestRegressor(n_estimators=160, max_features=0.3, n_jobs=-1)
m.fit(X_train, y_train)
print_score(m)
```

```
[0.05195797137929255, 0.13227507751203346, 0.9836279178153159, 0.8811015864920875]
```

In [67]:
```
#Now lets apply the rf on training set and predict the test set
#Separate out the dependent variable
df_training = df_raw.copy()
y = np.log(df_training['SalePrice'])
df_training = df_training.drop('SalePrice', axis=1)
```

In [88]:
```
df_training.shape, df_test.shape, y.shape
```

Out[88]:
```
((1460, 82), (1459, 80), (1460,))
```

In [69]:
```
#Concatenate training and test and process together
df_train_test = df_training.append(df_test)
```

In [71]:
```
df_train_test.shape
```

Out[71]:
```
(2919, 80)
```

In [74]:
```
#Change all object variable type to category
train_cats(df_train_test)
# Add age column and age of renovation
df_train_test['house_age'] = df_train_test['YrSold'] - df_train_test['YearBuilt']
df_train_test['renovation_age'] = df_train_test['YrSold'] - df_train_test['YearRemodAd
#Change all category to codes+1
cat_cols = list(df_train_test.select_dtypes(include=['category']).columns)  #Above Us
for col in cat_cols:
    s = df_train_test[col]
    df_train_test[col] = s.cat.codes+1
```

In [84]:
```
#Check for nulls
missing = pd.DataFrame({'missing' : df_train_test.isnull().sum()})
missing_nonzero = missing[missing['missing'] > 0]
```

In [85]:
```
missing_nonzero.index
```

Out[85]:
```
Index([], dtype='object')
```

In [77]:
```
num_cols = list(df_train_test.select_dtypes(include=['number']).columns)  #Above Usag
#for col in cat_cols:
```

In [83]:
```
for col in missing_nonzero.index:
    df_train_test[col+'_nan'] = df_train_test[col].isnull()
    df_train_test[col].fillna(df_train_test[col].median(), inplace=True)
```

In [89]:
```
#Use only columns in df_keep
df_train_test = df_train_test[df_keep.columns]
```

In [90]:
```
df_train_test.shape
```

Out[90]:
```
(2919, 21)
```

In [91]:
```
#Split into training and test
X_train = df_train_test.head(1460)
X_test = df_train_test.tail(1459)
```

In [92]:
```
#Run rf on whole dataset and predict y_test
m = RandomForestRegressor(n_estimators=40, max_features=0.3, n_jobs=-1)
m.fit(X_train, y)
```

Out[92]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=0.3, max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=40, n_jobs=-1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [94]:
```
pred = m.predict(X_test)
pred_prices = np.exp(pred)
pred_prices
```

Out[94]:
```
array([125324.27362609, 150673.54709622, 179887.43149293, ...,
       159432.13059774, 117066.25938073, 235648.4548463 ])
```

In [95]:
```
df_test.head()
```

Out[95]:
```
     Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0  1461          20       RH         80.0    11622   Pave   NaN      Reg
1  1462          20       RL         81.0    14267   Pave   NaN      IR1
2  1463          60       RL         74.0    13830   Pave   NaN      IR1
3  1464          60       RL         78.0     9978   Pave   NaN      IR1
4  1465         120       RL         43.0     5005   Pave   NaN      IR1

  LandContour Utilities      ...       ScreenPorch PoolArea PoolQC  Fence  \
0         Lvl    AllPub      ...               120        0    NaN  MnPrv
1         Lvl    AllPub      ...                 0        0    NaN    NaN
2         Lvl    AllPub      ...                 0        0    NaN  MnPrv
3         Lvl    AllPub      ...                 0        0    NaN    NaN
4         HLS    AllPub      ...               144        0    NaN    NaN

  MiscFeature MiscVal MoSold  YrSold  SaleType  SaleCondition
0         NaN       0      6    2010        WD         Normal
1        Gar2   12500      6    2010        WD         Normal
2         NaN       0      3    2010        WD         Normal
3         NaN       0      6    2010        WD         Normal
4         NaN       0      1    2010        WD         Normal

[5 rows x 80 columns]
```

In [96]:
```
#Prepare the submission df
df_submit = pd.DataFrame({'Id' : df_test.Id, 'SalePrice' : pred_prices})
```

In [98]:
```
df_submit, df_submit.shape
```

Out[98]:
```
(        Id      SalePrice
 0     1461  125324.273626
 1     1462  150673.547096
 2     1463  179887.431493
 3     1464  184931.374446
```

4	1465	190257.655318
5	1466	187669.870873
6	1467	167416.179455
7	1468	175606.755452
8	1469	181246.917042
9	1470	121499.489667
10	1471	202246.618938
11	1472	94281.135088
12	1473	99562.851254
13	1474	153940.736391
14	1475	121167.165278
15	1476	380216.489613
16	1477	240377.670441
17	1478	303460.410397
18	1479	287288.585949
19	1480	451732.204481
20	1481	294000.382304
21	1482	204681.078464
22	1483	180567.078641
23	1484	168136.337112
24	1485	175031.198112
25	1486	200374.951584
26	1487	318386.769830
27	1488	242939.210596
28	1489	209274.501629
29	1490	211039.691761
...	...	...
1429	2890	78073.653099
1430	2891	133036.688645
1431	2892	65376.336695
1432	2893	99375.637850
1433	2894	58641.692912
1434	2895	310870.847885
1435	2896	280328.322546
1436	2897	195818.291985
1437	2898	147978.059581
1438	2899	228075.100039
1439	2900	156291.235439
1440	2901	188352.217982
1441	2902	204356.630467
1442	2903	362091.916815
1443	2904	327557.587045
1444	2905	102716.513498
1445	2906	200862.151793
1446	2907	112676.234972
1447	2908	128097.788562
1448	2909	140846.027189
1449	2910	82892.945410

```
1450  2911   85481.821454
1451  2912  141570.930247
1452  2913   88765.206858
1453  2914   81818.252893
1454  2915   83659.602453
1455  2916   86910.674027
1456  2917  159432.130598
1457  2918  117066.259381
1458  2919  235648.454846

[1459 rows x 2 columns], (1459, 2))
```

In [99]:
```
df_submit.to_csv('submission.csv')
```