Blue books for bulldozers

The goal of the notebook is to predict the sale price of a particular piece of heavy equiment at auction based on it's usage, equipment type, and configuaration. The data is sourced from auction result postings and includes information on usage and equipment configurations.

In [1]:
```
%load_ext autoreload
%autoreload 2

%matplotlib inline
```

In [2]:
```
from fastai.imports import *
from fastai.structured import *
```

```
/home/amit/anaconda3/lib/python3.7/site-packages/sklearn/uti
ls/deprecation.py:143: FutureWarning: The sklearn.ensemble.f
orest module is  deprecated in version 0.22 and will be remo
ved in version 0.24. The corresponding classes / functions s
hould instead be imported from sklearn.ensemble. Anything th
at cannot be imported from sklearn.ensemble is now part of t
he private API.
  warnings.warn(message, FutureWarning)
```

In [3]:
```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor, RandomFor
from IPython.display import display
import numpy as np
from sklearn import metrics
```

In [4]:
```
PATH = 'data/'
!ls {PATH}
```

```
Train.csv  Valid.csv
```

In [5]:
```
df_raw = pd.read_csv(f'{PATH}Train.csv', low_memory=False, par
```

In [6]:
```
def display_all(df):
    with pd.option_context('display.max_rows', 1000):
        with pd.option_context('display.max_columns', 1000):
            display(df)
```

In [7]:
```
display_all(df_raw.tail().transpose())
```

	401120	401121	401122	401123	4011
SalesID	6333336	6333337	6333338	6333341	63334
SalePrice	10500	11000	11500	9000	77
MachineID	1840702	1830472	1887659	1903570	19269
ModelID	21439	21439	21439	21435	214
datasource	149	149	149	149	1
auctioneerID	1	1	1	2	
YearMade	2005	2005	2005	2005	20
MachineHoursCurrentMeter	NaN	NaN	NaN	NaN	Na
UsageBand	NaN	NaN	NaN	NaN	Na
saledate	2011-11-02 00:00:00	2011-11-02 00:00:00	2011-11-02 00:00:00	2011-10-25 00:00:00	2011-10- 00:00:
fiModelDesc	35NX2	35NX2	35NX2	30NX	30N
fiBaseModel	35	35	35	30	
fiSecondaryDesc	NX	NX	NX	NX	N
fiModelSeries	2	2	2	NaN	Na
fiModelDescriptor	NaN	NaN	NaN	NaN	Na
ProductSize	Mini	Mini	Mini	Mini	Mi
fiProductClassDesc	Hydraulic Excavator, Track - 3.0 to 4.0 Metric...	Hydraulic Excavator, Track - 3.0 to 4.0 Metric...	Hydraulic Excavator, Track - 3.0 to 4.0 Metric...	Hydraulic Excavator, Track - 2.0 to 3.0 Metric...	Hydrau Excavat Track - 2 to 3 Metric
state	Maryland	Maryland	Maryland	Florida	Flori
ProductGroup	TEX	TEX	TEX	TEX	TE
ProductGroupDesc	Track Excavators	Track Excavators	Track Excavators	Track Excavators	Tra Excavato
Drive_System	NaN	NaN	NaN	NaN	Na
Enclosure	EROPS	EROPS	EROPS	EROPS	EROF
Forks	NaN	NaN	NaN	NaN	Na
Pad_Type	NaN	NaN	NaN	NaN	Na
Ride_Control	NaN	NaN	NaN	NaN	Na
Stick	NaN	NaN	NaN	NaN	Na
Transmission	NaN	NaN	NaN	NaN	Na
Turbocharged	NaN	NaN	NaN	NaN	Na
Blade_Extension	NaN	NaN	NaN	NaN	Na

In [41]:
```
df_raw.describe()
```

Out[41]:

	SalesID	SalePrice	MachineID	ModelID	datasource
count	4.011250e+05	401125.000000	4.011250e+05	401125.000000	401125.000000
mean	1.919713e+06	10.103096	1.217903e+06	6889.702980	134.665810
std	9.090215e+05	0.693621	4.409920e+05	6221.777842	8.962237
min	1.139246e+06	8.465900	0.000000e+00	28.000000	121.000000
25%	1.418371e+06	9.581904	1.088697e+06	3259.000000	132.000000
50%	1.639422e+06	10.085809	1.279490e+06	4604.000000	132.000000
75%	2.242707e+06	10.596635	1.468067e+06	8724.000000	136.000000
max	6.333342e+06	11.863582	2.486330e+06	37198.000000	172.000000

In [8]:
```
df_raw.SalePrice = np.log(df_raw.SalePrice) #convert sale pri
```

In [9]:
```
fld=df_raw.saledate
```

In [10]:
```
add_datepart(df_raw,'saledate')
df_raw.saleYear.head()
```

Out[10]:
```
0    2006
1    2004
2    2004
3    2011
4    2009
Name: saleYear, dtype: int64
```

In [11]:
```
df_raw.columns
```

Out[11]:
```
Index(['SalesID', 'SalePrice', 'MachineID', 'ModelID', 'data
source',
       'auctioneerID', 'YearMade', 'MachineHoursCurrentMeter
', 'UsageBand',
       'fiModelDesc', 'fiBaseModel', 'fiSecondaryDesc', 'fiM
odelSeries',
       'fiModelDescriptor', 'ProductSize', 'fiProductClassDe
sc', 'state',
       'ProductGroup', 'ProductGroupDesc', 'Drive_System', '
Enclosure',
       'Forks', 'Pad_Type', 'Ride_Control', 'Stick', 'Transm
ission',
       'Turbocharged', 'Blade_Extension', 'Blade_Width', 'En
closure_Type',
       'Engine_Horsepower', 'Hydraulics', 'Pushblock', 'Ripp
er', 'Scarifier',
       'Tip_Control', 'Tire_Size', 'Coupler', 'Coupler_Syste
m',
       'Grouser_Tracks', 'Hydraulics_Flow', 'Track_Type',
       'Undercarriage_Pad_Width', 'Stick_Length', 'Thumb', '
Pattern_Changer',
       'Grouser_Type', 'Backhoe_Mounting', 'Blade_Type', 'Tr
avel_Controls',
       'Differential_Type', 'Steering_Controls', 'saleYear',
'saleMonth',
       'saleWeek', 'saleDay', 'saleDayofweek', 'saleDayofyea
r',
       'saleIs_month_end', 'saleIs_month_start', 'saleIs_qua
rter_end',
       'saleIs_quarter_start', 'saleIs_year_end', 'saleIs_ye
ar_start',
       'saleElapsed'],
      dtype='object')
```

In [12]:
```
train_cats(df_raw)
```

In [13]:
```
df_raw.UsageBand.cat.categories
```

Out[13]:
```
Index(['High', 'Low', 'Medium'], dtype='object')
```

In [14]:
```
df_raw.UsageBand.cat.set_categories(['High', 'Medium', 'Low']
```

In [15]:
```
display_all(df_raw.isnull().sum().sort_index()/len(df_raw))
```

```
Backhoe_Mounting            0.803872
Blade_Extension             0.937129
Blade_Type                  0.800977
Blade_Width                 0.937129
Coupler                     0.466620
Coupler_System              0.891660
Differential_Type           0.826959
Drive_System                0.739829
Enclosure                   0.000810
Enclosure_Type              0.937129
Engine_Horsepower           0.937129
Forks                       0.521154
Grouser_Tracks              0.891899
Grouser_Type                0.752813
Hydraulics                  0.200823
Hydraulics_Flow             0.891899
MachineHoursCurrentMeter    0.644089
MachineID                   0.000000
ModelID                     0.000000
Pad_Type                    0.802720
Pattern_Changer             0.752651
ProductGroup                0.000000
ProductGroupDesc            0.000000
ProductSize                 0.525460
Pushblock                   0.937129
Ride_Control                0.629527
Ripper                      0.740388
SalePrice                   0.000000
SalesID                     0.000000
Scarifier                   0.937102
Steering_Controls           0.827064
Stick                       0.802720
Stick_Length                0.752651
Thumb                       0.752476
Tip_Control                 0.937129
Tire_Size                   0.763869
Track_Type                  0.752813
Transmission                0.543210
Travel_Controls             0.800975
Turbocharged                0.802720
Undercarriage_Pad_Width     0.751020
UsageBand                   0.826391
YearMade                    0.000000
auctioneerID                0.050199
datasource                  0.000000
fiBaseModel                 0.000000
fiModelDesc                 0.000000
fiModelDescriptor           0.820707
fiModelSeries               0.858129
fiProductClassDesc          0.000000
fiSecondaryDesc             0.342016
saleDay                     0.000000
saleDayofweek               0.000000
```

In [18]:
```
os.makedirs('tmp', exist_ok=True)
df_raw.to_feather('tmp/raw') #save data frame
```

In [19]:
```
df_raw = pd.read_feather('tmp/raw') #start here
```

In [20]:
```
df, y, nas = proc_df(df_raw, 'SalePrice')
```

In [21]:
```
proc_df
```

Out[21]:
```
<function fastai.structured.proc_df(df, y_fld=None, skip_fld
s=None, ignore_flds=None, do_scale=False, na_dict=None, prep
roc_fn=None, max_n_cat=None, subset=None, mapper=None)>
```

In [22]:
```
m = RandomForestRegressor(n_jobs=-1)
m.fit(df, y)
m.score(df,y)
```

Out[22]:
```
0.9881541589114123
```

In [23]:
```
#return copies of the array that can be modified without affe
def split_vals(a,n): return a[:n].copy(), a[n:].copy()

n_valid = 12000 #same as Kaggle's test set size
n_trn = len(df) - n_valid
raw_train, raw_valid = split_vals(df_raw, n_trn)
X_train, X_valid = split_vals(df, n_trn)
y_train, y_valid = split_vals(y, n_trn)


X_train.shape, y_train.shape, X_valid.shape, y_valid.shape
```

Out[23]:
```
((389125, 66), (389125,), (12000, 66), (12000,))
```

Random Forest

In [24]:
```
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(
          m.score(X_train, y_train), m.score(X_valid, y_valid
    if hasattr(m, 'oob_score_'): res.append(m.oob_score_)
    print(res)
```

In [25]:
```
m = RandomForestRegressor(n_jobs=-1)
%time m.fit(X_train, y_train) # %time shows the time it takes
print_score(m)
```

```
CPU times: user 15min 13s, sys: 2.05 s, total: 15min 15s
Wall time: 1min 59s
[0.07580188012085007, 0.23509093419392463, 0.987991333572618
9, 0.9012994125546683]
```

validation score 0.90 vs training score 0.98 shows that it is over fitting

Speeding things up

Using only a subset of the training data while keeping the same amount of validcation data as before for accuracy

In [28]:
```
df_trn, y_trn, nas = proc_df(df_raw, 'SalePrice', subset = 30
X_train, _ = split_vals(df_trn, 20000) # _ is throw-away vari
y_train, _ = split_vals(y_trn, 20000)
```

In [29]:
```
m = RandomForestRegressor(n_jobs=-1)
%time m.fit(X_train, y_train)
print_score(m)
```

```
CPU times: user 29.4 s, sys: 24.5 ms, total: 29.4 s
Wall time: 3.99 s
[0.09367084705884349, 0.3633582890250942, 0.981038246807553
5, 0.7642139441650002]
```

Bagging

Out-of-bag score: allow us to see whether our model generalizes, even if we only have a small amount of data so to avoid separating some out to create a validation set Use SubSampling to avoid overfitting while increase speed.

In [32]:
```
df_trn, y_trn, nas = proc_df(df_raw, 'SalePrice')
X_train, X_valid = split_vals(df_trn, n_trn)
y_train, y_valid = split_vals(y_trn, n_trn)
```

In [33]:
```
set_rf_samples(20000) #Instead of limiting the total amount o
```

In [34]:
```
m = RandomForestRegressor(n_jobs=-1, oob_score=True)
%time m.fit(X_train, y_train)
print_score(m)
```

```
CPU times: user 15min 23s, sys: 2.54 s, total: 15min 26s
Wall time: 2min 6s
[0.07561297730927669, 0.23425890814807226, 0.988051111622064
7, 0.9019968119588786, 0.9129267589583753]
```

In [35]:
```
m = RandomForestRegressor(n_estimators=40, n_jobs=-1, oob_sco
m.fit(X_train, y_train)
print_score(m)
```

```
[0.07848251024892229, 0.23778830452889807, 0.987126975466430
8, 0.8990214908216662, 0.9083397692005049]
```

Tree building parameters

grow trees less deep

In [37]:
```
reset_rf_samples()
m = RandomForestRegressor(n_estimators=40, min_samples_leaf=3
m.fit(X_train, y_train)
print_score(m)
```

```
[0.11509254937503784, 0.23378793761967684, 0.972315974140590
8, 0.9023904807877438, 0.9085198291640226]
```

using different sets of features(columns) for each split in a tree

In [39]:
```
m = RandomForestRegressor(n_estimators=100, min_samples_leaf=
m.fit(X_train, y_train)
print_score(m)
```

```
[0.11746075403066676, 0.2262361165903815, 0.971164970838260
1, 0.9085946009994461, 0.915335672042507]
```

In [40]:
```
m = RandomForestRegressor(n_estimators=100, min_samples_leaf=
m.fit(X_train, y_train)
print_score(m)
```

```
[0.15578398034595864, 0.2574507538212359, 0.949279862971581
5, 0.8816314549132914, 0.8998328671299352]
```