The Recommender System Illustration

System Component Description

Feature Encoding of Data Extractor

TABLE III: Developer feature encoding

Features	Description
skills (46)	one-hot encoding of the skills
member age (1)	days the developer becomes a member of Topcoder
technique MD (1)	techniques match degree
language MD (1)	languages match degree
registration (2)	registration frequency and recency
submission (2)	submission frequency and recency
winning (2)	winning frequency and recency
performance (2)	the score and rank in the last challenge
registration rank (1)	the PageRank score in DIG on registration history
submission rank (1)	the PageRank score in DIG on submission history
winning rank (1)	the PageRank score in DIG on winning history

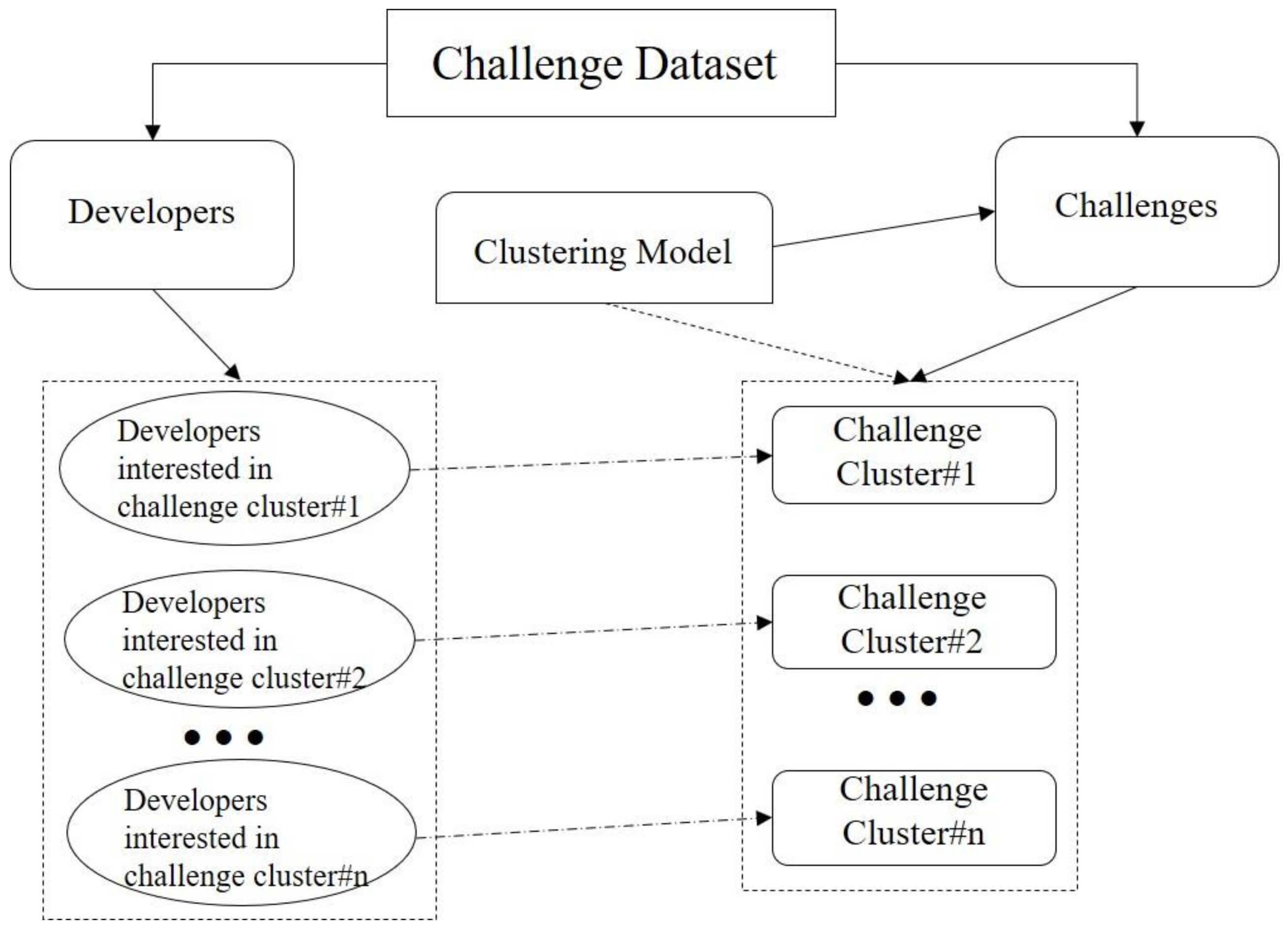
TABLE II: Challenge feature encoding

Features	Description
languages (18)	one-hot encoding of the programming language
techniques (48)	one-hot encoding of the technique used
title (20)	a title vector encoded by Paragraph Vector
requirements (40)	a requirement vector encoded by Paragraph Vector
posting date (1)	the time when the challenge is posted
duration (1)	the number of days the challenge lasts
prizes (1)	the award offered by the customer
difficulty (1)	difficulty of the challenge [25]

We encode the developer and challenge using the features described above.

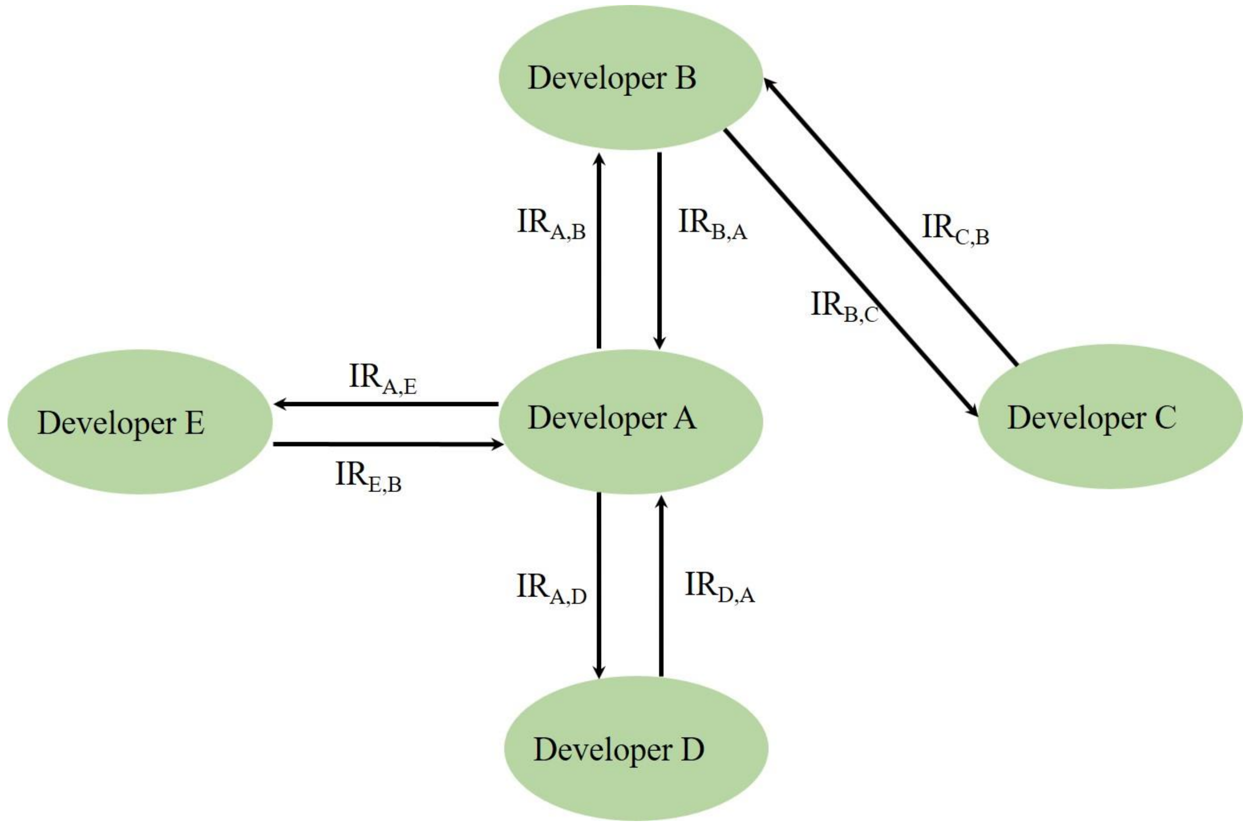
Clustering Component of Data Extractor

This is quite a matured technique and is widely used in recommender. So we just mention clustering method briefly in paper. Here is a detailed description.


Challenge Dataset
Developers
Clustering Model
Challenges
Developers interested in challenge cluster#1
Developers interested in challenge cluster#2
Developers interested in challenge cluster#n
Challenge Cluster#1
Challenge Cluster#2
Challenge Cluster#n

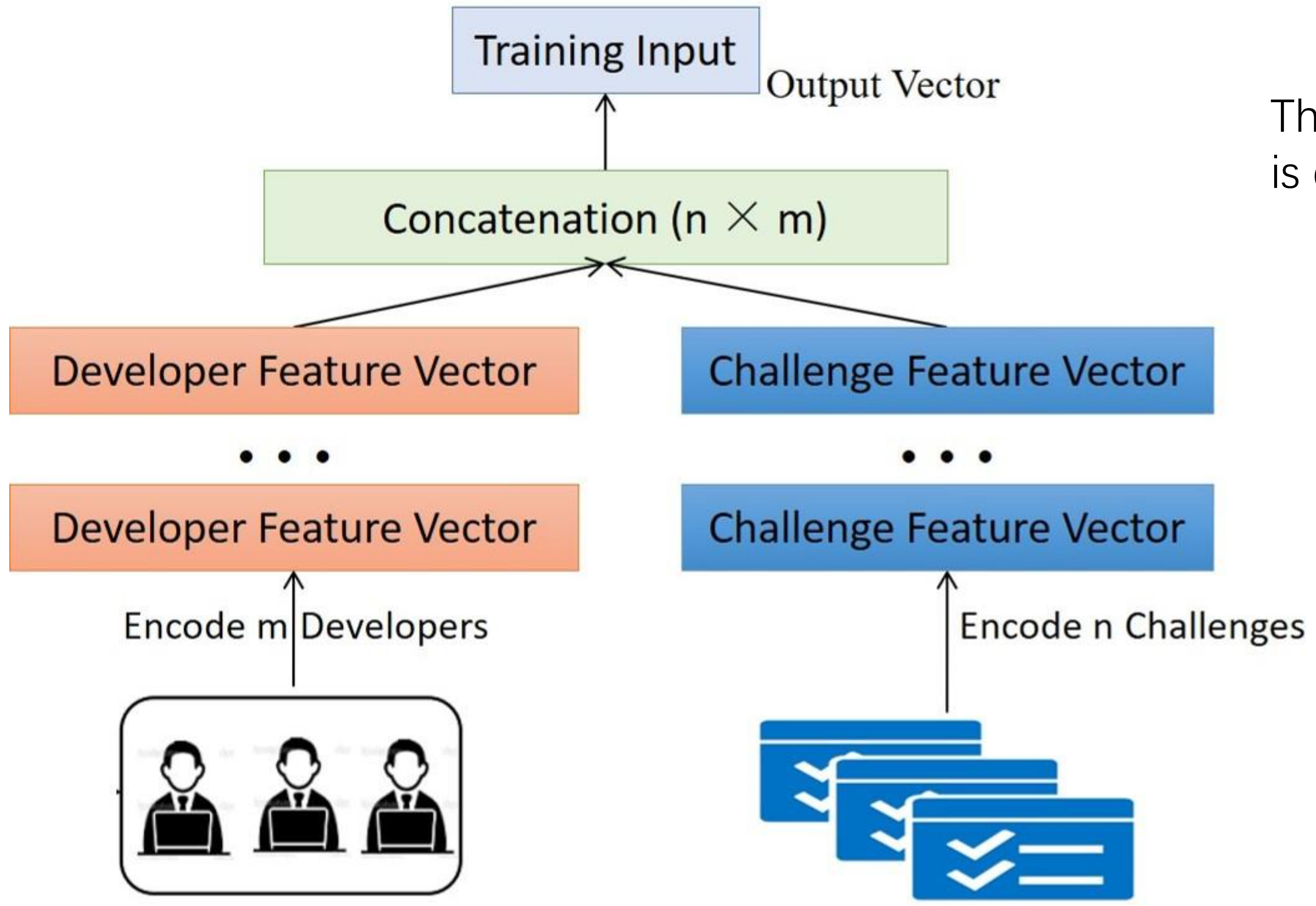
There is a gap between developer’s compacity of winning and interests which means that a developer may be able to win a challenge but he will not contribute works due to lack of interest. The case become very important when the population of developers and count of challenges are large enough. The current approach to solve such problem is to leverage collaborative filtering(CF) context. Previous works in CSD and recommender system design have demonstrated the effectiveness of clustering based CF. In our system, we use the k-means algorithm to build a clustering model to divide such dataset into smaller clusters and build models on them.
When testing a new challenge from the corresponding type of dataset, we use clustering model to predict which cluster it should be assigned to and use policy model built on that cluster to recommend developers.

DIG Component of Data Extractor



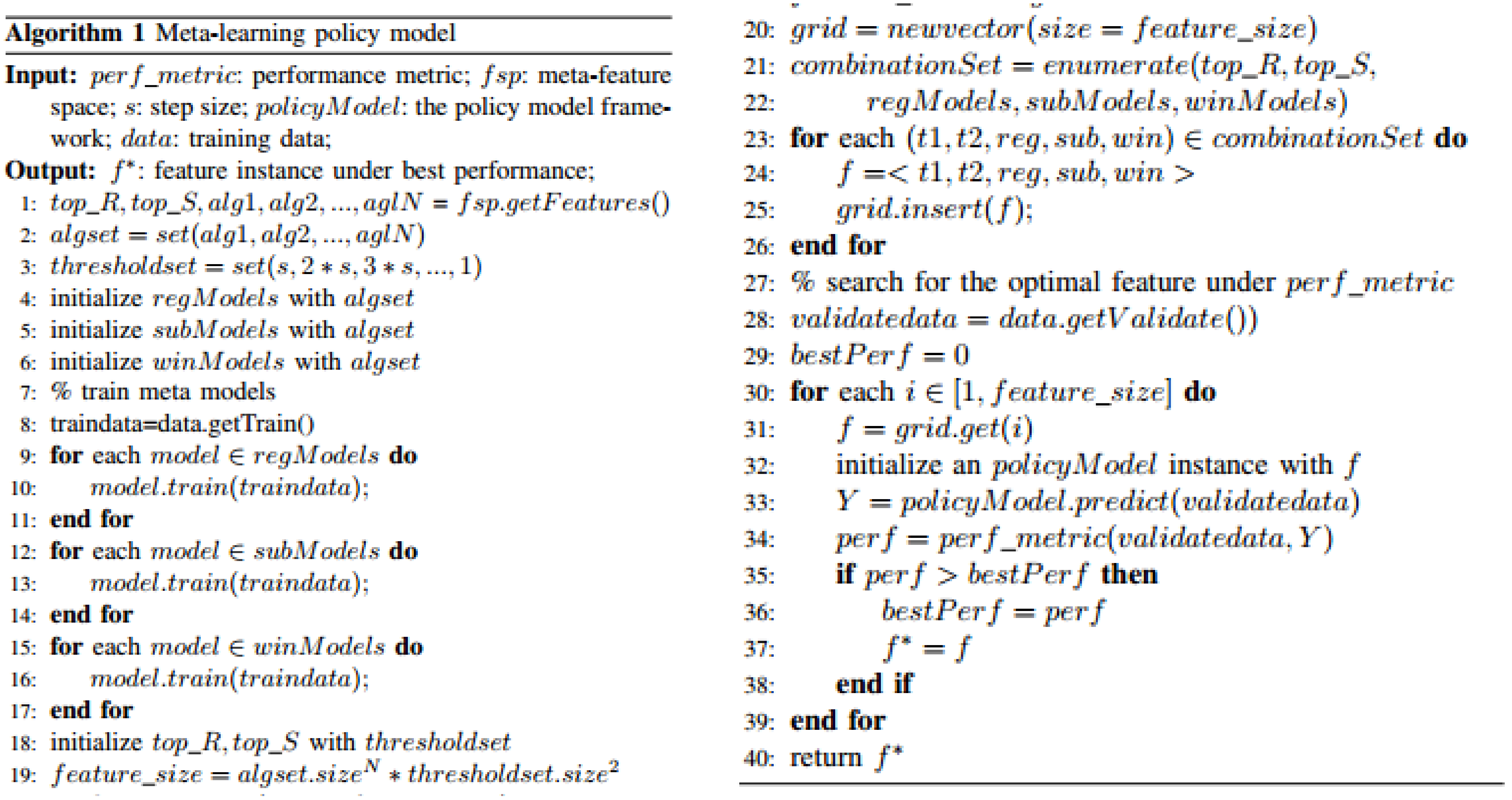
The purpose we build the DIG is that we want to quantify the between developer influence factor. Previous empirical study has shown developers interactive behavior influence. And we here use page rank to extract a rank score that can indicate the influential of others to a developer in a challenge via a directional graph. More details can be found in my paper.

System Input data construction of Data Extractor



This figure illustrates how our system input data is constructed.

Meta learning Algorithm description



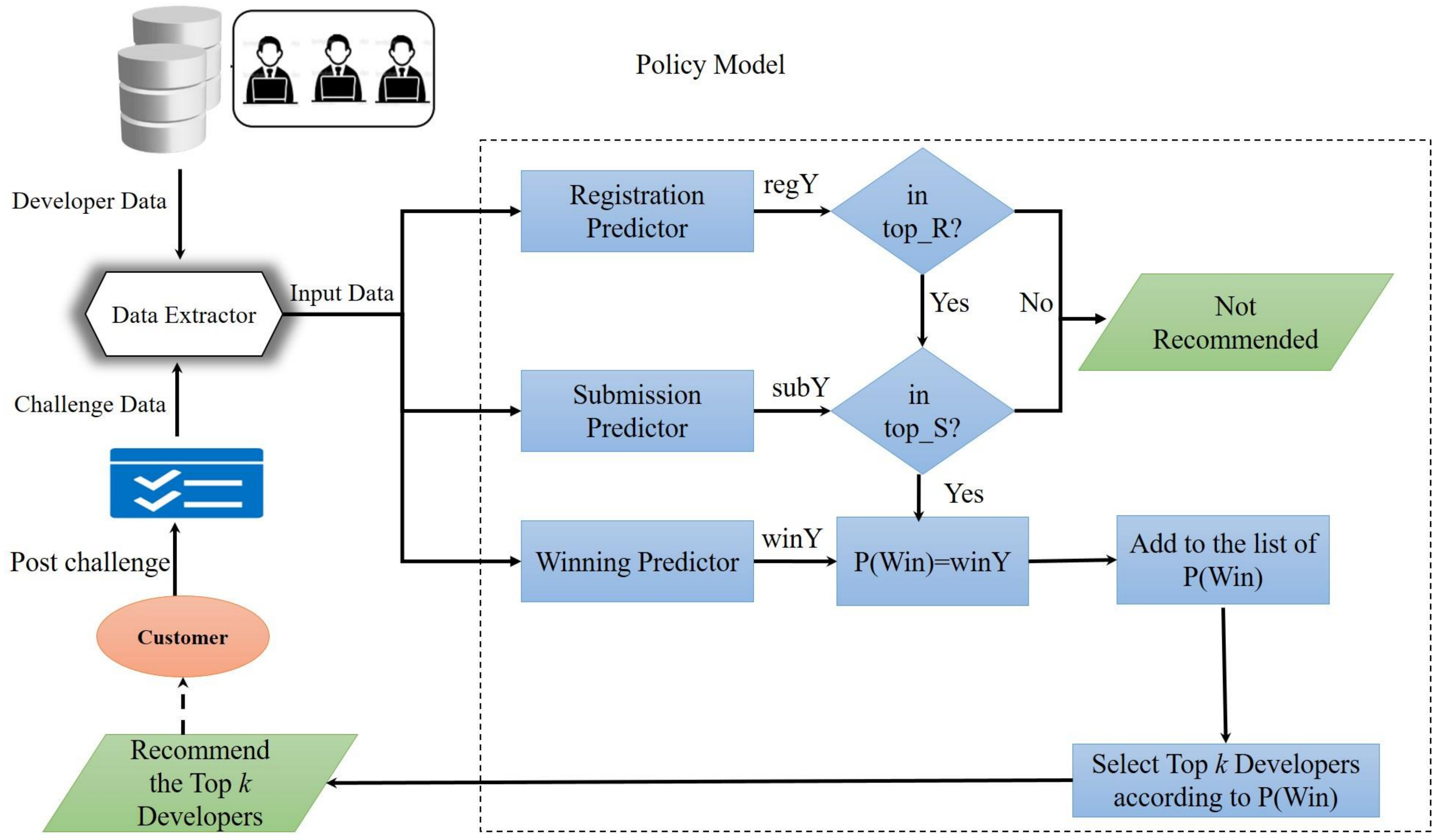
Algorithm 1 Meta-learning policy model

Input: $perf\_metric$: performance metric; fsp: meta-feature space; s: step size; $policyModel$: the policy model framework; $data$: training data;

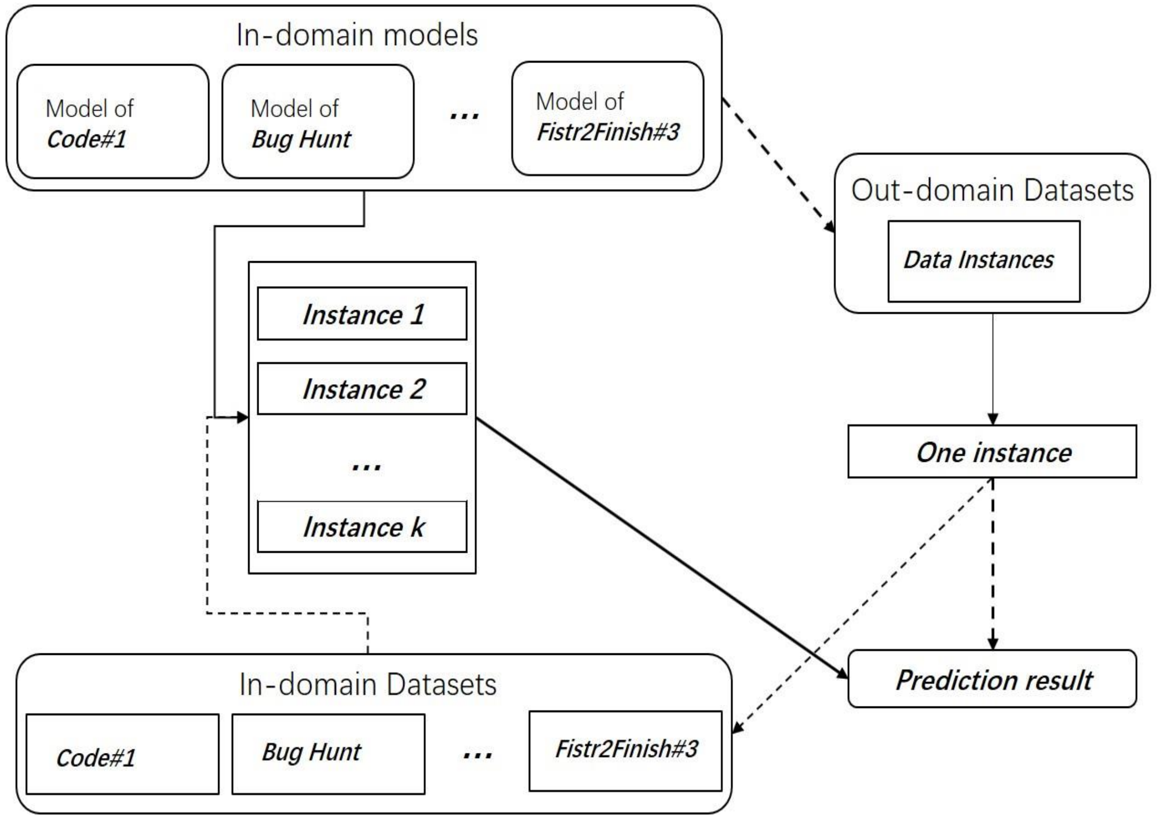
Output: f^*: feature instance under best performance;

```
1:  $top\_R, top\_S, alg1, alg2, ..., aglN = fsp.getFeatures()$
2:  $algset = set(alg1, alg2, ..., aglN)$
3:  $thresholdset = set(s, 2*s, 3*s, ..., 1)$
4:  initialize $regModels$ with $algset$
5:  initialize $subModels$ with $algset$
6:  initialize $winModels$ with $algset$
7:  % train meta models
8:  traindata=data.getTrain()
9:  for each $model \in regModels$ do
10:     $model.train(traindata);$
11: end for
12: for each $model \in subModels$ do
13:     $model.train(traindata);$
14: end for
15: for each $model \in winModels$ do
16:     $model.train(traindata);$
17: end for
18: initialize $top\_R, top\_S$ with $thresholdset$
19: $feature\_size = algset.size^N * thresholdset.size^2$
20: $grid = newvector(size = feature\_size)$
21: $combinationSet = enumerate(top\_R, top\_S,$
22:         $regModels, subModels, winModels)$
23: for each $(t1, t2, reg, sub, win) \in combinationSet$ do
24:     $f =< t1, t2, reg, sub, win >$
25:     $grid.insert(f);$
26: end for
27: % search for the optimal feature under $perf\_metric$
28: $validatedata = data.getValidate())$
29: $bestPerf = 0$
30: for each $i \in [1, feature\_size]$ do
31:     $f = grid.get(i)$
32:     initialize an $policyModel$ instance with $f$
33:     $Y = policyModel.predict(validatedata)$
34:     $perf = perf\_metric(validatedata, Y)$
35:     if $perf > bestPerf$ then
36:         $bestPerf = perf$
37:         $f^* = f$
38:     end if
39: end for
40: return $f^*$
```

Whole System Workflow Illustration



The K-Nearest-Neighbor Transfer Learning



In order to leverage knowledge from existing datasets to help predict winners for a new dataset, we propose a k-nearest neighbor based transfer learning method. For each testing instance in the target dataset, we find the top *k* (empirically we set *k* = 3) closest instances (i.e., the instances that have the highest cosine similarity) in the source dataset. When predicting for the instance in the target dataset, we use a weighted sum of the prediction results from the *k* most similar instances in the source datasets. The weight is computed as the cosine similarity between each data instance in source dataset and the target instance.

Our experiment results demonstrate that our approach is able to support new challenge types, with the help of transfer learning. In practice, we would recommend to use a large-scale existing dataset as the source to predict for a new challenge type.

Source Code Usage Instruction

A List of things you need to keep in mind

- The DIG is implemented in CompetitionGraph Package.
- The machine learning algorithms and policy model are implemented in ML_Models package.
- For challenge and developer feature encoding and some data preprocessing modules of the system, refer to the DataPre package.
- The Utility package contains some personalized tag definition, user function and testing scripts.
- Before running the system, make sure to configure following settings:
  - install python 3.x;
  - pip install keras, tensorflow, scikit-learn, imbalance-learn, numpy. Pymysql, networkx;
  - install mysql database;
  - refer to the topcoder project at: https://github.com/lifeloner/topcoder for newest data crawler implemented in JAVA.
  - customize local mysql database ip and port according to local machine configuration.
- Make sure that the hierarchy of data folder is same in local disk.
- Run DataPre package script to start Data Extractor and generate input data for the system.
- Run ML_Models package script to start meta-model training and optimal meta feature searching.
- Test Policy Model.

Start Cyber Crawler to collect Data

We do have a database in our laboratory, but due to the size and continuously updating of our database, it is not a good way to put the database here. Instead, we put the tools for data collection here, thus everyone can get enough data as they want. If you are eager for our data, contact me via the anonymous email mail@{1196641807@qq.com}.

- Install mysql database into your computer with a linux system, and configure mysql ip and port according to the instruction of https://www.mysql.com/.
- refer to the topcoder project at: https://github.com/lifeloner/topcoder for newest data crawler implemented in JAVA.
- After downloading the java crawler maven project, please use intelliJ idea at: https://www.jetbrains.com/idea/ to deploy the crawler jar package in your machine
- Configure the ip and port of your crawler according to the the configure of mysql database
- Start run the crawler by the following command which will run in background:

```
nohup java –jar crawler.jar &
```

Prepare Your Programming Environment

- We develop the whole system using python, so we recommend you to install an anaconda virtual python3.6 environment at: https://www.anaconda.com/
- We need to connect to mysql database, apply machine learning and deep learning algorithms, use page rank to generate developer rank score, etc. So readers have to install following python packages:
  - Pymysql, scikit-learn, lightgbm, xgboost, numpy, tensorflow, keras, network, imbalance-learn,pandas
- Besides, due to the computation and ram requirement is quite large, we recommend you prepare a machine with following requirement
  - 256G RAM or more
  - At least 12 cores of CPU
  - 1TB$^+$ Disk memory
  - TitanXP NVIDIA GPU is recommended for boosting computation
  - Make sure the bandwidth is at least 1000Mb/s if the mysql database is not in your programming machine

Construct Input Data

- Configure the datra/dbSetup.xml and set ip and port as same as the machine running mysql database, copy data/viewdef.sql and run it in your mysql client to create view for initial data cleaning.
- You need to encode Developer and Challenge features at first
  - Run TaskContent.py of DataPre package to generate challenge feature encoding vectors and build clustering model
  - Run UserHistory.py of DataPre package to generate developer history data
  - Run DIG.py of CompetitionGraph package to generate developer rank score data
- Run TaskUserInstances.py of DataPre package to generate input data
  - Adjust the maxProcessNum of DataInstances class to adapt your computer CPU and RAM
  - For training,set global variant testInst=False. The value of variant mode in global means 0-registration training data input, 1-submission training data input, 2-winning training data input. You have to run the script under the 3 values.
  - Generate test input data via set mode=2 and testinst=True

- After finished running all the above scripts, check whether the generate traing input and test input data is completed via running the TopcoderDataset.py

Train Meta Models

- Run XGBoostModel.py of ML_Models package
  - Feed “keepd” as key of tasktypes and run the script for 3 times with mode =0,1,and 2
  - Feed “clustered” as key of tasktrypes and run the script for 3 times with mode=0,1,and 2
  - After finished this, the meta model implemented using XGBoost algorithms can extract registration meta-feature, submission meta-feature and winning met-feature of all datasets
- Run DNNModel.py of ML_Models package in the same way as XGBoostModel.py
- Run EnsembleModel.py of ML_Models package in the same way as XGBoostModel.py

- Generate the performance of all the winning meta models via running MetaModelTest.py of ML_Models package
  - Readers can build winning predictor based on the performance results

Run Policy Model and Baselines

- Run BaselineModel.py of ML_Models package to build the baseline models we mentioned in the paper
    - After building baseline models, run the MetaModelTest.py of ML_Models package again but pass the model name as the names of classes of the baseline model in BaselineModel.py to generate performance results

- Run PolicyModelTuning.py of ML_Models package to apply meta-learning meta-feature selection process and generate the meta data (meta-feature, performance).
    - Readers can refer to MetaLearning.py of ML_Models package which implemented some new learning process but may not be global optima