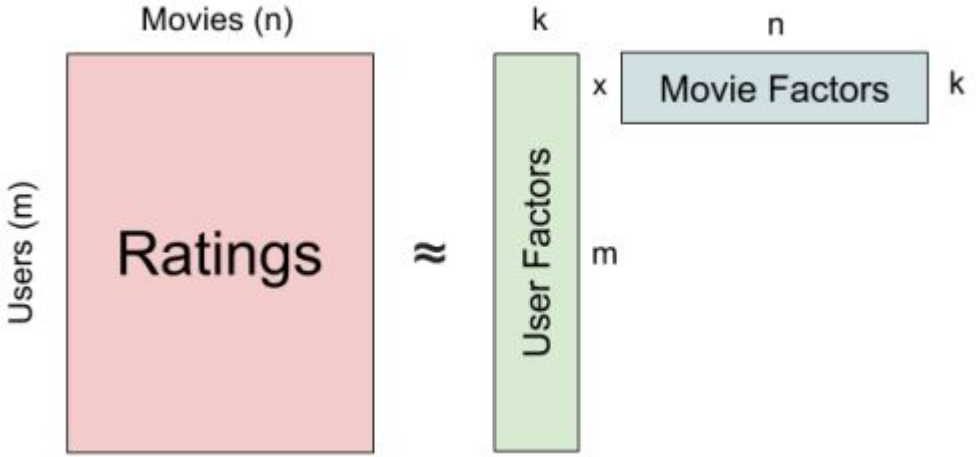
Implementation of ALS Algorithm in Phylanx

Shahrzad Shirzad

Alternating Least Squares ALgorithm

- Recommender system



http://blog2.sigopt.com/post/148703071378/sigopt-for-ml-bayesian-optimization-for

Alternating Least Squares ALgorithm

$$\min_{x_\star, y_\star} \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2)$$

$$y_i = \left(X^T X + \lambda I\right)^{-1} \bar{X}^T r_i$$

$$x_u = \left(Y^T Y + \lambda I\right)^{-1} Y^T r_u$$

- Start with random X, Y
- Find X that minimizes the loss function
- Find Y that minimizes the loss function
- Repeat until desired convergence

Required Primitives

- read_csv
- shape
- diag
- identity
- random
- dot
- trans
- row_slice
- column_slice
- set_row
- not_equal

```
import numpy as np
def ALS(ratings, regularization, num_factors, iterations, alpha=40):
    num_users = np.shape(ratings)[0]
    num_items = np.shape(ratings)[1]
    conf = alpha * ratings
    np.random.seed(0)
    X = np.random.rand(num_users, num_factors)
    Y = np.random.rand(num_items, num_factors)
    I_f = np.identity(num_factors)
    I_i = np.identity(num_items)
    I_u = np.identity(num_users)
    for k in range(iterations):
        YtY = np.dot(Y.T, Y)
        XtX = np.dot(X.T, X)
        for u in range(num_users):
            conf_u = conf[u, :]
            c_u = np.diag(conf_u)
            p_u = conf_u.copy()
            p_u[p_u != 0] = 1
            A = YtY + np.dot(np.dot(Y.T, c_u), Y) + regularization * I_f
            b = np.dot(np.dot(Y.T, c_u + I_i), p_u.T)
            X[u, :] = np.dot(np.linalg.inv(A), b)
        for i in range(num_items):
            conf_i = conf[:, i]
            c_i = np.diag(conf_i)
            p_i = conf_i.copy()
            p_i[p_i != 0] = 1
            A = XtX + np.dot(np.dot(X.T, c_i), X) + regularization * I_f
            b = np.dot(np.dot(X.T, c_i + I_u), p_i.T)
            Y[i, :] = np.dot(np.linalg.inv(A), b)
    return X, Y
```

PhySL vs Python code

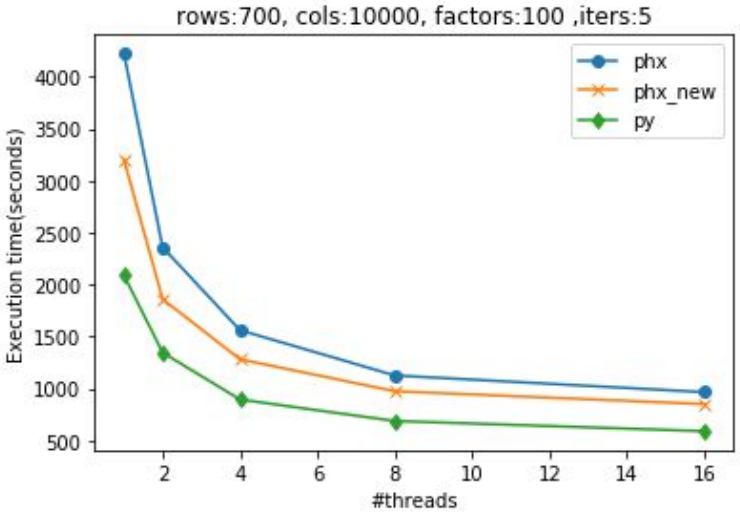
```
define(als, ratings, regularization, num_factors, iterations, alpha,
enable_output,
  block(
    define(num_users, shape(ratings, 0)),
    define(num_items, shape(ratings, 1)),
    define(conf, alpha * ratings),
    set_seed(0),
    define(X, random(make_list(num_users, num_factors))),
    define(Y, random(make_list(num_items, num_factors))),
    while(k < iterations,
      block(
        store(YtY, dot(transpose(Y), Y) + regularization*I_f),
        store(XtX, dot(transpose(X), X) + regularization*I_f),

        while(u < num_users,
          block(
            store(conf_u, slice_row(conf, u)),
            store(c_u, diag(conf_u)),
            store(p_u, __ne(conf_u,0.0,true)),
            store(A, dot(dot(transpose(Y),c_u),Y)+ YtY),
            store(b, dot(dot(transpose(Y),(c_u + I_i)),transpose(p_u))),
            set_row(X,u,u+1,1,dot(inverse(A),b)),
            store(u, u+1)
          ) ),
        store(u, 0),
        while(i < num_items,
          block(
            store(conf_i, slice_column(conf, i)),
            store(c_i, diag(conf_i)),
            store(p_i, __ne(conf_i,0.0,true)),
            store(A, dot(dot(transpose(X),c_i),X) + XtX),
            store(b, dot(dot(transpose(X),(c_i + I_u)),transpose(p_i))),
            set_row(Y,i,i+1,1,dot(inverse(A),b)),
            store(i, i+1)
          ) ),
        store(i, 0),
        store(k,k+1)
      ) ),
    make_list(X, Y)
  )
```

```
import numpy as np
def ALS(ratings, regularization, num_factors, iterations, alpha=40):
    num_users = np.shape(ratings)[0]
    num_items = np.shape(ratings)[1]
    conf = alpha * ratings
    np.random.seed(0)
    X = np.random.rand(num_users, num_factors)
    Y = np.random.rand(num_items, num_factors)
    I_f = np.identity(num_factors)
    I_i = np.identity(num_items)
    I_u = np.identity(num_users)
    for k in range(iterations):
        YtY = np.dot(Y.T, Y)
        XtX = np.dot(X.T, X)
        for u in range(num_users):
            conf_u = conf[u, :]
            c_u = np.diag(conf_u)
            p_u = conf_u.copy()
            p_u[p_u != 0] = 1
            A = YtY + np.dot(np.dot(Y.T, c_u), Y) + regularization * I_f
            b = np.dot(np.dot(Y.T, c_u + I_i), p_u.T)
            X[u, :] = np.dot(np.linalg.inv(A), b)
        for i in range(num_items):
            conf_i = conf[:, i]
            c_i = np.diag(conf_i)
            p_i = conf_i.copy()
            p_i[p_i != 0] = 1
            A = XtX + np.dot(np.dot(X.T, c_i), X) + regularization * I_f
            b = np.dot(np.dot(X.T, c_i + I_u), p_i.T)
            Y[i, :] = np.dot(np.linalg.inv(A), b)
    return X, Y
```

Performance

Dataset:

- MovieLens dataset with 20,000,263 ratings and 465,564 tag applications across 27,278 movies



Future Improvements

- Using solvers
- Adding parallelization
- Using sparse representation
- Using conjugate gradient method

Thank you!

Implicit ALS

- P is a binary matrix
- C is the confidence matrix, usually $c_{ui} = 1 + \alpha r_{ui}$

$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i)$$