The Price is Right! - Model 04 - NN

Rotem Nizhar & Batel Mankovsky

Imports & Settings

In [ ]:
```
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
from functools import reduce

from contextlib import contextmanager
from functools import partial
from operator import itemgetter
from multiprocessing.pool import ThreadPool
import time
from typing import List, Dict

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras as ks
from sklearn.feature_extraction import DictVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer as Tfidf
from sklearn.pipeline import make_pipeline, make_union, Pipeline
from sklearn.preprocessing import FunctionTransformer, StandardScaler
from sklearn.metrics import mean_squared_log_error
from sklearn.model_selection import KFold

%matplotlib inline
```

Prepare Data

In [38]:
```
# Preprocessing in this case is much simpler: we append many categorical features to the "text" variable, and add
# some additional features that we already know are interesting as boolean features.

def preprocess(df: pd.DataFrame) -> pd.DataFrame:
  df["title"] = df["brand"].fillna("") + " " + df["title"].fillna("")
  df["seller_notes"] = df["seller_notes"].str.strip('“').str.strip('”').str.strip('"')
  df["text"] = reduce(lambda x, y: x + " " + y, (df[x].fillna("") for x in ["seller_notes", "title", "category", "style", "material", "upper_material", "occasion", "color", "colour", "condition"]))
  df["longtime_member"] = df["longtime_member"].fillna(0)
  df["free_shipping"] = df["free_shipping"].fillna(0)
  df["same_day_shipping"] = df["same_day_shipping"].fillna(0)
  df["fast_safe_shipping"] = df["fast_safe_shipping"].fillna(0)
  df["returns"] = df["returns"].fillna(0)
  df["feedback"] = df["feedback"].fillna(0)

  df = df[["title", "text", "longtime_member", "free_shipping", "same_day_shipping", "fast_safe_shipping", "returns", "feedback"]]
  return df
```

In [39]:
```
@contextmanager
def timer(name):
    t0 = time.time()
    yield
    print(f'[{name}] done in {time.time() - t0:.0f} s')
```

In [40]:
```
def on_field(f: str, *vec) -> Pipeline:
    return make_pipeline(FunctionTransformer(itemgetter(f), validate=False), *vec)
```

In [41]:
```
def to_records(df: pd.DataFrame) -> List[Dict]:
    return df.to_dict(orient='records')
```

In [95]:
```
# Build model
def create_model(xs):
    model_in = ks.Input(shape=(xs.shape[1],), dtype='float32', sparse=True)
    out = ks.layers.Dense(192, activation='relu')(model_in)
    out = ks.layers.Dense(64, activation='relu')(out)
    out = ks.layers.Dense(64, activation='relu')(out)
    out = ks.layers.Dense(1)(out)
    model = ks.Model(model_in, out)
    model.compile(loss='mean_squared_error', optimizer=ks.optimizers.Adam(learning_rate=3e-3))
    return model
```

In [96]:
```
# Fit model on data - run for 10 epochs with increasing batch size
def fit_predict(model, xs, y_train) -> np.ndarray:
    X_train, X_test = xs
    for i in range(10):
      with timer(f'epoch {i + 1}'):
        model.fit(x=X_train, y=y_train, batch_size=2**(11 + i), epochs=1, verbose=0)
    return model.predict(X_test)[:, 0]
```

In [98]:
```
# vectorizer - for preprocessing of X data
vectorizer = make_union(
    on_field('title', Tfidf(max_features=100000, token_pattern='\w+')),
    on_field('text', Tfidf(max_features=100000, token_pattern='\w+', ngram_range=(1, 2))),
    on_field(['longtime_member', 'free_shipping', 'same_day_shipping', 'returns'],
             FunctionTransformer(to_records, validate=False), DictVectorizer()),
    n_jobs=4)

# Scaler - standard scaler for scaling the y values
y_scaler = StandardScaler()

train = pd.read_csv('data/ebay_women_shoes_train.csv')
train = train[train['price'] > 0].reset_index(drop=True)

cv = KFold(n_splits=5, shuffle=True, random_state=42)
train_ids, valid_ids = next(cv.split(train))
train, valid = train.iloc[train_ids], train.iloc[valid_ids]

# train, valid = train, train.iloc[valid_ids]  # <-- this is the line we ran we we trained model on WHOLE data before submission

# Fit & transform
y_train = y_scaler.fit_transform(np.log(train['price'].values.reshape(-1, 1)))
X_train = vectorizer.fit_transform(preprocess(train)).astype(np.float32)

print(f'X_train: {X_train.shape} of {X_train.dtype}')
del train
X_valid = vectorizer.transform(preprocess(valid)).astype(np.float32)
```

```
X_train: (11200, 78065) of float32
```

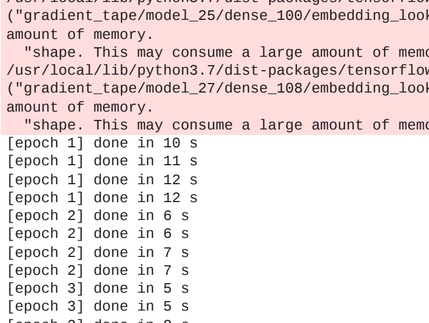
In [99]:
```
ensemble_size = 4
models = [create_model(X_train) for _ in range(ensemble_size)]

# Xb_train is a trick we saw they used to achieve better results as it allowed them to give the model "more data".
Xb_train, Xb_valid = [x.astype(bool).astype(np.float32) for x in [X_train, X_valid]]
xs = [(Xb_train, Xb_valid), (X_train, X_valid)] * (ensemble_size//2)
with ThreadPool(processes=ensemble_size) as pool:
  y_pred = np.mean(pool.starmap(partial(fit_predict, y_train=y_train), zip(models, xs)), axis=0)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/indexed_slices.py:446: UserWarning: Converting sparse IndexedSlices(IndexedSlices(indices=Tensor("gradient_tape/model_24/dense_96/embedding_lookup_sparse/Reshape_1:0", shape=(None,), dtype=int32), values=Tensor
("gradient_tape/model_24/dense_96/embedding_lookup_sparse/Reshape:0", shape=(None, 192), dtype=float32), dense_shape=Tensor("gradient_tape/model_24/dense_96/embedding_lookup_sparse/Cast:0", shape=(2,), dtype=int32))) to a dense Tensor of unknown shape. This may consume a large a
mount of memory.
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/indexed_slices.py:446: UserWarning: Converting sparse IndexedSlices(IndexedSlices(indices=Tensor("gradient_tape/model_26/dense_104/embedding_lookup_sparse/Reshape_1:0", shape=(None,), dtype=int32), values=Tensor
("gradient_tape/model_26/dense_104/embedding_lookup_sparse/Reshape:0", shape=(None, 192), dtype=float32), dense_shape=Tensor("gradient_tape/model_26/dense_104/embedding_lookup_sparse/Cast:0", shape=(2,), dtype=int32))) to a dense Tensor of unknown shape. This may consume a large
amount of memory.
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/indexed_slices.py:446: UserWarning: Converting sparse IndexedSlices(IndexedSlices(indices=Tensor("gradient_tape/model_25/dense_100/embedding_lookup_sparse/Reshape_1:0", shape=(None,), dtype=int32), values=Tensor
("gradient_tape/model_25/dense_100/embedding_lookup_sparse/Reshape:0", shape=(None, 192), dtype=float32), dense_shape=Tensor("gradient_tape/model_25/dense_100/embedding_lookup_sparse/Cast:0", shape=(2,), dtype=int32))) to a dense Tensor of unknown shape. This may consume a large
amount of memory.
  "shape. This may consume a large amount of memory." % value)
/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/indexed_slices.py:446: UserWarning: Converting sparse IndexedSlices(IndexedSlices(indices=Tensor("gradient_tape/model_27/dense_108/embedding_lookup_sparse/Reshape_1:0", shape=(None,), dtype=int32), values=Tensor
("gradient_tape/model_27/dense_108/embedding_lookup_sparse/Reshape:0", shape=(None, 192), dtype=float32), dense_shape=Tensor("gradient_tape/model_27/dense_108/embedding_lookup_sparse/Cast:0", shape=(2,), dtype=int32))) to a dense Tensor of unknown shape. This may consume a large
amount of memory.
  "shape. This may consume a large amount of memory." % value)
[epoch 1] done in 10 s
[epoch 1] done in 11 s
[epoch 1] done in 12 s
[epoch 1] done in 12 s
[epoch 2] done in 6 s
[epoch 2] done in 6 s
[epoch 2] done in 7 s
[epoch 3] done in 5 s
[epoch 3] done in 5 s
[epoch 3] done in 8 s
[epoch 3] done in 5 s
[epoch 4] done in 4 s
[epoch 4] done in 4 s
[epoch 4] done in 4 s
[epoch 4] done in 5 s
[epoch 5] done in 4 s
[epoch 5] done in 4 s
[epoch 5] done in 4 s
[epoch 5] done in 4 s
[epoch 6] done in 4 s
[epoch 6] done in 3 s
[epoch 6] done in 4 s
[epoch 6] done in 5 s
[epoch 7] done in 4 s
[epoch 7] done in 3 s
[epoch 7] done in 5 s
[epoch 8] done in 3 s
[epoch 7] done in 5 s
[epoch 8] done in 4 s
[epoch 9] done in 3 s
[epoch 8] done in 3 s
[epoch 9] done in 4 s
[epoch 9] done in 3 s
[epoch 8] done in 8 s
[epoch 10] done in 4 s
[epoch 10] done in 2 s
[epoch 10] done in 3 s
[epoch 9] done in 4 s
[epoch 10] done in 1 s
```

In [100…
```
y_pred2 = y_scaler.inverse_transform(y_pred.reshape(-1, 1))[:, 0]
print('Valid RMSE: {:.4f}'.format(np.sqrt(mean_squared_error(np.log(valid['price']), y_pred2))))
```

```
Valid RMSE: 0.4687
```

In [101…
```
_ = plt.hist(y_pred2, bins="auto")
plt.show()
```



In [102…
```
np.mean(y_pred2)
```

Out[102]:
```
3.5279834
```

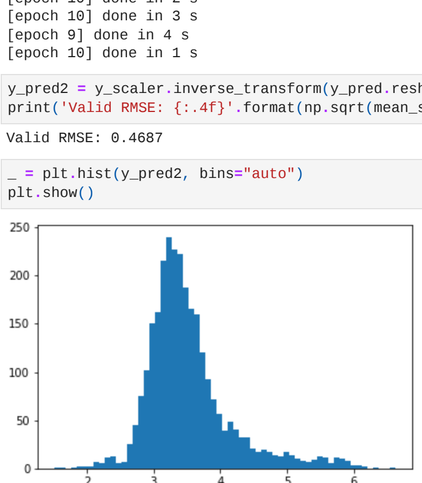
In [103…
```
test = pd.read_csv('data/ebay_women_shoes_test.csv')
test_ids = test.id
test = test.reset_index(drop=True)

# Create X test data
X_test = vectorizer.transform(preprocess(test)).astype(np.float32)
Xb_test = X_test.astype(bool).astype(np.float32)
xs = [Xb_test, X_test] * (ensemble_size//2)
```

In [104…
```
result = np.mean([models[i].predict(xs[i])[:, 0] for i in range(ensemble_size)], axis=0)
result = y_scaler.inverse_transform(result.reshape(-1, 1))[:, 0]
```

In [105…
```
print(np.mean(result))
# print('Valid RMSE: {:.4f}'.format(np.sqrt(mean_squared_error(np.log(test['price']), result))))
```

```
3.5235581
```

In [106…
```
_ = plt.hist(result, bins="auto")
plt.show()
```



In [88]:
```
df = pd.DataFrame({"id": test_ids, "price_pred": result})
df.to_csv("model_05_v2.csv", index=False)
```