```
#hide
! [ -e /content ] && pip install -Uqq fastbook kaggle waterfallcharts treeinterpreter dtre
import fastbook
fastbook.setup_book()
```

```
     Mounted at /content/gdrive
```

```
#hide
from fastbook import *
from pandas.api.types import is_string_dtype, is_numeric_dtype, is_categorical_dtype
from fastai.tabular.all import *
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from dtreeviz.trees import *
from IPython.display import Image, display_svg, SVG

pd.options.display.max_rows = 20
pd.options.display.max_columns = 8
```

Tabular Modeling Deep Dive

Categorical Embeddings

Beyond Deep Learning

The Dataset

Kaggle Competitions

```
creds = '{"username":"brenorev","key":"71b8a5307cea76cb6fcda2781b3bffd6"}'
```

```
cred_path = Path('~/.kaggle/kaggle.json').expanduser()
if not cred_path.exists():
    cred_path.parent.mkdir(exist_ok=True)
    cred_path.write_text(creds)
    cred_path.chmod(0o600)
```

```
comp = 'bluebook-for-bulldozers'
path = URLs.path(comp)
path
```

```
#hide
Path.BASE_PATH = path
```

```
from kaggle import api

if not path.exists():
    path.mkdir(parents=true)
    api.competition_download_cli(comp, path=path)
    shutil.unpack_archive(str(path/f'{comp}.zip'), str(path))

path.ls(file_type='text')
```

```
(#0) []
```

Look at the Data

```
df = pd.read_csv(path/'TrainAndValid.csv', low_memory=False)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-19-8e1ab06ebbc2> in <cell line: 1>()
----> 1 df = pd.read_csv(path/'TrainAndValid.csv', low_memory=False)

                              5 frames
/usr/local/lib/python3.9/dist-packages/pandas/io/common.py in
get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors,
storage_options)
    784         if ioargs.encoding and "b" not in ioargs.mode:
    785             # Encoding
--> 786             handle = open(
    787                 handle,
    788                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory:
'/root/.fastai/archive/bluebook-for-bulldozers/TrainAndValid.csv'
```

```
df.columns
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-b666bf274d0a> in <cell line: 1>()
----> 1 df.columns

NameError: name 'df' is not defined
```

SEARCH STACK OVERFLOW

```
df['ProductSize'].unique()
```

```
sizes = 'Large','Large / Medium','Medium','Small','Mini','Compact'
```

```
df['ProductSize'] = df['ProductSize'].astype('category')
df['ProductSize'].cat.set_categories(sizes, ordered=True, inplace=True)
```

```
dep_var = 'SalePrice'
```

```
df[dep_var] = np.log(df[dep_var])
```

Decision Trees

Handling Dates

```
df = add_datepart(df, 'saledate')
```

```
df_test = pd.read_csv(path/'Test.csv', low_memory=False)
df_test = add_datepart(df_test, 'saledate')
```

```
' '.join(o for o in df.columns if o.startswith('sale'))
```

Using TabularPandas and TabularProc

```
procs = [Categorify, FillMissing]
```

```
cond = (df.saleYear<2011) | (df.saleMonth<10)
train_idx = np.where( cond)[0]
valid_idx = np.where(~cond)[0]

splits = (list(train_idx),list(valid_idx))
```

```
cont,cat = cont_cat_split(df, 1, dep_var=dep_var)
```

```
to = TabularPandas(df, procs, cat, cont, y_names=dep_var, splits=splits)
```

```
len(to.train),len(to.valid)
```

```
to.show(3)
```

```
to1 = TabularPandas(df, procs, ['state', 'ProductGroup', 'Drive_System', 'Enclosure'], [],
to1.show(3)
```

```
to.items.head(3)
```

```
to1.items[['state', 'ProductGroup', 'Drive_System', 'Enclosure']].head(3)
```

```
to.classes['ProductSize']
```

```
save_pickle(path/'to.pkl',to)
```

Creating the Decision Tree

```
#hide
to = load_pickle(path/'to.pkl')
```

```
xs,y = to.train.xs,to.train.y
valid_xs,valid_y = to.valid.xs,to.valid.y
```

```
m = DecisionTreeRegressor(max_leaf_nodes=4)
m.fit(xs, y);
```

```
draw_tree(m, xs, size=10, leaves_parallel=True, precision=2)
```

```
samp_idx = np.random.permutation(len(y))[:500]
dtreeviz(m, xs.iloc[samp_idx], y.iloc[samp_idx], xs.columns, dep_var,
        fontname='DejaVu Sans', scale=1.6, label_fontsize=10,
        orientation='LR')
```

```
xs.loc[xs['YearMade']<1900, 'YearMade'] = 1950
valid_xs.loc[valid_xs['YearMade']<1900, 'YearMade'] = 1950
```

```
m = DecisionTreeRegressor(max_leaf_nodes=4).fit(xs, y)

dtreeviz(m, xs.iloc[samp_idx], y.iloc[samp_idx], xs.columns, dep_var,
        fontname='DejaVu Sans', scale=1.6, label_fontsize=10,
        orientation='LR')
```

```
m = DecisionTreeRegressor()
m.fit(xs, y);
```

```
def r_mse(pred,y): return round(math.sqrt(((pred-y)**2).mean()), 6)
def m_rmse(m, xs, y): return r_mse(m.predict(xs), y)
```

```
m_rmse(m, xs, y)
m_rmse(m, valid_xs, valid_y)
```

```
m.get_n_leaves(), len(xs)
```

```
m = DecisionTreeRegressor(min_samples_leaf=25)
m.fit(to.train.xs, to.train.y)
m_rmse(m, xs, y), m_rmse(m, valid_xs, valid_y)
```

```
m.get_n_leaves()
```

Categorical Variables

Random Forests

```
#hide
# pip install —pre -f https://sklearn-nightly.scdn8.secure.raxcdn.com scikit-learn —U
```

Creating a Random Forest

```
def rf(xs, y, n_estimators=40, max_samples=200_000,
       max_features=0.5, min_samples_leaf=5, **kwargs):
    return RandomForestRegressor(n_jobs=-1, n_estimators=n_estimators,
        max_samples=max_samples, max_features=max_features,
        min_samples_leaf=min_samples_leaf, oob_score=True).fit(xs, y)
```

```
m = rf(xs, y);
```

```
m_rmse(m, xs, y), m_rmse(m, valid_xs, valid_y)
```

```
preds = np.stack([t.predict(valid_xs) for t in m.estimators_])
```

```
r_mse(preds.mean(0), valid_y)
```

```
plt.plot([r_mse(preds[:i+1].mean(0), valid_y) for i in range(40)]);
```

Out-of-Bag Error

```
r_mse(m.oob_prediction_, y)
```

Model Interpretation

Tree Variance for Prediction Confidence

```
preds = np.stack([t.predict(valid_xs) for t in m.estimators_])
```

```
preds.shape
```

```
preds_std = preds.std(0)
```

```
preds_std[:5]
```

Feature Importance

```
def rf_feat_importance(m, df):
    return pd.DataFrame({'cols':df.columns, 'imp':m.feature_importances_}
                       ).sort_values('imp', ascending=False)
```

```
fi = rf_feat_importance(m, xs)
fi[:10]
```

```
def plot_fi(fi):
    return fi.plot('cols', 'imp', 'barh', figsize=(12,7), legend=False)

plot_fi(fi[:30]);
```

Removing Low-Importance Variables

```
to_keep = fi[fi.imp>0.005].cols
len(to_keep)
```

```
xs_imp = xs[to_keep]
valid_xs_imp = valid_xs[to_keep]
```

```
m = rf(xs_imp, y)
```

```
m_rmse(m, xs_imp, y), m_rmse(m, valid_xs_imp, valid_y)
```

```
len(xs.columns), len(xs_imp.columns)
```

```
plot_fi(rf_feat_importance(m, xs_imp));
```

Removing Redundant Features

```
cluster_columns(xs_imp)
```

```
def get_oob(df):
    m = RandomForestRegressor(n_estimators=40, min_samples_leaf=15,
        max_samples=50000, max_features=0.5, n_jobs=-1, oob_score=True)
    m.fit(df, y)
    return m.oob_score_
```

```
get_oob(xs_imp)
```

```
{c:get_oob(xs_imp.drop(c, axis=1)) for c in (
    'saleYear', 'saleElapsed', 'ProductGroupDesc','ProductGroup',
    'fiModelDesc', 'fiBaseModel',
    'Hydraulics_Flow','Grouser_Tracks', 'Coupler_System')}
```

```
to_drop = ['saleYear', 'ProductGroupDesc', 'fiBaseModel', 'Grouser_Tracks']
get_oob(xs_imp.drop(to_drop, axis=1))
```

```
xs_final = xs_imp.drop(to_drop, axis=1)
valid_xs_final = valid_xs_imp.drop(to_drop, axis=1)
```

```
save_pickle(path/'xs_final.pkl', xs_final)
save_pickle(path/'valid_xs_final.pkl', valid_xs_final)
```

```
xs_final = load_pickle(path/'xs_final.pkl')
valid_xs_final = load_pickle(path/'valid_xs_final.pkl')
```

```
m = rf(xs_final, y)
m_rmse(m, xs_final, y), m_rmse(m, valid_xs_final, valid_y)
```

Partial Dependence

```
p = valid_xs_final['ProductSize'].value_counts(sort=False).plot.barh()
c = to.classes['ProductSize']
plt.yticks(range(len(c)), c);
```

```
ax = valid_xs_final['YearMade'].hist()
```

```
from sklearn.inspection import PartialDependenceDisplay

fig,ax = plt.subplots(figsize=(12, 4))
```

Data Leakage

Tree Interpreter

```
#hide
import warnings
warnings.simplefilter('ignore', FutureWarning)

from treeinterpreter import treeinterpreter
from waterfall_chart import plot as waterfall
```

```
row = valid_xs_final.iloc[:5]
```

```
prediction,bias,contributions = treeinterpreter.predict(m, row.values)
```

```
prediction[0], bias[0], contributions[0].sum()
```

```
waterfall(valid_xs_final.columns, contributions[0], threshold=0.08,
          rotation_value=45,formatting='{:,.3f}');
```

Extrapolation and Neural Networks

The Extrapolation Problem

```
#hide
np.random.seed(42)
```

```
x_lin = torch.linspace(0,20, steps=40)
y_lin = x_lin + torch.randn_like(x_lin)
plt.scatter(x_lin, y_lin);
```

```
xs_lin = x_lin.unsqueeze(1)
x_lin.shape,xs_lin.shape
```

```
x_lin[:,None].shape
```

```
m_lin = RandomForestRegressor().fit(xs_lin[:30],y_lin[:30])
```

```
plt.scatter(x_lin, y_lin, 20)
plt.scatter(x_lin, m_lin.predict(xs_lin), color='red', alpha=0.5);
```

Finding Out-of-Domain Data

```
df_dom = pd.concat([xs_final, valid_xs_final])
is_valid = np.array([0]*len(xs_final) + [1]*len(valid_xs_final))

m = rf(df_dom, is_valid)
rf_feat_importance(m, df_dom)[:6]
```

```
m = rf(xs_final, y)
print('orig', m_rmse(m, valid_xs_final, valid_y))

for c in ('SalesID','saleElapsed','MachineID'):
    m = rf(xs_final.drop(c,axis=1), y)
    print(c, m_rmse(m, valid_xs_final.drop(c,axis=1), valid_y))
```

```
time_vars = ['SalesID','MachineID']
xs_final_time = xs_final.drop(time_vars, axis=1)
valid_xs_time = valid_xs_final.drop(time_vars, axis=1)

m = rf(xs_final_time, y)
m_rmse(m, valid_xs_time, valid_y)
```

```
xs['saleYear'].hist();
```

```
filt = xs['saleYear']>2004
xs_filt = xs_final_time[filt]
y_filt = y[filt]
```

```
m = rf(xs_filt, y_filt)
m_rmse(m, xs_filt, y_filt), m_rmse(m, valid_xs_time, valid_y)
```

Using a Neural Network

```
df_nn = pd.read_csv(path/'TrainAndValid.csv', low_memory=False)
df_nn['ProductSize'] = df_nn['ProductSize'].astype('category')
df_nn['ProductSize'].cat.set_categories(sizes, ordered=True, inplace=True)
df_nn[dep_var] = np.log(df_nn[dep_var])
df_nn = add_datepart(df_nn, 'saledate')
```

```
df_nn_final = df_nn[list(xs_final_time.columns) + [dep_var]]
```

```
cont_nn,cat_nn = cont_cat_split(df_nn_final, max_card=9000, dep_var=dep_var)
```

```
cont_nn
```

```
df_nn_final[cat_nn].nunique()
```

```
xs_filt2 = xs_filt.drop('fiModelDescriptor', axis=1)
valid_xs_time2 = valid_xs_time.drop('fiModelDescriptor', axis=1)
m2 = rf(xs_filt2, y_filt)
m_rmse(m2, xs_filt2, y_filt), m_rmse(m2, valid_xs_time2, valid_y)
```

```
cat_nn.remove('fiModelDescriptor')
```

```
procs_nn = [Categorify, FillMissing, Normalize]
to_nn = TabularPandas(df_nn_final, procs_nn, cat_nn, cont_nn,
                      splits=splits, y_names=dep_var)
```

```
dls = to_nn.dataloaders(1024)
```

```
y = to_nn.train.y
y.min(),y.max()
```

```
learn = tabular_learner(dls, y_range=(8,12), layers=[500,250],
                        n_out=1, loss_func=F.mse_loss)
```

```
learn.lr_find()
```

```
learn.fit_one_cycle(5, 1e-2)
```

```
preds,targs = learn.get_preds()
r_mse(preds,targs)
```

```
learn.save('nn')
```

Sidebar: fastai's Tabular Classes

End sidebar

Ensembling

```
rf_preds = m.predict(valid_xs_time)
ens_preds = (to_np(preds.squeeze()) + rf_preds) /2


r_mse(ens_preds,valid_y)
```

Boosting

Combining Embeddings with Other Methods

Conclusion: Our Advice for Tabular Modeling

Questionnaire

1. What is a continuous variable?
2. What is a categorical variable?
3. Provide two of the words that are used for the possible values of a categorical variable.
4. What is a "dense layer"?
5. How do entity embeddings reduce memory usage and speed up neural networks?
6. What kinds of datasets are entity embeddings especially useful for?
7. What are the two main families of machine learning algorithms?
8. Why do some categorical columns need a special ordering in their classes? How do you do this in Pandas?
9. Summarize what a decision tree algorithm does.
10. Why is a date different from a regular categorical or continuous variable, and how can you preprocess it to allow it to be used in a model?
11. Should you pick a random validation set in the bulldozer competition? If no, what kind of validation set should you pick?
12. What is pickle and what is it useful for?
13. How are `mse`, `samples`, and `values` calculated in the decision tree drawn in this chapter?
14. How do we deal with outliers, before building a decision tree?
15. How do we handle categorical variables in a decision tree?
16. What is bagging?
17. What is the difference between `max_samples` and `max_features` when creating a random forest?
18. If you increase `n_estimators` to a very high value, can that lead to overfitting? Why or why not?

19. In the section "Creating a Random Forest", just after <>, why did `preds.mean(0)` give the same result as our random forest?
20. What is "out-of-bag-error"?
21. Make a list of reasons why a model's validation set error might be worse than the OOB error. How could you test your hypotheses?
22. Explain why random forests are well suited to answering each of the following question:
    - How confident are we in our predictions using a particular row of data?
    - For predicting with a particular row of data, what were the most important factors, and how did they influence that prediction?
    - Which columns are the strongest predictors?
    - How do predictions vary as we vary these columns?
23. What's the purpose of removing unimportant variables?
24. What's a good type of plot for showing tree interpreter results?
25. What is the "extrapolation problem"?
26. How can you tell if your test or validation set is distributed in a different way than your training set?
27. Why do we ensure `saleElapsed` is a continuous variable, even although it has less than 9,000 distinct values?
28. What is "boosting"?
29. How could we use embeddings with a random forest? Would we expect this to help?
30. Why might we not always use a neural net for tabular modeling?

Further Research

1. Pick a competition on Kaggle with tabular data (current or past) and try to adapt the techniques seen in this chapter to get the best possible results. Compare your results to the private leaderboard.
2. Implement the decision tree algorithm in this chapter from scratch yourself, and try it on the dataset you used in the first exercise.
3. Use the embeddings from the neural net in this chapter in a random forest, and see if you can improve on the random forest results we saw.
4. Explain what each line of the source of `TabularModel` does (with the exception of the `BatchNorm1d` and `Dropout` layers).