fastFM Documentation

Release 0.1.0

Immanuel Bayer

July 03, 2015

CONTENTS

CHAPTER

ONE

TUTORIALS

The following sections show how to use different features of the fastFM library. The focus in on how to use the library and not on the Factorization Machine model. I recommend to read [TIST2012] for examples that show how FM's can emulate and extend many matrix factorization model through feature engineering.

1.1 Regression with ALS Solver

We first set up a small toy dataset for a simple regression problem. Please refere to [SIGIR2011] for background information on the implemented ALS solver.

```
from fastFM.datasets import make_user_item_regression
from sklearn.cross_validation import train_test_split

# This sets up a small test dataset.
X, y, _ = make_user_item_regression(label_stdev=.4)
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

The number of iterations *n_iter*, the standart derivation *init_stdev* for The random initization and the number of hiden variables *rank* per feature have to be specified for each solver and task. The ALS solver requires us also to set the regularization for the first *l2_reg_w* and second order *l2_reg_V* interactions.

```
from fastFM import als
fm = als.FMRegression(n_iter=1000, init_stdev=0.1, rank=2, l2_reg_w=0.1, l2_reg_V=0.5)
fm.fit(X_train, y_train)
y_pred = fm.predict(X_test)
```

We can easely evaluate our model using the scikit-learn library.

```
from sklearn.metrics import mean_squared_error
'mse:', mean_squared_error(y_test, y_pred)
```

1.2 Logit Classification with SGD Solver

We convert the target of our toy dataset to -1/1 values as currently only binary classification is supported.

```
import numpy as np
# Convert dataset to binary classification task.
y_labels = np.ones_like(y)
y_labels[y < np.mean(y)] = -1
X_train, X_test, y_train, y_test = train_test_split(X, y_labels)
```

We could have used the ALS solver module for this problem as well but for the sake of illustration we use the SGD module instead. In addition to the hyperparameter from the privious example we need also to specifiy the SGD specific *step_size* parameter.

```
from fastFM import sgd
fm = sgd.FMClassification(n_iter=1000, init_stdev=0.1, l2_reg_w=0,
                          l2_reg_V=0, rank=2, step_size=0.1)
fm.fit(X_train, y_train)
y_pred = fm.predict(X_test)
```

All classifier can return not only the predicted targets but also privide, the *predict_proba* function to return the class probabilities instead.

```
y_pred_proba = fm.predict_proba(X_test)
```

Classification metrics such as the AUC score require the class probabilities as input.

```
from sklearn.metrics import accuracy_score, roc_auc_score
'acc:', accuracy_score(y_test, y_pred)
'auc:', roc_auc_score(y_test, y_pred_proba)
```

1.3 Bayesian Probit Classification with MCMC Solver

The MCMC module has the advantage that we need less hyperparameter as in any other module. This is because the model intigrates over the regularization parameters. The drawback is that we need to use *fit_predict* because fitting and the model and predicting new samples need to be done together. If we call *predict* on a mcmc returns predictions based on the last parameters of the MCMC chain, this can be used for diagnostic purposes but the predictions are usully not as good as averaged predictions retured by *fit_predict*.

cant be seperated .

```
from fastFM import mcmc
fm = mcmc.FMClassification(n_iter=1000, rank=2, init_stdev=0.1)
```

Our last example showes how to use the MCMC module for binary classification. He we use the Bernulli distribution to model the classification intead of the sigmoid function as in the SGD implementation. In practise the results are usually very similar.

```
y_pred = fm.fit_predict(X_train, y_train, X_test)
y_pred_proba = fm.fit_predict_proba(X_train, y_train, X_test)
```

```
from sklearn.metrics import accuracy_score, roc_auc_score
'acc:', accuracy_score(y_test, y_pred)
'auc:', roc_auc_score(y_test, y_pred_proba)
```

CHAPTER

TWO

GUIDE

2.1 How to choose the right Solver.

This section explains the trade off between the three solvers available in fastFM. The following applies for both **classification** and **regression** tasks.

```
import fastFM.mcmc
```

- (+) least number of hyper parameters
- (+) automatic regularization
- (-) predictions need to be calculated at training time

Note: The predict method of the mcmc model returns predictions based on only the last draw of the model parameters. This evaluation is fast but usually of low quality. Don't use mcmc if you need fast predictions!

```
import fastFM.als
```

- (+) fast predictions
- (+) less hyper parameter then SGD
- (-) regularization must be specified

```
import fastFM.sgd
```

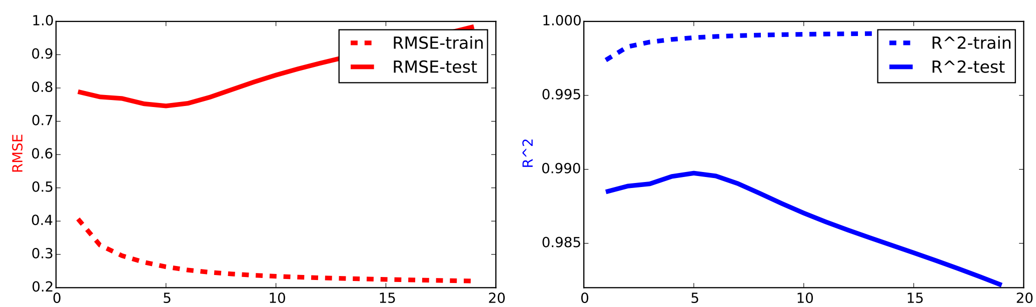
- (+) fast predictions
- (+) warm start can be used to iterate junks of a large dataset
- (-) regularization must be specified
- (-) highest number of hyper parameter (requires, *step_size*)

2.2 Learning Curves

Learning curves are an important tool to understand the model behaviour and alows technics such as early stopping to avoid overfitting. You can use the *warm_start* option with every fastFM model to calculate statistics during during the model fitting process. The following example uses *RMSE* and *R^2* to demonstrate how we can monitore model performance on train and test set efficiently for any metric we want.

```
from fastFM import als
from fastFM.datasets import make_user_item_regression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
```

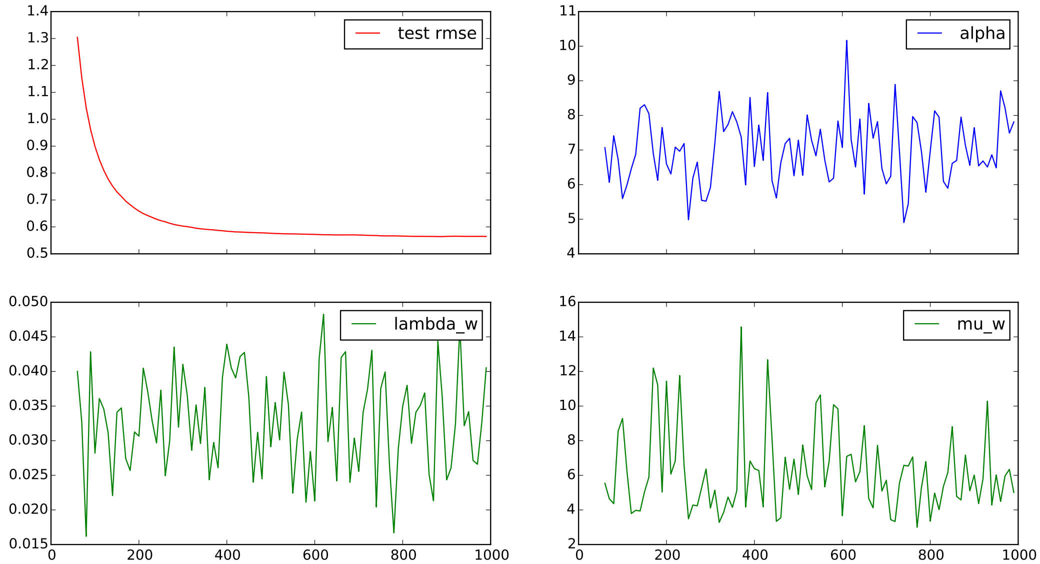
```
X, y, coef = make_user_item_regression(label_stdev=.4)
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)

n_iter = 20
step_size = 1
l2_reg_w = 0
l2_reg_V = 0

fm = als.FMRegression(n_iter=0, l2_reg_w=0, l2_reg_V=0, rank=4)
# Allocates and initalizes the model parameter.
fm.fit(X_train, y_train)

rmse_train = []
rmse_test = []
r2_score_train = []
r2_score_test = []

for i in range(1, n_iter):
    fm.fit(X_train, y_train, n_more_iter=step_size)
    y_pred = fm.predict(X_test)

    rmse_train.append(np.sqrt(mean_squared_error(fm.predict(X_train), y_train)))
    rmse_test.append(np.sqrt(mean_squared_error(fm.predict(X_test), y_test)))

    r2_score_train.append(r2_score(fm.predict(X_train), y_train))
    r2_score_test.append(r2_score(fm.predict(X_test), y_test))


from matplotlib import pyplot as plt
fig, axes = plt.subplots(ncols=2, figsize=(15, 4))

x = np.arange(1, n_iter) * step_size
with plt.style.context('fivethirtyeight'):
    axes[0].plot(x, rmse_train, label='RMSE-train', color='r', ls="--")
    axes[0].plot(x, rmse_test, label='RMSE-test', color='r')
    axes[1].plot(x, r2_score_train, label='R^2-train', color='b', ls="--")
    axes[1].plot(x, r2_score_test, label='R^2-test', color='b')
axes[0].set_ylabel('RMSE', color='r')
axes[1].set_ylabel('R^2', color='b')
axes[0].legend()
axes[1].legend()
```



2.3 Visualizing MCMC Traces

Our MCMC implementation samples model and hyper parameter and every iteration and calculates a running mean of the predictions. MCMC traces can be used to evaluate the convergence and mixing behaviour of the chains. The follwing example demonstrates how to calculate statistics for predictions, hyper parameter and model parameter efficiently using the *warm_start* option.

```
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.cross_validation import train_test_split

from fastFM.datasets import make_user_item_regression
from fastFM import mcmc

n_iter = 100
step_size = 10
seed = 123
rank = 3

X, y, coef = make_user_item_regression(label_stdev=.4)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33)

fm = mcmc.FMRegression(n_iter=0, rank=rank, random_state=seed)
# Allocates and initalizes the model and hyper parameter.
fm.fit_predict(X_train, y_train, X_test)

rmse_test = []
rmse_new = []
hyper_param = np.zeros((n_iter -1, 3 + 2 * rank), dtype=np.float64)
for nr, i in enumerate(range(1, n_iter)):
    fm.random_state = i * seed
    y_pred = fm.fit_predict(X_train, y_train, X_test, n_more_iter=step_size)
    rmse_test.append(np.sqrt(mean_squared_error(y_pred, y_test)))
    hyper_param[nr, :] = fm.hyper_param_

values = np.arange(1, n_iter)
x = values * step_size
burn_in = 5
x = x[burn_in:]

from matplotlib import pyplot as plt
fig, axes = plt.subplots(nrows=2, ncols=2, sharex=True, figsize=(15, 8))

axes[0, 0].plot(x, rmse_test[burn_in:], label='test rmse', color="r")
axes[0, 0].legend()
axes[0, 1].plot(x, hyper_param[burn_in:,0], label='alpha', color="b")
axes[0, 1].legend()
axes[1, 0].plot(x, hyper_param[burn_in:,1], label='lambda_w', color="g")
axes[1, 0].legend()
axes[1, 1].plot(x, hyper_param[burn_in:,3], label='mu_w', color="g")
axes[1, 1].legend()
```



test rmse
alpha
lambda_w
mu_w
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
11
10
9
8
7
6
5
4
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
16
14
12
10
8
6
4
2
0
200
400
600
800
1000

CHAPTER

THREE

THE FASTFM API REFERENCE

3.1 The MCMC module

class `fastFM.mcmc.`**`FMClassification`**(*n_iter=100*, *init_stdev=0.1*, *rank=8*, *random_state=123*, *copy_X=True*)

Factorization Machine Classification with a MCMC solver.

Parameters

- **`n_iter`** (*int, optional*) – The number of samples for the MCMC sampler, number or iterations over the training set for ALS and number of steps for SGD.
- **`init_stdev`** (*float, optional*) – Sets the stdev for the initialization of the parameter
- **`random_state`** (*int, optional*) – The seed of the pseudo random number generator that initializes the parameters and mcmc chain.
- **`rank`** (*int*) – The rank of the factorization used for the second order interactions.

`w0_`

float

bias term

`w_`

float | array, shape = (n_features)

Coefficients for linear combination.

`V_`

float | array, shape = (rank_pair, n_features)

Coefficients of second order factor matrix.

`fit_predict`(*X_train*, *y_train*, *X_test*)

Return average class probabilities of posterior estimates of the test samples. Use only with MCMC!

Parameters

- **`X_train`** (*scipy.sparse.csc_matrix, (n_samples, n_features)*) –
- **`y_train`** (*array, shape (n_samples)*) – the targets have to be encodes as {-1, 1}.
- **`X_test`** (*scipy.sparse.csc_matrix, (n_test_samples, n_features)*) –

Returns **y_pred** – Returns predicted class labels.

Return type array, shape (n_test_samples)

`fit_predict_proba`(*X_train*, *y_train*, *X_test*)

Return average class probabilities of posterior estimates of the test samples. Use only with MCMC!

Parameters

- **X_train** (*scipy.sparse.csc_matrix, (n_samples, n_features)*) –
- **y_train** (*array, shape (n_samples)*) – the targets have to be encodes as {-1, 1}.
- **X_test** (*scipy.sparse.csc_matrix, (n_test_samples, n_features)*) –

Returns **y_pred** – Returns probability estimates for the class with lowest classification label.

Return type array, shape (n_test_samples)

class fastFM.mcmc.**FMRegression**(*n_iter=100*, *init_stdev=0.1*, *rank=8*, *random_state=123*, *copy_X=True*)

Factorization Machine Regression with a MCMC solver.

Parameters

- **n_iter** (*int, optional*) – The number of samples for the MCMC sampler, number or iterations over the training set for ALS and number of steps for SGD.
- **init_stdev** (*float, optional*) – Sets the stdev for the initialization of the parameter
- **random_state** (*int, optional*) – The seed of the pseudo random number generator that initializes the parameters and mcmc chain.
- **rank** (*int*) – The rank of the factorization used for the second order interactions.

w0_

float

bias term

w_

float | array, shape = (n_features)

Coefficients for linear combination.

V_

float | array, shape = (rank_pair, n_features)

Coefficients of second order factor matrix.

fit_predict(*X_train*, *y_train*, *X_test*, *n_more_iter=0*)

Return average of posterior estimates of the test samples.

Parameters

- **X_train** (*scipy.sparse.csc_matrix, (n_samples, n_features)*) –
- **y_train** (*array, shape (n_samples)*) –
- **X_test** (*scipy.sparse.csc_matrix, (n_test_samples, n_features)*) –
- **n_more_iter** (*int*) – Number of iterations to continue from the current Coefficients.

Returns **T**

Return type array, shape (n_test_samples)

3.2 The ALS module

class fastFM.als.**FMClassification**(*n_iter=100*, *init_stdev=0.1*, *rank=8*, *random_state=123*, *l2_reg_w=0*, *l2_reg_V=0*, *l2_reg=0*)

Factorization Machine Classification trained with a ALS (coordinate descent) solver.

Parameters

- **n_iter** (*int, optional*) – The number of samples for the MCMC sampler, number or iterations over the training set for ALS and number of steps for SGD.
- **init_stdev** (*float, optional*) – Sets the stdev for the initialization of the parameter
- **random_state** (*int, optional*) – The seed of the pseudo random number generator that initializes the parameters and mcmc chain.
- **rank** (*int*) – The rank of the factorization used for the second order interactions.
- **l2_reg_w** (*float*) – L2 penalty weight for pairwise coefficients.
- **l2_reg_V** (*float*) – L2 penalty weight for linear coefficients.
- **l2_reg** (*float*) – L2 penalty weight for all coefficients (default=0).
- **Attributes** –
- **---------** –
- **w0** (*float*) – bias term
- **w** (*float | array, shape = (n_features)*) – Coefficients for linear combination.
- **V** (*float | array, shape = (rank_pair, n_features)*) – Coefficients of second order factor matrix.

fit (*X_train*, *y_train*)

Fit model with specified loss.

Parameters

- **X** (*scipy.sparse.csc_matrix, (n_samples, n_features)*) –
- **y** (*float | ndarray, shape = (n_samples,)*) – the targets have to be encodes as {-1, 1}.

class fastFM.als.**FMRegression** (*n_iter=100*, *init_stdev=0.1*, *rank=8*, *random_state=123*, *l2_reg_w=0*, *l2_reg_V=0*, *l2_reg=0*)

Factorization Machine Regression trained with a als (coordinate descent) solver.

Parameters

- **n_iter** (*int, optional*) – The number of samples for the MCMC sampler, number or iterations over the training set for ALS and number of steps for SGD.
- **init_stdev** (*float, optional*) – Sets the stdev for the initialization of the parameter
- **random_state** (*int, optional*) – The seed of the pseudo random number generator that initializes the parameters and mcmc chain.
- **rank** (*int*) – The rank of the factorization used for the second order interactions.
- **l2_reg_w** (*float*) – L2 penalty weight for pairwise coefficients.
- **l2_reg_V** (*float*) – L2 penalty weight for linear coefficients.
- **l2_reg** (*float*) – L2 penalty weight for all coefficients (default=0).
- **Attributes** –
- **---------** –
- **w0** (*float*) – bias term
- **w** (*float | array, shape = (n_features)*) – Coefficients for linear combination.

- **V** (*float | array, shape = (rank_pair, n_features)*) – Coefficients of second order factor matrix.

fit (*X_train*, *y_train*, *n_more_iter=0*)
Fit model with specified loss.

Parameters

- **X** (*scipy.sparse.csc_matrix, (n_samples, n_features)*) –
- **y** (*float | ndarray, shape = (n_samples,)*) –
- **n_more_iter** (*int*) – Number of iterations to continue from the current Coefficients.

3.3 The SGD module

class fastFM.sgd.**FMClassification** (*n_iter=100*, *init_stdev=0.1*, *rank=8*, *random_state=123*, *l2_reg_w=0*, *l2_reg_V=0*, *l2_reg=0*, *step_size=0.1*)
Factorization Machine Classification trained with a stochastic gradient descent solver.

Parameters

- **n_iter** (*int, optional*) – The number of interations of individual samples .
- **init_std** (*float, optional*) – Sets the stdev for the initialization of the parameter
- **random_state** (*int, optional*) – The seed of the pseudo random number generator that initializes the parameters and mcmc chain.
- **rank** (*int*) – The rank of the factorization used for the second order interactions.
- **l2_reg_w** (*float*) – L2 penalty weight for pairwise coefficients.
- **l2_reg_V** (*float*) – L2 penalty weight for linear coefficients.
- **l2_reg** (*float*) – L2 penalty weight for all coefficients (default=0).
- **step_size** (*float*) – Stepsize for the SGD solver, the solver uses a fixed step size and might require a tunning of the number of iterations *n_iter*.
- **Attributes** –
- **----------** –
- **w0** (*float*) – bias term
- **w** (*float | array, shape = (n_features)*) – Coefficients for linear combination.
- **V** (*float | array, shape = (rank_pair, n_features)*) – Coefficients of second order factor matrix.

fit (*X*, *y*)
Fit model with specified loss.

Parameters

- **X** (*scipy.sparse.csc_matrix, (n_samples, n_features)*) –
- **y** (*float | ndarray, shape = (n_samples,)*) – the targets have to be encodes as {-1, 1}.

class fastFM.sgd.**FMRegression** (*n_iter=100*, *init_stdev=0.1*, *rank=8*, *random_state=123*, *l2_reg_w=0*, *l2_reg_V=0*, *l2_reg=0*, *step_size=0.1*)
Factorization Machine Regression trained with a stochastic gradient descent solver.

Parameters

- **n_iter** (*int, optional*) – The number of interations of individual samples .
- **init_stdev** (*float, optional*) – Sets the stdev for the initialization of the parameter
- **random_state** (*int, optional*) – The seed of the pseudo random number generator that initializes the parameters and mcmc chain.
- **rank** (*int*) – The rank of the factorization used for the second order interactions.
- **l2_reg_w** (*float*) – L2 penalty weight for pairwise coefficients.
- **l2_reg_V** (*float*) – L2 penalty weight for linear coefficients.
- **l2_reg** (*float*) – L2 penalty weight for all coefficients (default=0).
- **step_size** (*float*) – Stepsize for the SGD solver, the solver uses a fixed step size and might require a tunning of the number of iterations *n_iter*.
- **Attributes** –
- **----------** –
- **w0** (*float*) – bias term
- **w** (*float | array, shape = (n_features)*) – Coefficients for linear combination.
- **V** (*float | array, shape = (rank_pair, n_features)*) – Coefficients of second order factor matrix.

fit (*X, y*)
Fit model with specified loss.

Parameters

- **X** (*scipy.sparse.csc_matrix, (n_samples, n_features)*) –
- **y** (*float | ndarray, shape = (n_samples,)*) –

3.4 The Ranking module

class fastFM.bpr.**FMRecommender** (*n_iter=100, init_stdev=0.1, rank=8, random_state=123, l2_reg_w=0, l2_reg_V=0, l2_reg=0, step_size=0.1*)
Factorization Machine Recommender with pairwise (BPR) loss solver.

Parameters

- **n_iter** (*int, optional*) – The number of interations of individual samples .
- **init_stdev** (*float, optional*) – Sets the stdev for the initialization of the parameter
- **random_state** (*int, optional*) – The seed of the pseudo random number generator that initializes the parameters and mcmc chain.
- **rank** (*int*) – The rank of the factorization used for the second order interactions.
- **l2_reg_w** (*float*) – L2 penalty weight for pairwise coefficients.
- **l2_reg_V** (*float*) – L2 penalty weight for linear coefficients.
- **l2_reg** (*float*) – L2 penalty weight for all coefficients (default=0).
- **step_size** (*float*) – Stepsize for the SGD solver, the solver uses a fixed step size and might require a tunning of the number of iterations *n_iter*.
- **Attributes** –

- **----------** –
- **w0** (*float*) – bias term
- **w** (*float | array, shape = (n_features)*) – Coefficients for linear combination.
- **V** (*float | array, shape = (rank_pair, n_features)*) – Coefficients of second order factor matrix.

fit (*X*, *pairs*)

Fit model with specified loss.

Parameters

- **X** (*scipy.sparse.csc_matrix, (n_samples, n_features)*) –
- **y** (*float | ndarray, shape = (n_compares, 2)*) – Each row *i* defines a pair of samples such that the first returns a high value then the second FM(X[i,0]) > FM(X[i, 1]).

[TIST2012] Rendle, Steffen. “Factorization machines with libfm.” ACM Transactions on Intelligent Systems and Technology (TIST) 3.3 (2012): 57.

[SIGIR2011] Rendle, Steffen, et al. “Fast context-aware recommendations with factorization machines.” Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval. ACM, 2011.

PYTHON MODULE INDEX

f

F

V

W