easyrec

Release 0.0.1

Zhiwei Xu

Aug 19, 2021

GET STARTED

CHAPTER

ONE

PREREQUISITES

- Windows, Linux or MacOS
- Python 3+
- Tensorflow 2+

CHAPTER

TWO

INSTALLATION

2.1 Prepare environment

1. The very first thing to install *easyrec* is to ensure that Python is installed. If not, it is recommended to use Anaconda instead of pure Python.
2. Create a conda virtual environment.

```
conda create -n easyrec python=3.7
```

1. Activate the virtual environment.

```
conda activate easyrec
```

2.2 Install *easyrec*

2.2.1 PyPI

The easiest way to install *easyrec* is PyPI. You can directly run the following command to install:

```
pip install easyrec-python
```

2.2.2 Source

In addition, you can also install *easyrec* from source. First, locate the directory that you want to keep codes:

```
cd /path/for/codes/
```

Next, clone the source codes from Github:

```
git clone git@github.com:xu-zhiwei/easyrec.git
```

Finally, install *easyrec* in your Python environment:

```
cd easyrec
pip install requirements.txt
python setup.py install # or "python setup.py develop" for developers who wants to␣
↪modify the codes
```

2.3 Verification

After installation, you can verify it by the following:

1. Switch to Python environment.

```
python
```

1. Verify installation.

```
import easyrec
```

The above code is supposed to run successfully upon you finish installation.

CHAPTER

THREE

BACKGROUND OF EXAMPLE

easyrec provides a number of existing models proposed in recommender system fields. It is extremely easy to use and what you only need is to prepare the input of models.

To quickly acquire the usage of *easyrec*, we have finished some examples and make them open-sourced in Github.

Here we take MovieLens 1M Dataset (short as ml-1m) for dataset and Factorization Machine (FM) for model as an example.

CHAPTER

FOUR

PREPARE DATASET

After you have downloaded the dataset, you may clean the data as follows:

```
from pathlib import Path
import pandas as pd

dataset_path = Path('/path/for/dataset/')

# load ratings.dat
rating_df = pd.read_csv(dataset_path / 'ratings.dat', sep='::', engine='python',␣
↪header=None,
                        names=['user_id', 'item_id', 'ctr', 'timestamp'])
rating_df.loc[rating_df['ctr'] <= 3, 'ctr'] = 0
rating_df.loc[rating_df['ctr'] > 3, 'ctr'] = 1
rating_df.pop('timestamp')

# load users.dat
user_df = pd.read_csv(dataset_path / 'users.dat', sep='::', engine='python', header=None,
                      names=['user_id', 'sex_id', 'age_id', 'occupation_id', 'zip_code_id
↪'])
user_df['age_id'] = user_df['age_id'].astype(str)
user_df['occupation_id'] = user_df['occupation_id'].astype(str)
user_df['zip_code_id'] = user_df['zip_code_id'].astype(str)

# load movies.dat
item_df = pd.read_csv(dataset_path / 'movies.dat', sep='::', engine='python',␣
↪header=None,
                      names=['item_id', 'title', 'genre_ids'])
item_df.pop('title')  # title is not used in the example
item_df['genre_ids'] = item_df['genre_ids'].apply(lambda x: x.split('|'))

# join 3 tables
df = pd.merge(rating_df, user_df, how='left', on='user_id')
df = pd.merge(df, item_df, how='left', on='item_id')
```

Then, based on feature columns in Tensorflow 2, you can formally define the format of input for models and obtain the dataset generator.

Note: detailed introduction of feature columns is illustrated in Tutorial.

```
import tensorflow as tf
```

(continues on next page)

(continued from previous page)

```
# define the feature columns
categorical_column_with_identity = tf.feature_column.categorical_column_with_identity
categorical_column_with_vocabulary_list = tf.feature_column.categorical_column_with_
↪vocabulary_list
one_hot_feature_columns = [
    categorical_column_with_identity(key='user_id', num_buckets=df['user_id'].max() + 1,␣
↪default_value=0),
    categorical_column_with_vocabulary_list(
        key='sex_id', vocabulary_list=set(df['sex_id'].values), num_oov_buckets=1),
    categorical_column_with_vocabulary_list(
        key='age_id', vocabulary_list=set(df['age_id'].values), num_oov_buckets=1),
    categorical_column_with_vocabulary_list(
        key='occupation_id', vocabulary_list=set(df['occupation_id'].values), num_oov_
↪buckets=1),
    categorical_column_with_vocabulary_list(
        key='zip_code_id', vocabulary_list=set(df['zip_code_id'].values), num_oov_
↪buckets=1),
    categorical_column_with_identity(key='item_id', num_buckets=df['item_id'].max() + 1,␣
↪default_value=0),
]


# construct dataset generator
def train_validation_test_split(dataset: tf.data.Dataset,
                                dataset_size: int,
                                train_ratio: float,
                                validation_ratio: float
                                ) -> Tuple[tf.data.Dataset, tf.data.Dataset, tf.data.
↪Dataset]:
    if train_ratio + validation_ratio >= 1:
        raise ValueError('train_size + validation_size should be less than 1')
    train_size, validation_size = round(train_ratio * dataset_size), round(validation_
↪ratio * dataset_size)
    train_dataset = dataset.take(train_size)
    test_dataset = dataset.skip(train_size)
    validation_dataset = test_dataset.take(validation_size)
    test_dataset = test_dataset.skip(validation_size)
    return train_dataset, validation_dataset, test_dataset


def transform_ragged_lists_to_sparse_tensor(ragged_lists: list):
    indices, values = [], []
    max_length = 0
    for i, ragged_list in enumerate(ragged_lists):
        for j, value in enumerate(ragged_list):
            indices.append((i, j))
            values.append(value)
        max_length = max(max_length, len(ragged_list))

    return tf.SparseTensor(
        indices=indices,
        values=values,
```

(continues on next page)

(continued from previous page)

```
        dense_shape=(len(ragged_lists), max_length)
    )


train_ratio, validation_ratio, test_ratio = [0.6, 0.2, 0.2]
batch_size = 128

labels = df.pop('ctr')
features = dict(df)
features['genre_ids'] = transform_ragged_lists_to_sparse_tensor(features['genre_ids'])
dataset = tf.data.Dataset.from_tensor_slices((features, labels))
dataset = dataset.shuffle(buffer_size=200, seed=42)
train_dataset, validation_dataset, test_dataset = train_validation_test_split(dataset,
                                                                               len(df),
                                                                               train_
↪ratio,
                                                                               validation_
↪ratio
                                                                               )
train_dataset = train_dataset.batch(batch_size)
validation_dataset = validation_dataset.batch(batch_size)
test_dataset = test_dataset.batch(batch_size)
```

CHAPTER

FIVE

LOW-LEVEL APIS

Next, train the model according to Low-level APIs (or the High-level APIs mentioned below).

```
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.metrics import Mean, AUC
from tensorflow.keras.optimizers import SGD

learning_rate = 1e-1
epochs = 50

output_ckpt_path = Path(output_ckpt_path)
model = FM(
    one_hot_feature_columns,
    k=32
)
start_epoch = 0

loss_obj = BinaryCrossentropy()
optimizer = SGD(learning_rate=learning_rate)

train_loss = Mean(name='train_loss')
train_auc = AUC(name='train_auc')
validation_loss = Mean(name='validation_loss')
validation_auc = AUC(name='validation_auc')
best_auc = 0


@tf.function
def train_step(x, y):
    with tf.GradientTape() as tape:
        predictions = model(x)
        loss = loss_obj(y, predictions)
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))

    train_loss(loss)
    train_auc(y, predictions)


@tf.function
def validation_step(x, y):
```

(continues on next page)

(continued from previous page)

```
    predictions = model(x)
    loss = loss_obj(y, predictions)

    validation_loss(loss)
    validation_auc(y, predictions)


# train
for epoch in range(start_epoch, epochs):
    train_loss.reset_states()
    train_auc.reset_states()
    validation_loss.reset_states()
    validation_auc.reset_states()

    for features, labels in train_dataset:
        train_step(features, labels)
    for features, labels in validation_dataset:
        validation_step(features, labels)

    print('epoch: {}, train_loss: {}, train_auc: {}'.format(epoch + 1, train_loss.
↪result().numpy(),
                                                            train_auc.result().numpy()))
    print('epoch: {}, validation_loss: {}, validation_auc: {}'.format(epoch + 1,␣
↪validation_loss.result().numpy(),
                                                                validation_auc.
↪result().numpy()))

    model.save(output_ckpt_path / str(epoch + 1))
    if best_auc < validation_auc.result().numpy():
        best_auc = validation_auc.result().numpy()
        model.save(output_ckpt_path / 'best')
```

Finally, the model parameter with the best evaluation result can be loaded and carry out inference.

```
@tf.function
def test_step(x, y):
    predictions = model(x)
    loss = loss_obj(y, predictions)

    test_loss(loss)
    test_auc(y, predictions)


model = tf.keras.models.load_model(output_ckpt_path / 'best')
test_loss = Mean(name='test_loss')
test_auc = AUC(name='test_auc')
for features, labels in test_dataset:
    test_step(features, labels)
print('test_loss: {}, test_auc: {}'.format(test_loss.result().numpy(),
                                           test_auc.result().numpy()))
```

CHAPTER

SIX

HIGH-LEVEL APIS

Coming sooooooooon!

CHAPTER

SEVEN

FEATURE COLUMNS

The most important part of *easyrec* is feature columns, which basically determines whether you can train models within your customized dataset.

In *easyrec*, the type of feature columns can be concluded into 2 groups, i.e., by data type or by usage.

7.1 Data type-aware feature columns

7.1.1 One hot

One hot feature columns indicate a list of **categorical feature columns**, and a data sample can belong to one and only one of the categories.

```
"""
Example Args in Functions:
    one_hot_feature_columns: List[CategoricalColumn] encodes one hot feature fields,␣
↪such as sex_id.
"""
import tensorflow as tf

categorical_column_with_identity = tf.feature_column.categorical_column_with_identity
categorical_column_with_vocabulary_list = tf.feature_column.categorical_column_with_
↪vocabulary_list
one_hot_feature_columns = [
    categorical_column_with_identity(key='user_id', num_buckets=df['user_id'].max() + 1,␣
↪default_value=0),
    categorical_column_with_vocabulary_list(
        key='sex_id', vocabulary_list=set(df['sex_id'].values), num_oov_buckets=1),
    categorical_column_with_vocabulary_list(
        key='age_id', vocabulary_list=set(df['age_id'].values), num_oov_buckets=1),
    categorical_column_with_vocabulary_list(
        key='occupation_id', vocabulary_list=set(df['occupation_id'].values), num_oov_
↪buckets=1),
    categorical_column_with_vocabulary_list(
        key='zip_code_id', vocabulary_list=set(df['zip_code_id'].values), num_oov_
↪buckets=1),
    categorical_column_with_identity(key='item_id', num_buckets=df['item_id'].max() + 1,␣
↪default_value=0),
]
```

7.1.2 Multi hot

Multi hot feature columns indicate a list of **categorical feature columns**, and a data sample can belong to one or more than one of the categories.

```
"""
Example Args in Function:
    multi_hot_feature_columns: List[CategoricalColumn] encodes multi hot feature fields,␣
↪such as
        historical_item_ids.
"""
import tensorflow as tf

categorical_column_with_vocabulary_list = tf.feature_column.categorical_column_with_
↪vocabulary_list
multi_hot_feature_columns = [
    categorical_column_with_vocabulary_list(
        key='genre_ids', vocabulary_list=get_vocabulary_list_from_ragged_list_
↪series(item_df['genre_ids']),
        num_oov_buckets=1
    )
]
```

7.1.3 Dense

Dense feature columns indicate a list of **numerical feature columns**.

```
"""
Example Args in Function:
    dense_feature_columns: List[NumericalColumn] encodes numerical feature fields, such␣
↪as age.
"""
import tensorflow as tf

dense_feature_columns = [
    tf.feature_column.numeric_column(key='age')
]
```

7.2 Usage-aware feature columns

These feature columns indicate a list of feature columns that can be **directly** feed into model.

```
"""
Example Args:
    user_feature_columns: List[FeatureColumn] to directly feed into tf.keras.layers.
↪DenseFeatures, which
        basically contains user feature fields.
    item_feature_columns: List[FeatureColumn] to directly feed into tf.keras.layers.
↪DenseFeatures, which
        basically contains item feature fields.
```

(continues on next page)

(continued from previous page)

```
    feature columns: List[FeatureColumn] to directly feed into tf.keras.layers.
↪DenseFeatures, which basically
        contains all feature fields.
"""
import tensorflow as tf

categorical_column_with_identity = tf.feature_column.categorical_column_with_identity
categorical_column_with_vocabulary_list = tf.feature_column.categorical_column_with_
↪vocabulary_list
indicator_column = tf.feature_column.indicator_column
user_feature_columns = [
    categorical_column_with_identity(key='user_id', num_buckets=df['user_id'].max() + 1,␣
↪default_value=0),
    categorical_column_with_vocabulary_list(
        key='sex_id', vocabulary_list=set(df['sex_id'].values), num_oov_buckets=1),
    categorical_column_with_vocabulary_list(
        key='age_id', vocabulary_list=set(df['age_id'].values), num_oov_buckets=1),
    categorical_column_with_vocabulary_list(
        key='occupation_id', vocabulary_list=set(df['occupation_id'].values), num_oov_
↪buckets=1),
    categorical_column_with_vocabulary_list(
        key='zip_code_id', vocabulary_list=set(df['zip_code_id'].values), num_oov_
↪buckets=1),
]
```

CHAPTER

EIGHT

EASYREC.MODELS

8.1 easyrec.models.afm

class easyrec.models.afm.**AFM**(**args*, ***kwargs*)

Bases: keras.engine.training.Model

Attentional Factorization Machines (AFM). Reference: Jun Xiao et al. Attentional Factorization Machines:Learning the Weight of Feature Interactions via Attention Networks. arXiv. 2017.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **k** – Dimension of the second-order weights.

call(*inputs*, *training=None*, *mask=None*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.2 easyrec.models.autoint

class easyrec.models.autoint.**AutoInt**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Automatic Feature Interaction (AutoInt). Reference: AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. CIKM. 2019.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **multi_hot_feature_columns** – List[CategoricalColumn] encodes multi hot feature fields, such as historical_item_ids.
- **dense_feature_columns** – List[NumericalColumn] encodes numerical feature fields, such as age.
- **embedding_dimension** – Dimension of embedded Column.
- **num_heads** – Number of heads.
- **attention_qkv_dimension** – Dimension of Query, Key and Value in self attention.
- **attention_output_dimension** – Dimension of output in self attention.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.3 easyrec.models.dcn

class easyrec.models.dcn.**DCN**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Deep & Cross Network (DCN). Reference: Ruoxi Wang et al. Deep & Cross Network for ad Click Predictions. ADKDD. 2017.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.

- **multi_hot_feature_columns** – List[CategoricalColumn] encodes multi hot feature fields, such as historical_item_ids.
- **dense_feature_columns** – List[NumericalColumn] encodes numerical feature fields, such as age.
- **embedding_dimension** – Dimension of embedded CategoricalColumn.
- **num_crosses** – Number of crosses.
- **deep_units_list** – Dimension of fully connected stack outputs in deep dense block.
- **deep_activation** – Activation to use in deep dense block.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.4 easyrec.models.deep_crossing

class easyrec.models.deep_crossing.**DeepCrossing**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Deep Crossing. Reference: Ying Shan et al. Deep Crossing: Web-Scale Modeling without Manually Crafted Combinatorial Features. KDD. 2016.

Parameters

- **feature_columns** – List[FeatureColumn] to directly feed into tf.keras.layers.DenseFeatures, which basically contains all feature fields.
- **num_residual_blocks** – Number of residual blocks.
- **residual_units_list** – Dimension of fully connected stack outputs in residual block.
- **residual_activation** – Activation to use in residual block.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.5 easyrec.models.deepfm

class easyrec.models.deepfm.**DeepFM**(**args*, ***kwargs*)

Bases: keras.engine.training.Model

Deep Factorization Machine (DeepFM). Reference: Huifeng Guo et al. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. arXiv. 2017.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **k** – Dimension of the second-order weights.
- **deep_units_list** – Dimension of fully connected stack outputs in deep block.
- **deep_activation** – Activation to use in deep block.

call(*inputs*, *training=None*, *mask=None*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.6 easyrec.models.dssm

class easyrec.models.dssm.**DSSM**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Deep Structured Semantic Model (DSSM). Po-Sen Huang et al. Learning Deep Structured Semantic Models for Web Search using Clickthrough Data. CIKM. 2013.

Parameters

- **user_feature_columns** – List[FeatureColumn] to directly feed into tf.keras.layers.DenseFeatures, which basically contains user feature fields.
- **item_feature_columns** – List[FeatureColumn] to directly feed into tf.keras.layers.DenseFeatures, which basically contains item feature fields.
- **user_units_list** – Dimension of fully connected stack outputs in user dense block.
- **user_activation** – Activation to use in user dense block.
- **item_units_list** – Dimension of fully connected stack outputs in item dense block.
- **item_activation** – Activation to use in item dense block.
- **score_function** – Final output function to combine the user embedding and item embedding.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.7 easyrec.models.ffm

class easyrec.models.ffm.**FFM**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Field-aware Factorization Machine (FFM). Reference: Yuchin Juan et al. Field-aware Factorization Machines for CTR Prediction. RecSys. 2016.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **k** – Dimension of the second-order weights.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.8 easyrec.models.fm

class easyrec.models.fm.**FM**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Factorization Machine (FM). Reference: Steffen Rendle. Factorization Machines. ICDM. 2010.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **k** – Dimension of the second-order weights.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.9 easyrec.models.fnn

class easyrec.models.fnn.**FNN**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Factorization-machine supported Neural Network (FNN). Reference: Weinan Zhang. Deep Learning over Multi-field Categorical Data – A Case Study on User Response Prediction. ECIR. 2016.

fm: Pretrained Factorization Machines. one_hot_feature_columns: List[CategoricalColumn] encodes one hot feature fields, such as sex_id. units_list: Dimension of fully connected stack outputs. activation: Activation to use.

call(*inputs*, *pretraining=True*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.10 easyrec.models.lr

class easyrec.models.lr.**LR**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Logisitic Regression (LR).

Parameters **feature_columns** – List[FeatureColumn] to directly feed into tf.keras.layers.DenseFeatures, which basically contains all feature fields.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.11 easyrec.models.mlp

class easyrec.models.mlp.**MLP**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Multi-layer Perceptron (MLP).

Parameters

- **feature_columns** – List[FeatureColumn] to directly feed into tf.keras.layers.DenseFeatures, which basically contains all feature fields.
- **units_list** – Dimension of fully connected stack outputs.
- **activation** – Activation to use.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.12 easyrec.models.mmoe

class easyrec.models.mmoe.**MMOE**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Multi-gate Mixture-of-Experts. Reference: Jiaqi Ma et al. Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts. KDD. 2018.

Parameters

- **feature_columns** – List[FeatureColumn] to directly feed into tf.keras.layers.DenseFeatures, which basically contains all feature fields.
- **num_experts** – Number of experts.
- **expert_units_list** – Dimension of fully connected stack outputs in expert dense block.
- **expert_activation** – Activation to use in expert dense block.
- **num_towers** – Number of towers (tasks).
- **tower_units_list** – Dimension of fully connected stack outputs in tower dense block.

- **tower_activation** – Activation to use in tower dense block.

call(*inputs*, *use_tower=0*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the __*call*__ method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.13 easyrec.models.neumf

class easyrec.models.neumf.**NeuMF**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Neural Matrix Factorization (NeuMF). Xiangnan He et al. Neural Factorization Machines for Sparse Predictive Analytics. SIGIR. 2017.

Parameters

- **user_feature_column** – CategoricalColumn to represent user_id.
- **item_feature_column** – CategoricalColumn to represent item_id.
- **user_embedding_dimension** – Dimension of user embedding.
- **item_embedding_dimension** – Dimension of item embedding.
- **units_list** – Dimension of fully connected stack outputs.
- **activation** – Activation to use.
- **alpha** – Tendency parameter for GMF, thus, 1 - alpha is used for MLP.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the __*call*__ method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.

- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.14 easyrec.models.nfm

class easyrec.models.nfm.**NFM**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Neural Factorization Machine (NFM). Xiangnan He et al. Neural Factorization Machines for Sparse Predictive Analytics. SIGIR. 2017.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **k** – Dimension of the second-order weights.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.15 easyrec.models.pnn

class easyrec.models.pnn.**PNN**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Product-based Neural Network (PNN). Reference: Yanru Qu et al. Product-based Neural Networks for User Response Prediction. ICDM. 2016.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **multi_hot_feature_columns** – List[CategoricalColumn] encodes multi hot feature fields, such as historical_item_ids.
- **embedding_dimension** – embedding dimension of each field.
- **use_inner_product** – whether to use IPNN.

- **use_outer_product** – whether to use OPNN.
- **units_list** – Dimension of fully connected stack outputs.
- **activation** – Activation to use.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.16 easyrec.models.wide_and_deep

class easyrec.models.wide_and_deep.**WideAndDeep**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Wide & Deep. Reference: Heng-Tze Cheng et al. Wide & Deep Learning for Recommender Systems. RecSys. 2016.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **multi_hot_feature_columns** – List[CategoricalColumn] encodes multi hot feature fields, such as historical_item_ids.
- **dense_feature_columns** – List[NumericalColumn] encodes numerical feature fields, such as age.
- **embedding_dimension** – Dimension of embedded CategoricalColumn.
- **deep_units_list** – Dimension of fully connected stack outputs in deep dense block.
- **deep_activation** – Activation to use in deep dense block.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

8.17 easyrec.models.xdeepfm

class easyrec.models.xdeepfm.**xDeepFM**(**args*, ***kwargs*)

Bases: keras.engine.training.Model

Extreme Deep Factorization Machine (xDeepFM). Reference: Jianxun Lian et al. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. KDD. 2018.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **multi_hot_feature_columns** – List[CategoricalColumn] encodes multi hot feature fields, such as historical_item_ids.
- **k** – Dimension of the second-order weights.
- **deep_units_list** – Dimension of fully connected stack outputs in deep block.
- **deep_activation** – Activation to use in deep block.
- **cross_units_list** – Number of fields in the cross layer.

call(*inputs*, *training=None*, *mask=None*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

CHAPTER

NINE

EASYREC.BLOCKS

9.1 easyrec.blocks.interaction

class easyrec.blocks.interaction.**AFM**(**args*, ***kwargs*)

Bases: keras.engine.training.Model

Attentional factorization machine layer.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **k** – Dimension of the second-order weights.

call(*inputs*, *training=None*, *mask=None*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

class easyrec.blocks.interaction.**FFM**(**args*, ***kwargs*)

Bases: keras.engine.training.Model

Field-aware factorization machine layer.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **k** – Dimension of the second-order weights.

call(*inputs*, **args*, ***kwargs*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

class easyrec.blocks.interaction.**FM**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Factorization machine layer using vector w and matrix v.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **k** – Dimension of the second-order weights.

call(*inputs*, **args*, ***kwargs*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

class easyrec.blocks.interaction.**NFM**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Neural factorization machine layer.

Parameters

- **one_hot_feature_columns** – List[CategoricalColumn] encodes one hot feature fields, such as sex_id.
- **k** – Dimension of the second-order weights.
- **units_list** – Dimension of fully connected stack outputs.

- **activation** – Activation to use.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

9.2 easyrec.blocks.nn

class easyrec.blocks.nn.**DenseBlock**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Multi-perception layer.

Parameters

- **units_list** – Dimension of fully connected stack outputs.
- **activation** – Activation to use.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

class easyrec.blocks.nn.**MultiHeadSelfAttention**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Multi-head self attention layer.

Parameters

- **input_dimension** – Dimension of input.
- **qkv_dimension** – Dimension of Query, Key and Value.
- **num_heads** – Number of heads.
- **output_dimension** – Dimension of final output.
- **use_normalization** – Whether to use normalization in Query * Key process.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

class easyrec.blocks.nn.**ResidualBlock**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Residual layer.

call(*inputs*, *training=None*, *mask=None*)
Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

class easyrec.blocks.nn.**SelfAttention**(**args*, ***kwargs*)
Bases: keras.engine.training.Model

Self attention layer.

Parameters

- **input_dimension** – Dimension of input.
- **qkv_dimension** – Dimension of Query, Key and Value.

- **use_normalization** – Whether to use normalization in Query * Key process.

call(*inputs*, *training=None*, *mask=None*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing *tf.keras.Model*. To call a model on an input, always use the *__call__* method, i.e. *model(inputs)*, which relies on the underlying *call* method.

Parameters

- **inputs** – Input tensor, or dict/list/tuple of input tensors.
- **training** – Boolean or boolean scalar tensor, indicating whether to run the *Network* in training mode or inference mode.
- **mask** – A mask or list of masks. A mask can be either a tensor or None (no mask).

Returns A tensor if there is a single output, or a list of tensors if there are more than one outputs.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

e

INDEX

L

M

N

P

R

S

W

X