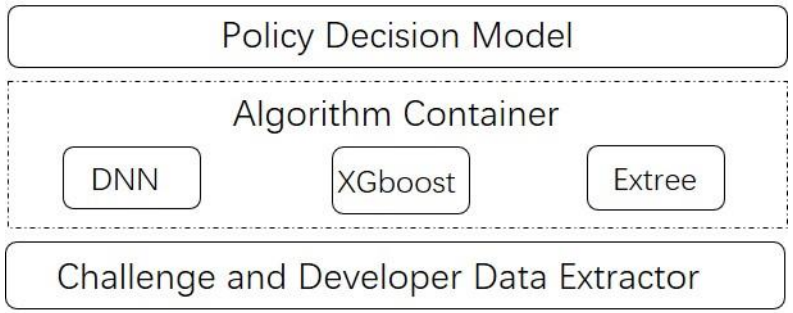
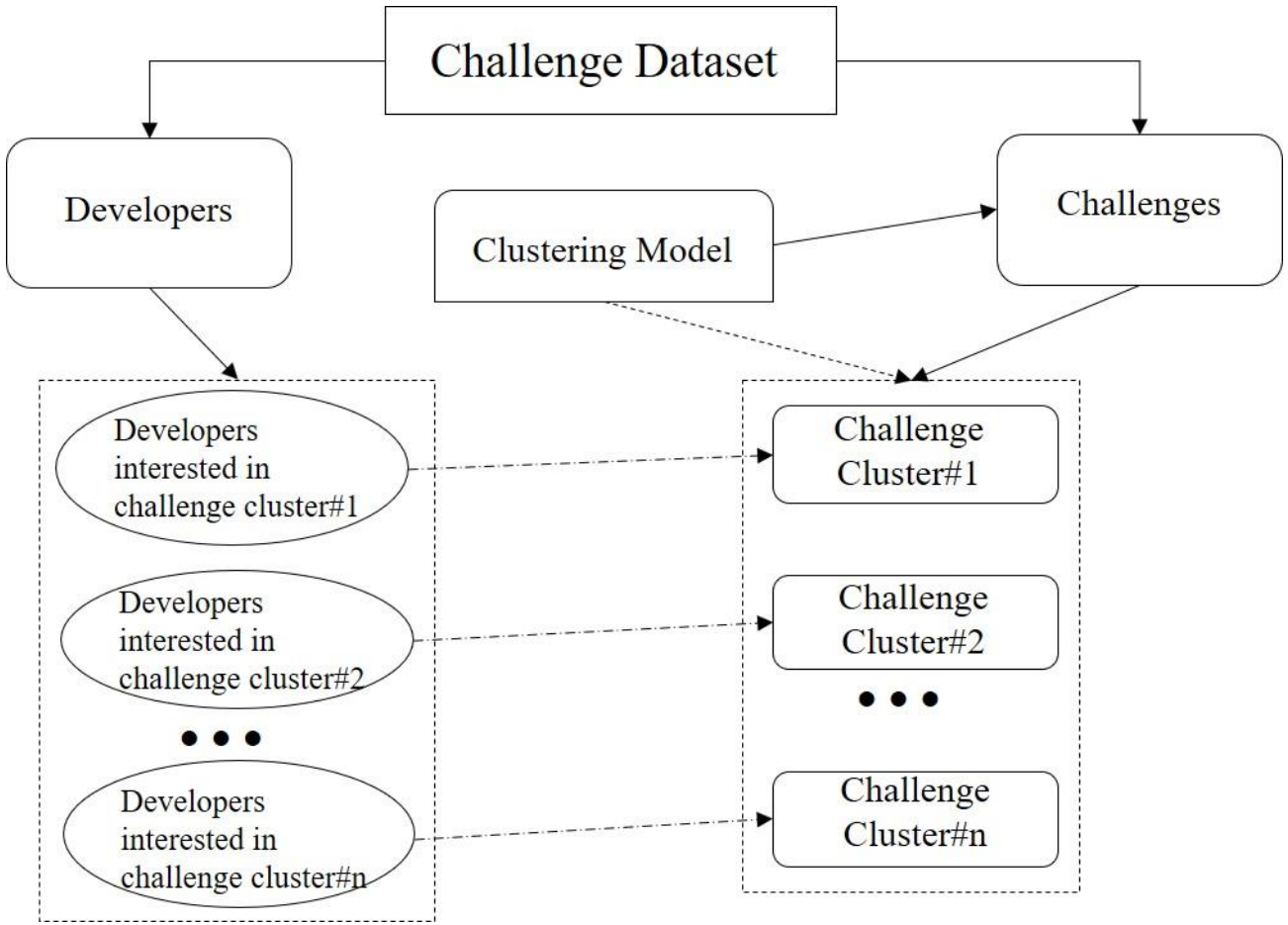
The Recommender System Illustration



System Component Description

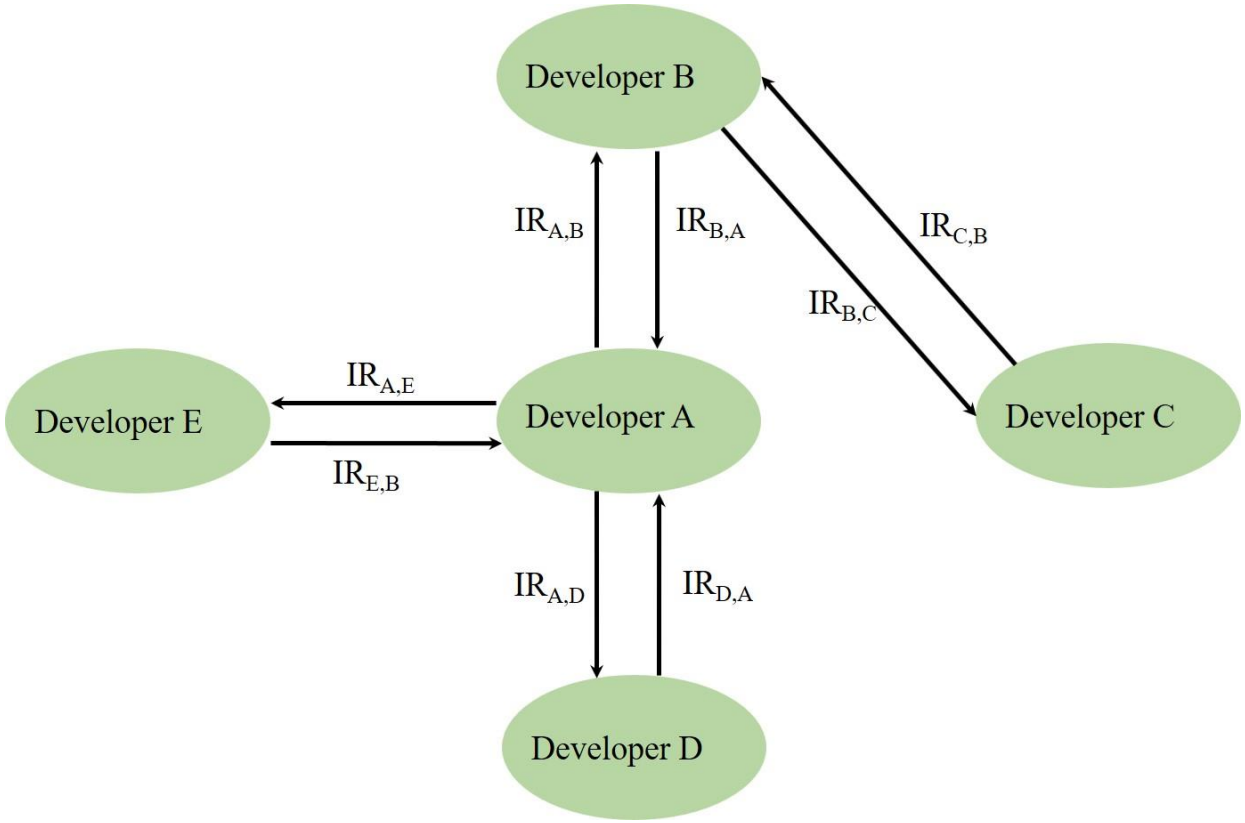
Clustering Dataset Component of Data Extractor



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 developers population and challenges count 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.

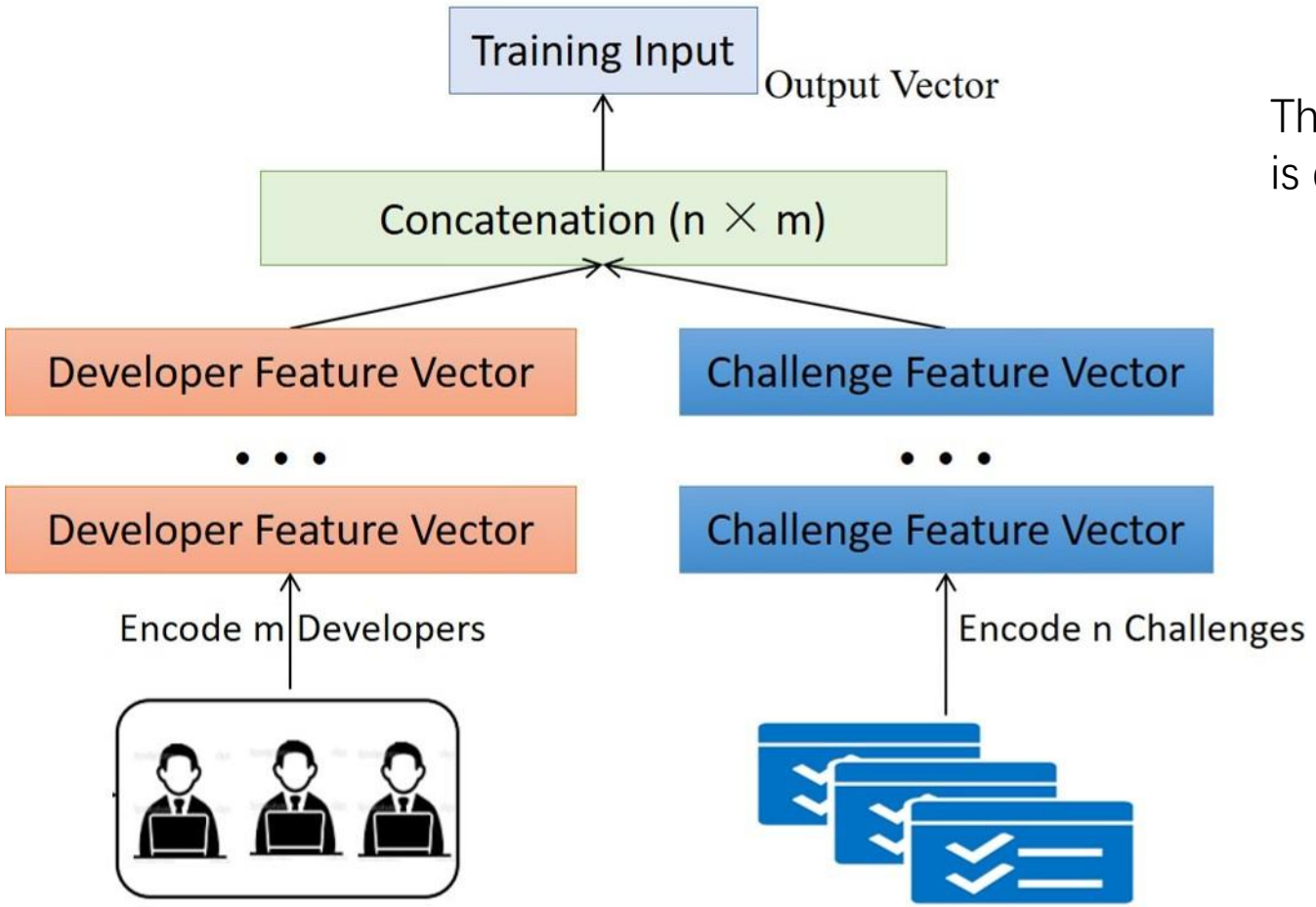
When testing a new challenge from the corresponding type of dataset, we use clustering model to predict which cluster it should be assigned to and use policy model built on that cluster to recommend developers.

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.

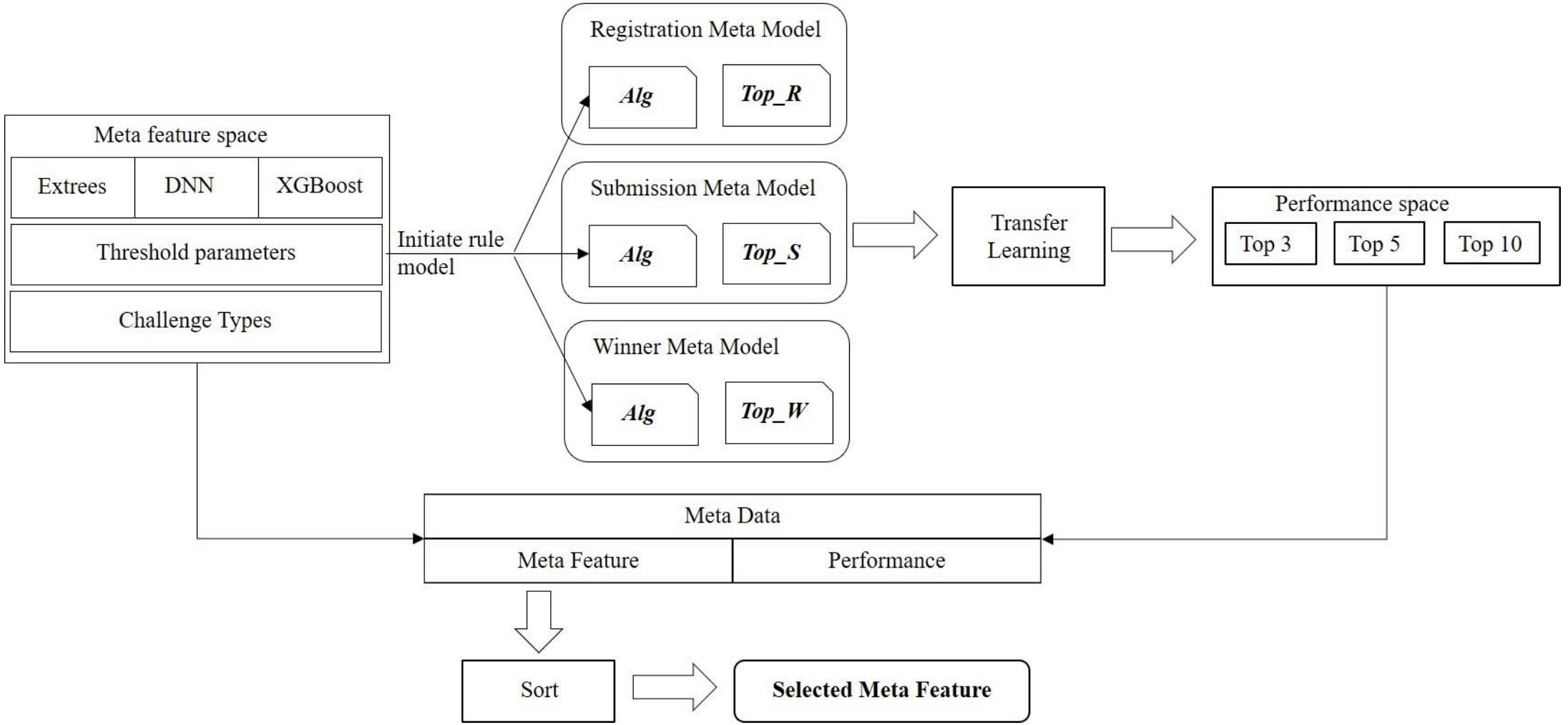
System Input data construction of Data Extractor

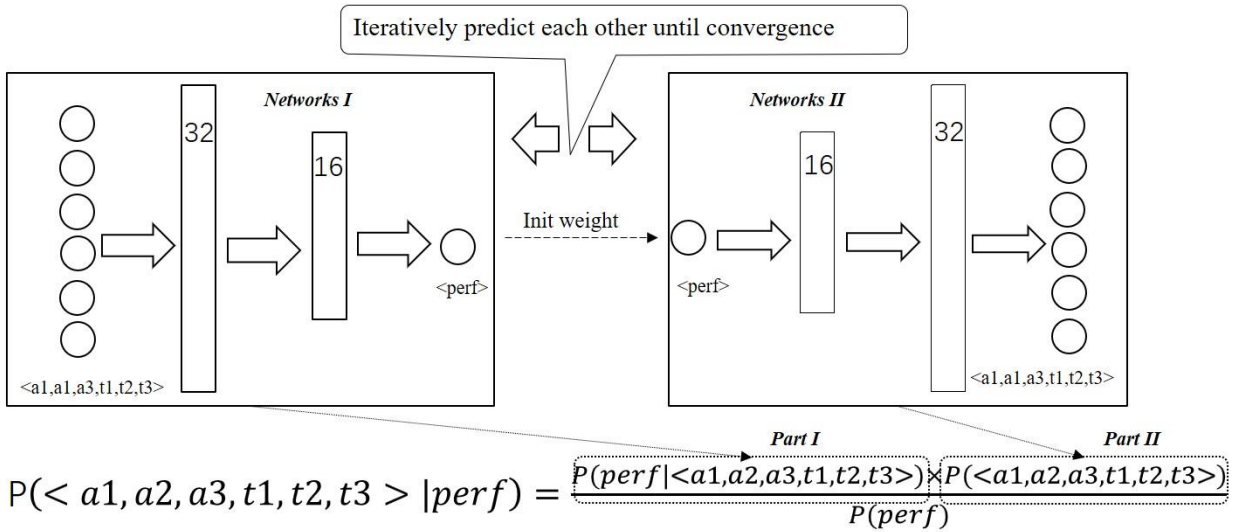


This figure illustrates how our system input data is constructed.

Meta learning System Illustration

More details can be found in my paper.
This is just how our meta-learning system is made up.

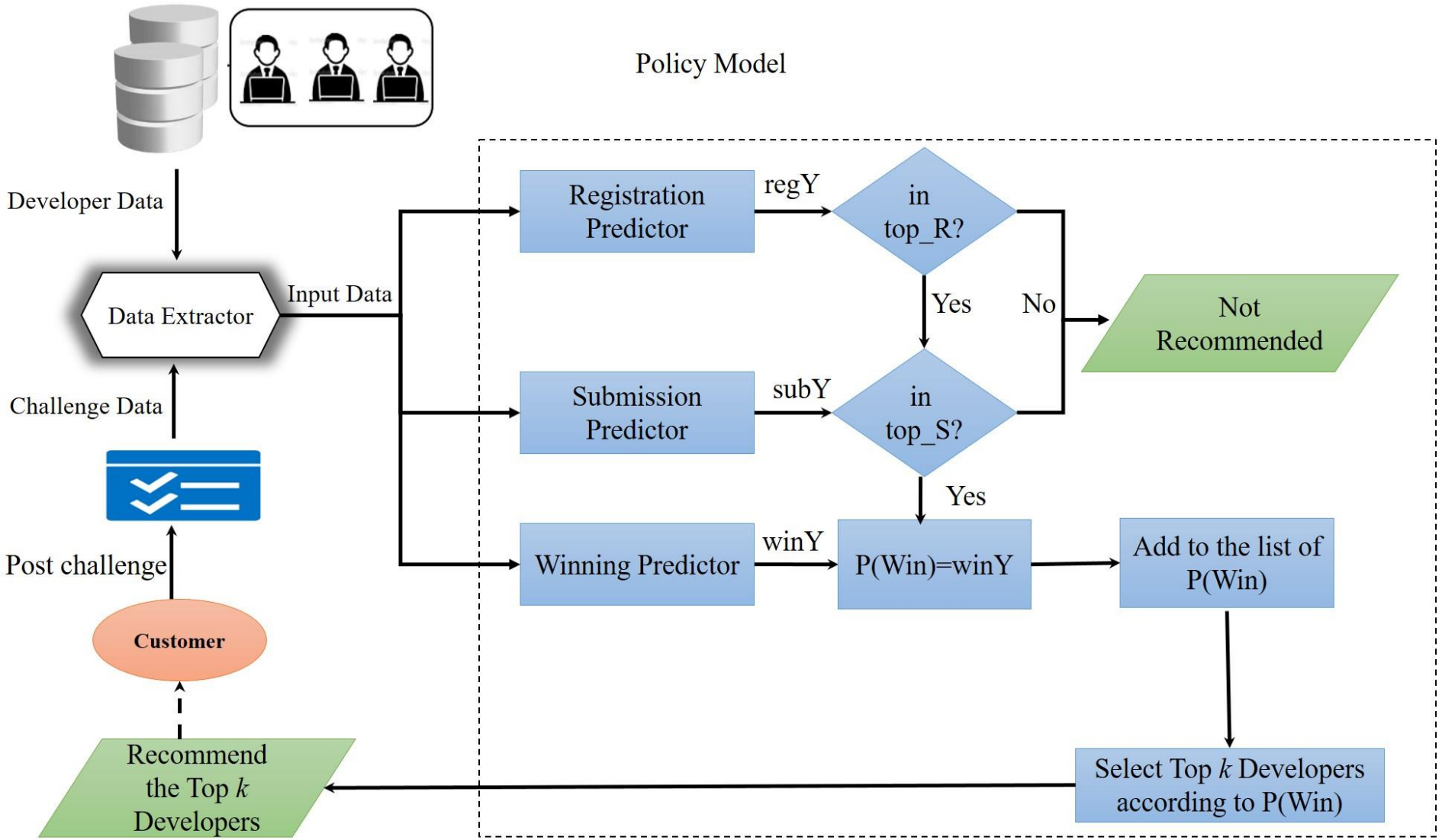




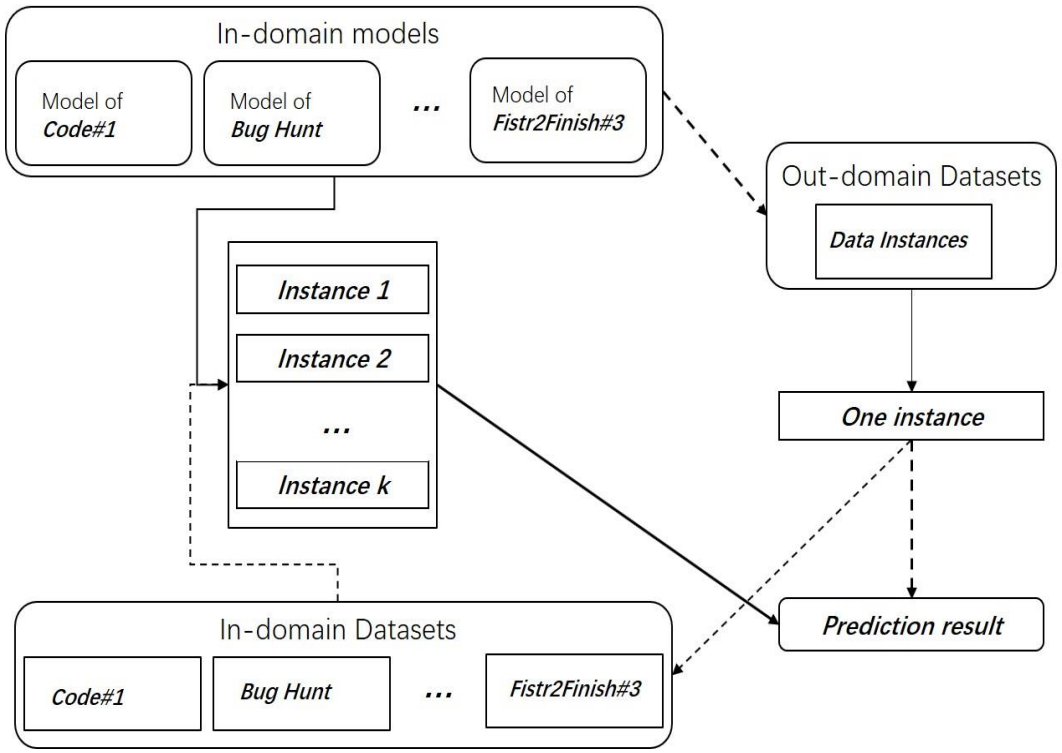
We tried to build the bi-NN to replace the sort and select best process. However, though the networks can result in a satisfied result, but we still cannot demonstrate its convergence and principles. So we abandoned in it in our system.

However, in some cases, we need to consider a much larger algorithm space and more accurate threshold step(e.g. 0.01), which will result in a much huge 6-dimension cube that is impossible to search optima. Thus we need to replace the search based learning process with a bi-DNN bayes learning algorithm we proposed for a general purpose. We design this algorithm according to the navie bayes classifiers proposed by Rennie et al. and strong represenattion of neural networks. we firstly build 2 networks with same hidden structure but with different forward direction and train them using the rough meta data we constructed. For the 3 predictors(registration, submission and winner), we need to find best algorithms ($a1$, $a2$ and $a3$) and 3 best threshold($t1$, $t2$, and $t3$). To predict the performance($perf$) under $<a1,a2,a3,t1,t2,t3>$, we need to learn two part of the bayes equation in Figure \ref{fig-biDNN}. The term $P(perf)$ is constant for a given dataset. However the 6 parameters are not independent of each other, so it is hard to directly decompose them according to some known probability disrtibution. The neural networks are good at simulate complex functions thus we can learn Part I of Figure \ref{fig-biDNN} via networks I and then use the weights of networks I to initiate networks II, which can be viewed as a warm start. Finally networks II can learn Part II. We can feed $perf=1$ to networks II to predict $<a1,a2,a3,t1,t2,t3>$ and then predict $perf$ via networks I using $<a1,a2,a3,t1,t2,t3>$ in return. The convergence condition is that the $|perf_{t}-perf_{t-1}|<0.01$ in tth iteration. After several iterations we will be able to get 6 fine predicted parameters and the final $perf<1$.

Whole System Workflow Illustration



The K-Neighbor Transfer Learning



Reader can refer to my paper for more information about CR value.

In order to leverage collaborative developers from neighbor datasets to help predict winners for current dataset,
we apply a k-neighbor transfer learning approach, which might be useful.
When predicting the results, we can use a weighted sum of prediction results of model from a neighbor dataset.
The weight here is defined by the cosine similarity of the data instances we construct.
For each instance from the out-domain dataset we feed to the transfered model,
we need to find k-nearest instances (empirically we set $k=3$ here) from the in-domain dataset where the base predictors of the transfered model is built.
The cosine similarity is computed using the average similarity value between the feed instance and the k-nearest neighbor instances.
In fact, the transfer learning method is just a trade-off between the consistency of the in-domain dataset and out-domain dataset and the knowledge of the transfered model.
Once the knowledge of the transfered model can outperform the effect caused by the inconsistency of different domains, we will get a lift in performance.
Otherwise, the transfered model will even perform worse than the original model in the out-domain dataset.
The parameter CR we defined according to the empirical study of Topcoder datasets can indicate the developers interest in participation of 2 types of challenges and the high CR indicates that the developers of 2 types challenges are highly overlapped both in entity and interest.
We can infer the out-domain developer behaviors from in-domain developers who are regarded as collaborators of out-domain in such a way.

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