# Mercari 1st place solution

Konstantin Lopuhin & Pawel Jankiewicz

2018-03-31

# About us

**Konstantin Lopuhin**

Software engineer at Scrapinghub

Москва, Россия

Joined 6 years ago · last seen in the past day

Competitions Master

| Current Rank | Highest Rank |
|---|---|
| 32 | 30 |
| of 80,984 | |

| Gold | Silver | Bronze |
|---|---|---|
| 4 | 1 | 0 |

| Competition | Place |
|---|---|
| Mercari Price Suggestion C... · a month ago · Top 1% | 1st of 2384 |
| NOAA Fisheries Steller Sea... · 9 months ago · Top 1% | 2nd of 385 |
| Dstl Satellite Imagery Feat... · a year ago · Top 2% | 5th of 419 |

**Paweł Jankiewicz**

Warszawa, mazowieckie, Polska

Joined 6 years ago · last seen in the past day

in http://logicai.io

Competitions Grandmaster

| Current Rank | Highest Rank |
|---|---|
| 76 | 40 |
| of 80,984 | |

| Gold | Silver | Bronze |
|---|---|---|
| 9 | 2 | 3 |

| Competition | Place |
|---|---|
| Mercari Price Suggestion C... · a month ago · Top 1% | 1st of 2384 |
| Will I Stay or Will I Go? · 5 years ago · Top 9% | 1st of 12 |
| GE Flight Quest · 5 years ago · Top 2% | 2nd of 173 |

# We won!

At the time of merging we were 1st (Konstantin) and 2nd (Pawel).

| # | △pub | Team Name | Kernel | Team Members | Score |
|---|---|---|---|---|---|
| 1 | — | Paweł and Konstantin | | | 0.37758 |
| 2 | — | Mercaring (Nima & Chahhou) | | | 0.38875 |
| 3 | ▲1 | bird | | | 0.39134 |
| 4 | ▲1 | Chenglong Chen | | | 0.39299 |
| 5 | ▲2 | anttip | | | 0.39603 |
| 6 | ▲5 | Fair trade | | | 0.39713 |
| 7 | ▲3 | Basil | | | 0.39720 |
| 8 | — | RDizzl3 and Sergei | | | 0.39734 |
| 9 | ▲3 | LeeYun | </> ensemble model | | 0.39752 |
| 10 | ▲3 | stooging the stooges | | +5 | 0.39766 |

# Mercari competition

https://www.kaggle.com/c/mercari-price-suggestion-challenge/

# Evaluation RMSLE

RMSLE

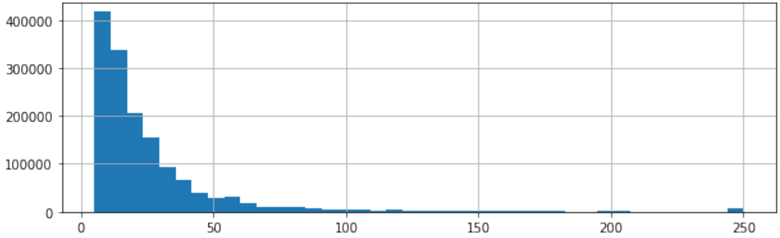
$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

Better to convert to RMSE - optimize directly

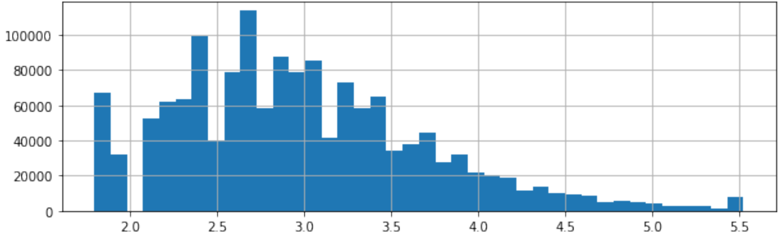
```
y_log = log(y + 1)
model.fit(X, y_log)
prediction_log = model.predict(X_test)
prediction = exp(prediction_log) - 1
```

# Why Logarithm?

Price distribution




Log(price) distribution



# Code competition

Only 60 minutes to train and predict!

System specification: 16 GB ram, 1 GB disk, $\sim$ 4 cores

Advantages

- No huge ensembles
- Team collaboration - common code base
- Small models, fast iteration

Disadvantages

- Pure optimization
- Unstable platform
- 2 stage competition (5x bigger test data)

# Build system

8f5b46e — Draft saved — Python — Commit & Run

Code | Data | Settings | Versions

```python
import gzip
import base64
import os
import sys
sys.path.append('/kaggle/working')

# this is base64 encoded source code
file_data = {'__init__.py': 'H4sIAMxEgVoC/wMAAAAAAAAAAAA=', 'config.py': 'H4sIAMxEgVoC/4VUXW+bMBR996+wOiFAahht162rlEk0eA1aQqOQTNteLBRuqFXAFXbabtP++y4QMmjS

os.system('mkdir mercari')


for fn, encoded in file_data.items():
    print(fn)
    with open('mercari/' + fn, 'wb') as out:
        out.write(gzip.decompress(base64.b64decode(encoded)))


with open('setup.py','wt') as out:
```

Console | Environment Variables — CPU 0% RAM 215MB/17.2GB Disk 55.4MB/1GB

```
[Initializing]
[Starting]
[Running]
Your kernel is now running in the cloud. Here are some things you can do with it:
 * Use the Play button or [SHIFT]+[ENTER] to execute the current line of your script (or whatever's highlighted).
 * Enter some code at the bottom of this Console tab and press [ENTER].
```

# About the data

1.5 million observations in the training data

| Column name | Type | # unique values |
|---|---|---|
| name | text | - |
| item_description | text | - |
| item_condition_id | categorical - ordinal | 5 |
| category_name | text/categorical | 1288 |
| brand_name | text/categorical | 4810 |
| shipping | boolean | - |

# Example Item


```
{
 "name": "NYX GLITTER GLUE+GLITTER BUNDLE",
 "item_description": "FREE FAST SHIPPING EXTRA FREE☆☆ BEAUTY GIFTS:)
¡¡♡♡☆☆☆ Extra free skincare gift:))¡¡♡♡☆⋄⋄⋄ NYX GLITTER PRIMER glue..
brand new sealed in box.. full size 4 NYX eye. glitters!! brand new..all in 5
grams jars each.. variety of colors to match just about any makeup look you
decide to create Wet n wild small concealer brush.. brand new and sealed..
perfect sized brush to apply glitter primer plus glitters onto your eyelid
without being messy.. works a lot like the real techniques detailer brush",
 "item_condition_id": 1,
 "category_name": "Beauty/Makeup/Eyes",
 "brand_name": "NYX",
 "shipping": 1,
 "price": 28.0
}
```

# Data preprocessing: Declarative vs Imperative

**Imperative**

```
D    vect = CountVectorizer()
A    vect.fit(X)
A    mat = vect.transform(X)
D    rf = RandomForestRegressor()
A    rf.fit(mat, y)
```

```
D = declaration, A = action
```

**Declarative**

```
D    model = make_pipeline(
D      CountVectorizer(),
D      RandomForestRegressor()
D    )
A    model.fit(X, y)
```

# "It's pipelines all the way down"

```
def prepare_vectorizer_1_tf(n_jobs=4):
    tokenizer = FastTokenizer()
    vectorizer = make_pipeline(
        FillEmpty(),
        PreprocessDataPJ(n_jobs=n_jobs),
        make_union_mp(

            make_pipeline(
                PandasSelector(columns=['name', 'item_description']),
                ConcatTexts(columns=['name', 'item_description'],
                            use_separators=True),
                PandasSelector(columns=['text_concat']),
                CountVectorizer(ngram_range=(1, 1), binary=True, min_df=5, tokenizer=tokenizer, dtype=np.float32)
            ),

            make_pipeline(PandasSelector(columns=['category_name_clean']),
                          CountVectorizer(tokenizer=tokenizer,
                                          binary=True,
                                          min_df=5,
                                          dtype=np.float32)),

            make_pipeline(PandasSelector(columns=['shipping', 'item_condition_id', 'brand_name_clean',
                                                  'cat_1', 'cat_2', 'cat_3', 'no_cat']),
                          PandasToRecords(),
                          DictVectorizer(dtype=np.float32)),

            n_jobs=n_jobs
        ),
        SparseMatrixOptimize(),
        SanitizeSparseMatrix(),
        ReportShape()
    )
    return vectorizer
```

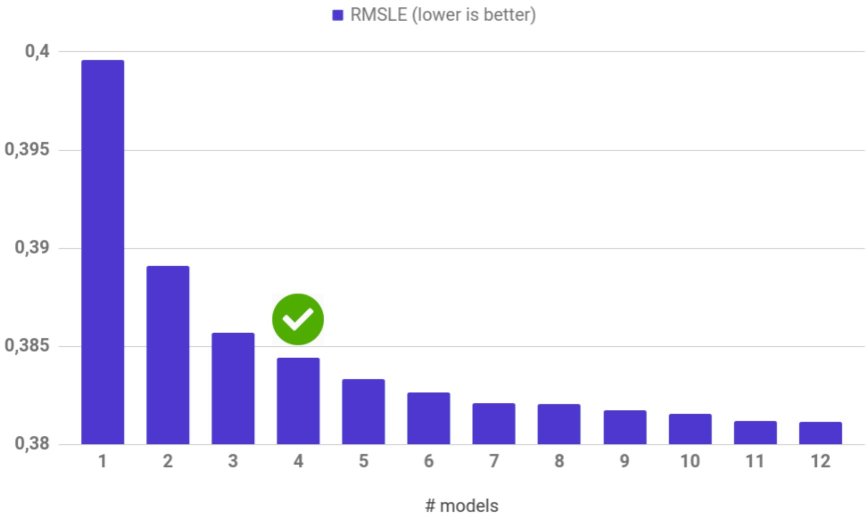
# Preprocessing

- Text preprocessing - stemming
- Bag of words - 1,2-grams (with/without Tf-Idf)
- One hot encoding for categorical columns

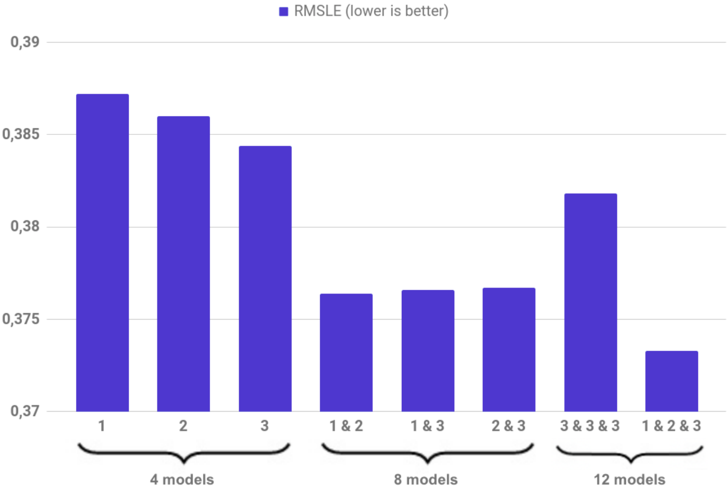
- Bag of character 3-grams

- Joining name, brand name and description into a single field
- NumericalVectorizer - vectorizing words using preceding numbers

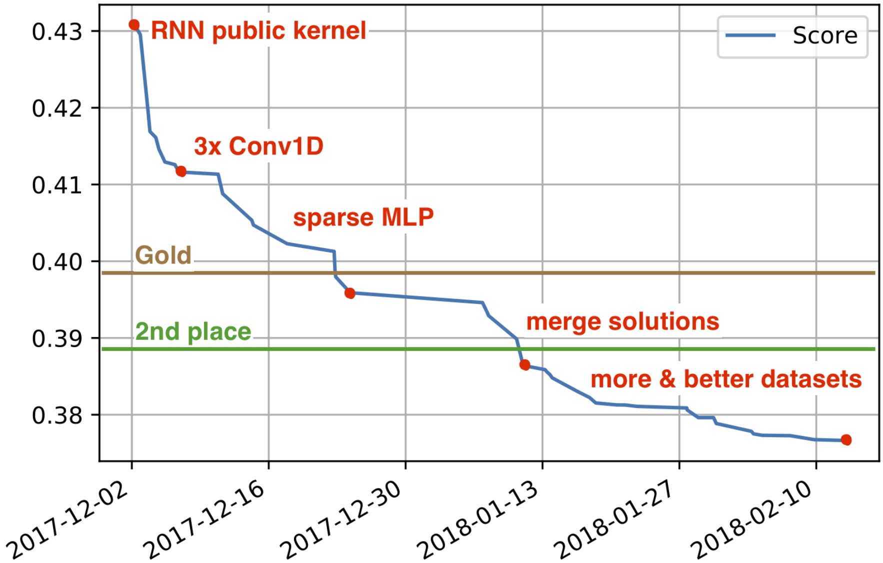
# Why ensemble?



# Why 3 datasets?



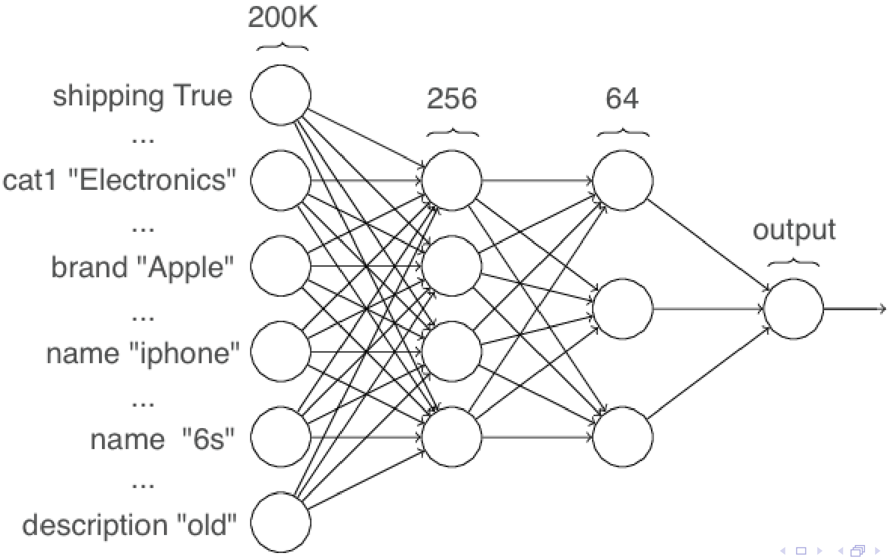
# Our progress



# To Deep Learn or not to Deep Learn?

| | Learning | Deep Learning |
|---|---|---|
| Architecture | **MLP** | LSTM, CNN |
| Activation | Tanh | **ReLU** |
| Optimization | SGD | **ADAM** |
| Categorical features | **One-hot encoding** | Embeddings |

# Workhorse model: sparse MLP (feedforward neural network)

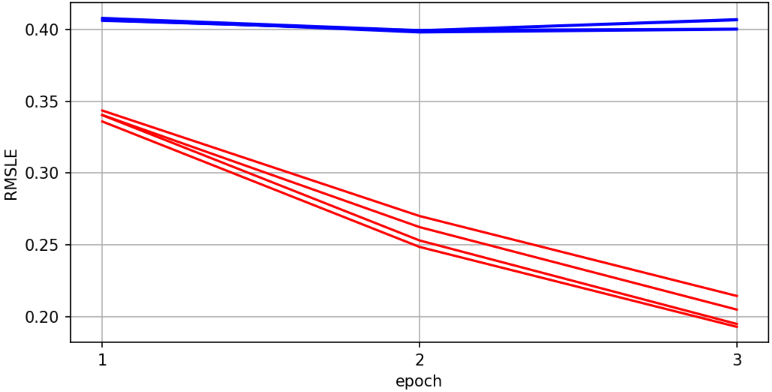


# Why MLP?

- Fast to train: can afford hidden size 256 instead of 32–64 for RNN or Conv1D.
- Captures interactions between text and categorical features.
- Huge variance gives a strong ensemble with a single model type.

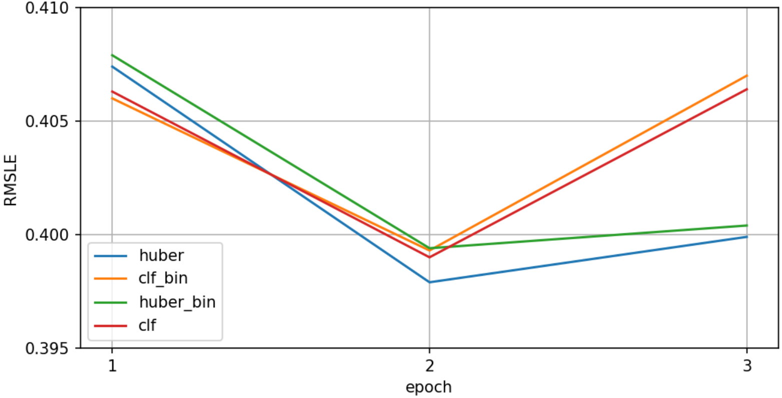
# Training

Adam, double batch size after each epoch, overfit



# Training

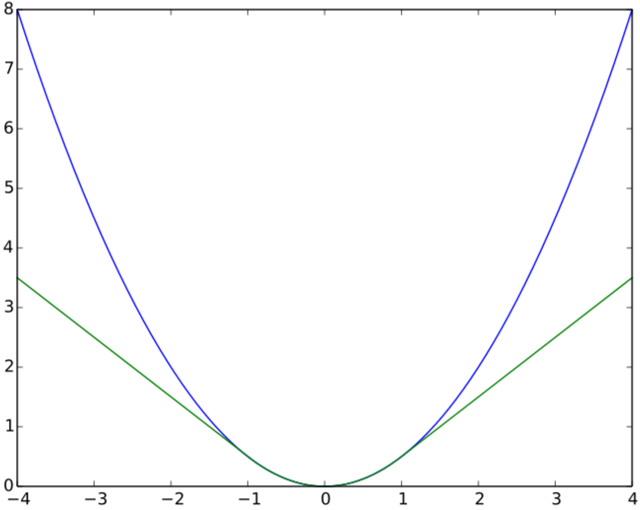
Adam, double batch size after each epoch, overfit, profit!



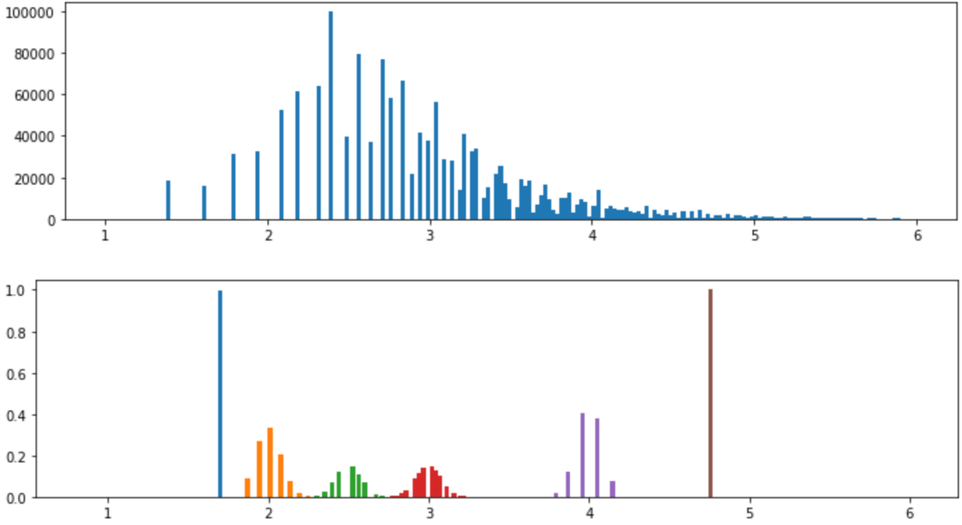
# Tricks

- Huber loss
- Regression via. classification
- Cheap feature binarization

# Huber Loss



# Regression via Classification



# Cheap feature binarization

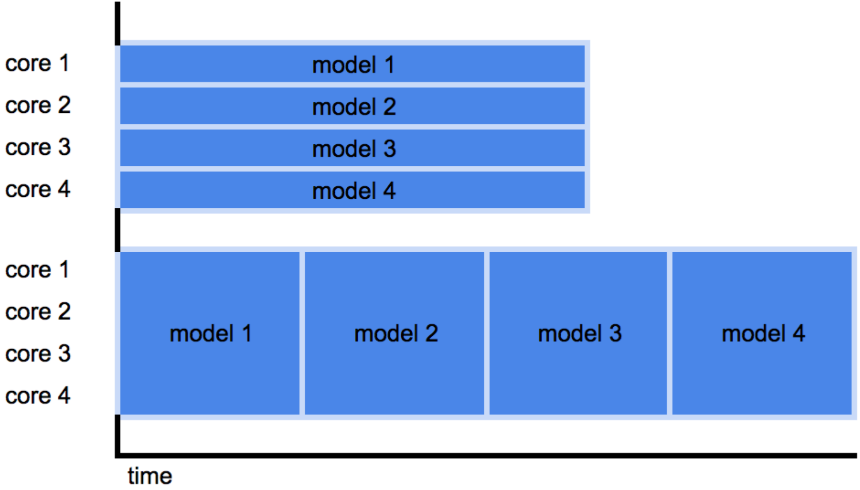
TF-IDF features $\Rightarrow$ Binary features

```
def feed_dict(self, X, binary_X=False):
    coo = X.tocoo()
    return {
        self.indices: np.stack([coo.row, coo.col]).T,
        self.values: coo.data if not binary_X else np.ones_like(coo.data),
        self.shape: np.array(X.shape),
    }
```

# Sparse MLP Implementation

- TensorFlow: `tf.sparse_tensor_dense_matmul`
- MXNet: `RowSparseNDArray`, sparse updates!
- Keras: `keras.Input(sparse=True)`
- Any framework: via embedding

# Optimization: One Model per Core



# Optimization: Memory

- TensorFlow: threading, `use_per_session_threads`
- MXNet: multiprocessing, memory efficient data loader

# Ensembling via Lasso

5% local validation, 1% on Kaggle. Very good LB correlation.

```
merge_predictions =
-0.0203
+0.0604 * data1_huber
+0.1051 * data1_huber
+0.0911 * data1_clf
+0.0760 * data1_clf
+0.0851 * data2_huber_bin
+0.0981 * data2_huber
+0.0819 * data2_clf_bin
+0.0717 * data2_clf
+0.0958 * data3_huber_bin
+0.1226 * data3_huber
+0.0578 * data3_clf_bin
+0.0642 * data3_clf
⇒ RMSLE 0.3733
```

# Didn't Work

- Grid Search
- Skip Connections
- Mixture of Experts
- Factorization Machines
- Fitting residuals

# Code Golf: 0.3875 CV in 75 LOC, 1900 s

- Sparse MLP in Keras
- Train 4 models on 4 cores
- Custom preprocessing

```python
import os; os.environ['OMP_NUM_THREADS'] = '1'
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 keras as ks
import pandas as pd
import numpy as np
import tensorflow as tf
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

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

big CPU win!

boring stuff

```python
def preprocess(df: pd.DataFrame) -> pd.DataFrame:
    df['name'] = df['name'].fillna('') + ' ' + df['brand_name'].fillna('')
    df['text'] = (df['item_description'].fillna('') + ' ' + df['name'] + ' ' +
df['category_name'].fillna(''))
    return df[['name', 'text', 'shipping', 'item_condition_id']]

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

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

def fit_predict(xs, y_train) -> np.ndarray:
    X_train, X_test = xs
    config = tf.ConfigProto(
        intra_op_parallelism_threads=1, use_per_session_threads=1, inter_op_parallelism_threads=1)
    with tf.Session(graph=tf.Graph(), config=config) as sess, timer('fit_predict'):
        ks.backend.set_session(sess)
        model_in = ks.Input(shape=(X_train.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(lr=3e-3))
        for i in range(3):
            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]
```

feature engineering

TF trick

the MODEL

```python
def main():
    vectorizer = make_union(
        on_field('name', Tfidf(max_features=100000, token_pattern='\w+')),
        on_field('text', Tfidf(max_features=100000, token_pattern='\w+', ngram_range=(1, 2))),
        on_field(['shipping', 'item_condition_id'],
                 FunctionTransformer(to_records, validate=False), DictVectorizer()),
        n_jobs=4)
    y_scaler = StandardScaler()
    with timer('process train'):
        train = pd.read_table('../input/train.tsv')
        train = train[train['price'] > 0].reset_index(drop=True)
        cv = KFold(n_splits=20, shuffle=True, random_state=42)
        train_ids, valid_ids = next(cv.split(train))
        train, valid = train.iloc[train_ids], train.iloc[valid_ids]
        y_train = y_scaler.fit_transform(np.log1p(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
    with timer('process valid'):
        X_valid = vectorizer.transform(preprocess(valid)).astype(np.float32)
    with ThreadPool(processes=4) as pool:
        Xb_train, Xb_valid = [x.astype(np.bool).astype(np.float32) for x in [X_train, X_valid]]
        xs = [[Xb_train, Xb_valid], [X_train, X_valid]] * 2
        y_pred = np.mean(pool.map(partial(fit_predict, y_train=y_train), xs), axis=0)
    y_pred = np.expm1(y_scaler.inverse_transform(y_pred.reshape(-1, 1))[:, 0])
    print('Valid RMSLE: {:.4f}'.format(np.sqrt(mean_squared_log_error(valid['price'], y_pred))))
```

feature engineering

scale target

extra dataset for free

4x!

# Feature Engineering

```python
def preprocess(df: pd.DataFrame) -> pd.DataFrame:
    df['name'] = df['name'].fillna('') + ' ' + df['brand_name'].fillna('')
    df['text'] = (df['item_description'].fillna('') + ' ' + df['name'] + ' ' +
df['category_name'].fillna(''))
    return df[['name', 'text', 'shipping', 'item_condition_id']]

    vectorizer = make_union(
        on_field('name', Tfidf(max_features=100000, token_pattern='\w+')),
        on_field('text', Tfidf(max_features=100000, token_pattern='\w+', ngram_range=(1, 2))),
        on_field(['shipping', 'item_condition_id'],
                 FunctionTransformer(to_records, validate=False), DictVectorizer()),
        n_jobs=4)
    y_scaler = StandardScaler()
```

join fields

more features!

# The Model

```python
    X_train, X_test = xs
    config = tf.ConfigProto(
        intra_op_parallelism_threads=1, use_per_session_threads=1, inter_op_parallelism_threads=1)
    with tf.Session(graph=tf.Graph(), config=config) as sess, timer('fit_predict'):
        ks.backend.set_session(sess)
        model_in = ks.Input(shape=(X_train.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(lr=3e-3))
        for i in range(3):
            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]
```

TF trick

MLP

increase batch size

# Other Solutions

Popular models:

- Ridge
- GRU and Conv1D
- LightGBM
- Wordbatch FTRL, FM_FTRL (@anttip)

# 2nd place solution: 0.3889 by Mercaring (Nima & Chahhou)

- Concatenate brand and category with name
- Ridge on concatenated name + description: 0.418
- Sparse NN
- fastText NN, shared name and description embeddings
- Sparse NN on a different dataset
- Double batch size after each epoch

# 3rd place solution: 0.3905 by @whitebird

- CNN model: 0.400
- Wordbatch FM_FTRL: 0.415
- A lot of effort on optimization

# @sergeif magic feature: any model (inc. Ridge) to <0.410

- brand × name, brand × description
- category × name, category × description
- etc ...

```
brand="Apple", name="iPhone 8s new" ⇒
{"Apple_iPhone", "Apple_8s", "Apple_new"}
```

# Main differences of our approach

- One model kind, 3 datasets
- Train 12 models
- Sparse MLP model
- Early merge: almost all good ideas created after merging

Questions?

# First Layer Hidden Size

| Hidden size | Score (delta) |
|---|---|
| 128 | 0.3757 (+0.0024) |
| 256 | 0.3733 (+0.0000) |
| 384 | 0.3728 (−0.0005) |

# Binariezed Features, Classification

| Setup | Score (delta) |
|---|---|
| default | 0.3733 (+0.0000) |
| no binary | 0.3740 (+0.0007) |
| no clf | 0.3742 (+0.0009) |
| no both | 0.3748 (+0.0015) |