



Kaggler

Kaggler is a Python package for lightweight online machine learning algorithms and utility functions for ETL and data analysis. It is distributed under the MIT License.

Its online learning algorithms are inspired by Kaggle user [tintngu](#)'s code. It uses the sparse input format that handles large sparse data efficiently. Core code is optimized for speed by using Cython.

Installation

Dependencies

Python packages required are listed in [requirements.txt](#) * cython * h5py * hyperopt * lightgbm * ml_metrics * numpy/scipy * pandas * scikit-learn

Using pip

Python package is available at PyPi for pip installation:

```
1 pip install -U Kaggler
```

If installation fails because it cannot find [MurmurHash3.h](#), please add `.` to `LD_LIBRARY_PATH` as described [here](#).

From source code

If you want to install it from source code:

```
1 python setup.py build_ext --inplace
2 python setup.py install
```

Feature Engineering

One-Hot, Label, Target, Frequency, and Embedding Encoders for Categorical Features

```

1 import pandas as pd
2 from kaggle.preprocessing import OneHotEncoder, LabelEncoder,
   TargetEncoder, FrequencyEncoder, EmbeddingEncoder
3
4 trn = pd.read_csv('train.csv')
5 target_col = trn.columns[-1]
6 cat_cols = [col for col in trn.columns if trn[col].dtype == 'object']
7
8 ohe = OneHotEncoder(min_obs=100) # grouping all categories with less
   than 100 occurrences
9 lbe = LabelEncoder(min_obs=100) # grouping all categories with less
   than 100 occurrences
10 te = TargetEncoder() # replacing each category with the
   average target value of the category
11 fe = FrequencyEncoder() # replacing each category with the
   frequency value of the category
12 ee = EmbeddingEncoder() # mapping each category to a vector of
   real numbers
13
14 X_ohe = ohe.fit_transform(trn[cat_cols]) # X_ohe is a scipy
   sparse matrix
15 trn[cat_cols] = lbe.fit_transform(trn[cat_cols])
16 trn[cat_cols] = te.fit_transform(trn[cat_cols])
17 trn[cat_cols] = fe.fit_transform(trn[cat_cols])
18 X_ee = ee.fit_transform(trn[cat_cols], trn[target_col]) # X_ee
   is a numpy matrix
19
20 tst = pd.read_csv('test.csv')
21 X_ohe = ohe.transform(tst[cat_cols])
22 tst[cat_cols] = lbe.transform(tst[cat_cols])
23 tst[cat_cols] = te.transform(tst[cat_cols])
24 tst[cat_cols] = fe.transform(tst[cat_cols])
25 X_ee = ee.transform(tst[cat_cols])

```

Denoising AutoEncoder (DAE)

For reference for DAE, please check out Vincent et al. (2010), “Stacked Denoising Autoencoders”.

```

1 import pandas as pd
2 from kaggle.preprocessing import DAE
3
4 trn = pd.read_csv('train.csv')
5 tst = pd.read_csv('test.csv')
6 target_col = trn.columns[-1]
7 cat_cols = [col for col in trn.columns if trn[col].dtype == 'object']
8 num_cols = [col for col in trn.columns if col not in cat_cols + [
   target_col]]
9

```

```

10 # Default DAE with only the swapping noise and a single encoder/decoder
    pair.
11 dae = DAE(cat_cols=cat_cols, num_cols=num_cols, n_encoding=128)
12 X = dae.fit_transform(pd.concat([trn, tst], axis=0)) # encoding
    input features into the encoding vectors with size of 128
13
14 # Stacked DAE with the Gaussian noise, swapping noise and zero masking
    in 3 pairs of the encoder/decoder.
15 sdae = DAE(cat_cols=cat_cols, num_cols=num_cols, n_encoding=128,
    n_layer=3,
16             noise_std=.05, swap_prob=.2, mask_prob=.1)
17 X = sdae.fit_transform(pd.concat([trn, tst], axis=0))
18
19 # Supervised DAE with the Gaussian noise, swapping noise and zero
    masking in 3 encoders in the encoder/decoder pair.
20 sdae = SDAE(cat_cols=cat_cols, num_cols=num_cols, n_encoding=128,
    n_encoder=3,
21             noise_std=.05, swap_prob=.2, mask_prob=.1)
22 X = sdae.fit_transform(trn, trn[target_col])

```

AutoML

Feature Selection & Hyperparameter Tuning

```

1 import pandas as pd
2 from sklearn.datasets import make_classification
3 from sklearn.model_selection import train_test_split
4 from kaggle.metrics import auc
5 from kaggle.model import AutoLGB
6
7
8 RANDOM_SEED = 42
9 N_OBS = 10000
10 N_FEATURE = 100
11 N_IMP_FEATURE = 20
12
13 X, y = make_classification(n_samples=N_OBS,
14                           n_features=N_FEATURE,
15                           n_informative=N_IMP_FEATURE,
16                           random_state=RANDOM_SEED)
17 X = pd.DataFrame(X, columns=['x{}'.format(i) for i in range(X.shape[1])
18 ])
18 y = pd.Series(y)
19
20 X_trn, X_tst, y_trn, y_tst = train_test_split(X, y,
21                                               test_size=.2,
22                                               random_state=
23                                               RANDOM_SEED)

```

```
23
24 model = AutoLGB(objective='binary', metric='auc')
25 model.tune(X_trn, y_trn)
26 model.fit(X_trn, y_trn)
27 p = model.predict(X_tst)
28 print('AUC: {:.4f}'.format(auc(y_tst, p)))
```

Ensemble

Netflix Blending

```
1 import numpy as np
2 from kaggler.ensemble import netflix
3 from kaggler.metrics import rmse
4
5 # Load the predictions of input models for ensemble
6 p1 = np.loadtxt('model1_prediction.txt')
7 p2 = np.loadtxt('model2_prediction.txt')
8 p3 = np.loadtxt('model3_prediction.txt')
9
10 # Calculate RMSEs of model predictions and all-zero prediction.
11 # At a competition, RMSEs (or RMLSEs) of submissions can be used.
12 y = np.loadtxt('target.txt')
13 e0 = rmse(y, np.zeros_like(y))
14 e1 = rmse(y, p1)
15 e2 = rmse(y, p2)
16 e3 = rmse(y, p3)
17
18 p, w = netflix([e1, e2, e3], [p1, p2, p3], e0, l=0.0001) # l is an
    optional regularization parameter.
```

Algorithms

Currently algorithms available are as follows:

Online learning algorithms

- Stochastic Gradient Descent (SGD)
- Follow-the-Regularized-Leader (FTRL)
- Factorization Machine (FM)
- Neural Networks (NN) - with a single (NN) or two (NN_H2) ReLU hidden layers
- Decision Tree

Batch learning algorithm

- Neural Networks (NN) - with a single hidden layer and L-BFGS optimization

Examples

```
1 from kaggler.online_model import SGD, FTRL, FM, NN
2
3 # SGD
4 clf = SGD(a=.01,                # learning rate
5           l1=1e-6,              # L1 regularization parameter
6           l2=1e-6,              # L2 regularization parameter
7           n=2**20,              # number of hashed features
8           epoch=10,             # number of epochs
9           interaction=True)      # use feature interaction or not
10
11 # FTRL
12 clf = FTRL(a=.1,                # alpha in the per-coordinate rate
13            b=1,                  # beta in the per-coordinate rate
14            l1=1.,               # L1 regularization parameter
15            l2=1.,               # L2 regularization parameter
16            n=2**20,             # number of hashed features
17            epoch=1,             # number of epochs
18            interaction=True)     # use feature interaction or not
19
20 # FM
21 clf = FM(n=1e5,                 # number of features
22          epoch=100,             # number of epochs
23          dim=4,                 # size of factors for interactions
24          a=.01)                 # learning rate
25
26 # NN
27 clf = NN(n=1e5,                 # number of features
28          epoch=10,              # number of epochs
29          h=16,                  # number of hidden units
30          a=.1,                  # learning rate
31          l2=1e-6)               # L2 regularization parameter
32
33 # online training and prediction directly with a libsvm file
34 for x, y in clf.read_sparse('train.sparse'):
35     p = clf.predict_one(x)      # predict for an input
36     clf.update_one(x, p - y)    # update the model with the target
37                                 # using error
38
39 for x, _ in clf.read_sparse('test.sparse'):
40     p = clf.predict_one(x)
41
42 # online training and prediction with a scipy sparse matrix
43 from kaggler import load_data
```

```
43
44 X, y = load_data('train.sps')
45
46 clf.fit(X, y)
47 p = clf.predict(X)
```

Data I/O

Kaggler supports CSV ([.csv](#)), LibSVM ([.sps](#)), and HDF5 ([.h5](#)) file formats:

```
1 # CSV format: target,feature1,feature2,...
2 1,1,0,0,1,0.5
3 0,0,1,0,0,5
4
5 # LibSVM format: target feature-index1:feature-value1 feature-index2:
   feature-value2
6 1 1:1 4:1 5:0.5
7 0 2:1 5:1
8
9 # HDF5
10 - issparse: binary flag indicating whether it stores sparse data or not
   .
11 - target: stores a target variable as a numpy.array
12 - shape: available only if issparse == 1. shape of scipy.sparse.
   csr_matrix
13 - indices: available only if issparse == 1. indices of scipy.sparse.
   csr_matrix
14 - indptr: available only if issparse == 1. indptr of scipy.sparse.
   csr_matrix
15 - data: dense feature matrix if issparse == 0 else data of scipy.sparse
   .csr_matrix
```

```
1 from kaggler.data_io import load_data, save_data
2
3 X, y = load_data('train.csv')    # use the first column as a target
   variable
4 X, y = load_data('train.h5')     # load the feature matrix and target
   vector from a HDF5 file.
5 X, y = load_data('train.sps')    # load the feature matrix and target
   vector from LibSVM file.
6
7 save_data(X, y, 'train.csv')
8 save_data(X, y, 'train.h5')
9 save_data(X, y, 'train.sps')
```

Documentation

Package documentation is available at [here](#)