Learning how the algorithm works

Spotlight has 2 usages:

1. Factorization models
2. Sequential models

1. Factorization Models

Factorization models are a class of machine learning models commonly used for collaborative filtering in recommendation systems. Collaborative filtering aims to make automatic **predictions about the interests** of a user **by collecting preferences from many users** (collaborating). Factorization models are used for matrix factorization, where a large user-item interaction matrix is decomposed into lower-dimensional matrices.

1.1 Implicit factorization models

The class, ImplicitFactorizationModel, represents an implicit feedback matrix factorization model designed for recommender systems. It adopts a classic matrix factorization approach where latent vectors are employed to represent both users and items. The dot product of these latent vectors yields the predicted score for a user-item pair. The latent representation is implemented using BilinearNet from the spotlight.factorization.representations module.

The model employs negative sampling during training: for each known user-item pair, one or more items are randomly sampled as negatives, expressing a lack of preference by the user for the sampled items.

class spotlight.factorization.implicit.ImplicitFactorizationModel(loss='pointwise', embedding_dim=32, n_iter=10, batch_size=256, l2=0.0, learning_rate=0.01, optimizer_func=None, use_cuda=False, representation=None, sparse=False, random_state=None, num_negative_samples=5)

Parameters explained:

- **loss** (string, optional): Specifies the loss function, which can be 'pointwise', 'bpr', 'hinge', or 'adaptive hinge', corresponding to losses from spotlight.losses.
- **embedding_dim** (int, optional): Number of embedding dimensions for users and items.
- **n_iter** (int, optional): Number of iterations during training.
- **batch_size** (int, optional): Minibatch size.
- **l2** (float, optional): L2 loss penalty.
- **learning_rate** (float, optional): Initial learning rate.
- **optimizer_func** (function, optional): Custom PyTorch optimizer function, overriding l2 and learning rate. Defaults to ADAM if not supplied.
- **use_cuda** (boolean, optional): Boolean flag to run the model on a GPU.
- **representation** (a representation module, optional): If provided, overrides default settings and serves as the main network module in the model, allowing full freedom to specify the network topology.
- **sparse** (boolean, optional): Specifies whether to use sparse gradients for embedding layers.
- **random_state** (instance of numpy.random.RandomState, optional): Random state for reproducibility.
- **num_negative_samples** (int, optional): Number of negative samples to generate for adaptive hinge loss.

fit(interactions, verbose=False)

Parameters explained:

- **interactions** (spotlight.interactions.Interactions) – The input dataset.
- **verbose** (bool) – Output additional information about current epoch and loss.

`predict(user_ids, item_ids=None)`

Parameters explained:
- **user_ids** (int or array): If provided as an integer, the function predicts recommendation scores for this user across all items specified in item_ids. If it's an array, the function predicts scores for all combinations of users and items defined by user_ids and item_ids.
- **item_ids** (array, optional): An optional array containing the item IDs for which prediction scores are desired. If this parameter is not supplied, the function will compute predictions for all items.

Returns:
- **predictions**: An array containing the predicted scores for all items specified in item_ids.

The output type is a NumPy array.

1.2 Explicit factorization models

This class defines an explicit feedback matrix factorization model used for solving recommendation problems. It implements a classic matrix factorization approach where latent vectors represent users and items. The dot product of these latent vectors predicts scores for user-item pairs. The model uses the BilinearNet class for latent representations.

Class `spotlight.factorization.explicit.ExplicitFactorizationModel`(loss='regression', embedding_dim=32, n_iter=10, batch_size=256, l2=0.0, learning_rate=0.01, optimizer_func=None, use_cuda=False, representation=None, sparse=False, random_state=None)

Parameters explained:
- **loss** (string, optional): Specifies the loss function, which can be 'regression', 'poisson', or 'logistic', corresponding to losses from the spotlight.losses module.
- **embedding_dim** (int, optional): Number of dimensions for the embedding of users and items.
- **n_iter** (int, optional): Number of iterations to run during training.
- **batch_size** (int, optional): Size of minibatches during training.
- **l2** (float, optional): L2 loss penalty.
- **learning_rate** (float, optional): Initial learning rate.
- **optimizer_func** (function, optional): Custom PyTorch optimizer function. It overrides l2 and learning rate if supplied; otherwise, ADAM is used by default.
- **use_cuda** (boolean, optional): Boolean flag to specify whether to run the model on a GPU.
- **representation** (a representation module, optional): If provided, it overrides default settings and serves as the main network module in the model. This allows full freedom to specify the network topology.
- **sparse** (boolean, optional): Specifies whether to use sparse gradients for embedding layers.
- **random_state** (instance of numpy.random.RandomState, optional): Random state used during the fitting process.

`fit(interactions, verbose=False):`

Fits the model to the provided interactions dataset. If called repeatedly, resumes training from the previous fit call.

Parameters explained:
- **interactions** (spotlight.interactions.Interactions) – The input dataset. Must have ratings.
- **verbose** (bool) – Output additional information about current epoch and loss.

`predict(user_ids, item_ids=None):`

Generates predictions for recommendation scores given user IDs and optional item IDs.

Parameters explained:
- **user_ids** (int or array): If it's an integer, the function predicts recommendation scores for this user across all items specified in item_ids. If it's an array, the function predicts scores for all pairs of (user, item) defined by user_ids and item_ids.
- **item_ids** (array, optional): An optional array containing the item IDs for which prediction scores are desired. If not provided, the function will compute predictions for all items.

Returns:
- **predictions**: An np.array containing the predicted scores for all items specified in item_ids.

2. Sequential models

Sequential models in the context of machine learning typically refer to models that process input data **sequentially, element by element**, in a specific order. These models are particularly relevant for tasks where the order or sequence of the input data is crucial for understanding and making predictions.

2.1 Implicit feedback models

Models for recommending items given a sequence of previous items a user has interacted with.

Class `spotlight.sequence.implicit.ImplicitSequenceModel`(loss='pointwise', representation='pooling', embedding_dim=32, n_iter=10, batch_size=256, l2=0.0, learning_rate=0.01, optimizer_func=None, use_cuda=False, sparse=False, random_state=None, num_negative_samples=5)

Parameters explained:

- **loss** (string, optional): Specifies the loss function for approximating a softmax with negative sampling. Options include 'pointwise', 'bpr', 'hinge', and 'adaptive_hinge', corresponding to losses from spotlight.losses.
- **representation** (string or instance of spotlight.sequence.representations, optional): Determines the sequence representation method. If a string, it should be one of 'pooling', 'cnn', 'lstm', or 'mixture'. If an instance, it must be one of the representations from spotlight.sequence.representations.
- **embedding_dim** (int, optional): Specifies the number of embedding dimensions for representing items. This value is overridden if representation is an instance of a representation class.
- **n_iter** (int, optional): Sets the number of iterations during training.
- **batch_size** (int, optional): Defines the minibatch size.
- **l2** (float, optional): Introduces an L2 loss penalty.
- **learning_rate** (float, optional): Specifies the initial learning rate.
- **optimizer_func** (function, optional): A function that takes module parameters as the first argument and returns a PyTorch optimizer instance. It overrides l2 and learning rate if supplied. If no optimizer is provided, ADAM is used by default.
- **use_cuda** (boolean, optional): Indicates whether to run the model on a GPU.
- **sparse** (boolean, optional): Decides whether to use sparse gradients for embedding layers.
- **random_state** (instance of numpy.random.RandomState, optional): Specifies the random state for fitting.
- **num_negative_samples** (int, optional): Determines the number of negative samples to generate for adaptive hinge loss.

Notes:

During the fitting process, the model computes the loss for each timestep of the supplied sequence.

`fit`(interactions, verbose=False)

Train the model using the provided sequence interactions dataset. If called multiple times, the training process will resume from where it left off in the previous fit call.

Parameters explained:

- **Interactions** (spotlight.interactions.SequenceInteractions): The input sequence dataset.

`predict`(sequences, item_ids=None)

Generate predictions based on a given sequence of interactions to forecast the next item in the sequence.

Parameters explained:

- **Sequences** (array, (1 x max_sequence_length)): An array containing the indices of items in the sequence.
- **Item_ids** (array (num_items x 1), optional): An optional array containing the item IDs for which prediction scores are desired. If not provided, predictions for all items will be computed.

Returns:

- **Predictions**: Predicted scores for all items in item_ids in an array.

LOSS FUNCTIONS

- The **pointwise, BPR, and hinge** losses are a good fit for **implicit** feedback models trained through negative sampling.
- The **regression and Poisson** losses are used for **explicit** feedback models.

How to use them:

`spotlight.losses.adaptive_hinge_loss(positive_predictions, negative_predictions, mask=None)`

`spotlight.losses.bpr_loss(positive_predictions, negative_predictions, mask=None)`

`spotlight.losses.hinge_loss(positive_predictions, negative_predictions, mask=None)`

`spotlight.losses.logistic_loss(observed_ratings, predicted_ratings)`

`spotlight.losses.pointwise_loss(positive_predictions, negative_predictions, mask=None)`

`spotlight.losses.poisson_loss(observed_ratings, predicted_ratings)`

`spotlight.losses.regression_loss(observed_ratings, predicted_ratings)`

EVALUATION

Choose the desired one:

`spotlight.evaluation.mrr_score(model, test, train=None)`

`spotlight.evaluation.precision_recall_score(model, test, train=None, k=10)`

`spotlight.evaluation.rmse_score(model, test)`

`spotlight.evaluation.sequence_mrr_score(model, test, exclude_preceding=False)`

`spotlight.evaluation.sequence_precision_recall_score(model, test, k=10, exclude_preceding=False)`