

Team #9

Game Popularity Prediction System

 Jiaao Yu
 Miner Yang
 Runjie Li





Outline

Part1 System introduction & design

Part2 System implementation & test



Part1

- Goals & Data sources
- Popularity definition
- Use cases
- Processes & Methodology
- Acceptance Criteria result






Goals

- Predict popularity of a new released game
- Back the predicted result to our client (game developer and publisher)
- Popularity would be classified into multi-level
- Assist to evaluate steam game market better



Popularity Definition



Label (ratings)	stars	Popularity levels
0		Overwhelmingly Positive (95%-99%)
1		Very Positive (90%-95%)
2		Positive (80%-90%)
3		Mostly Positive (70%-80%)
4		Negative (0%-70%)

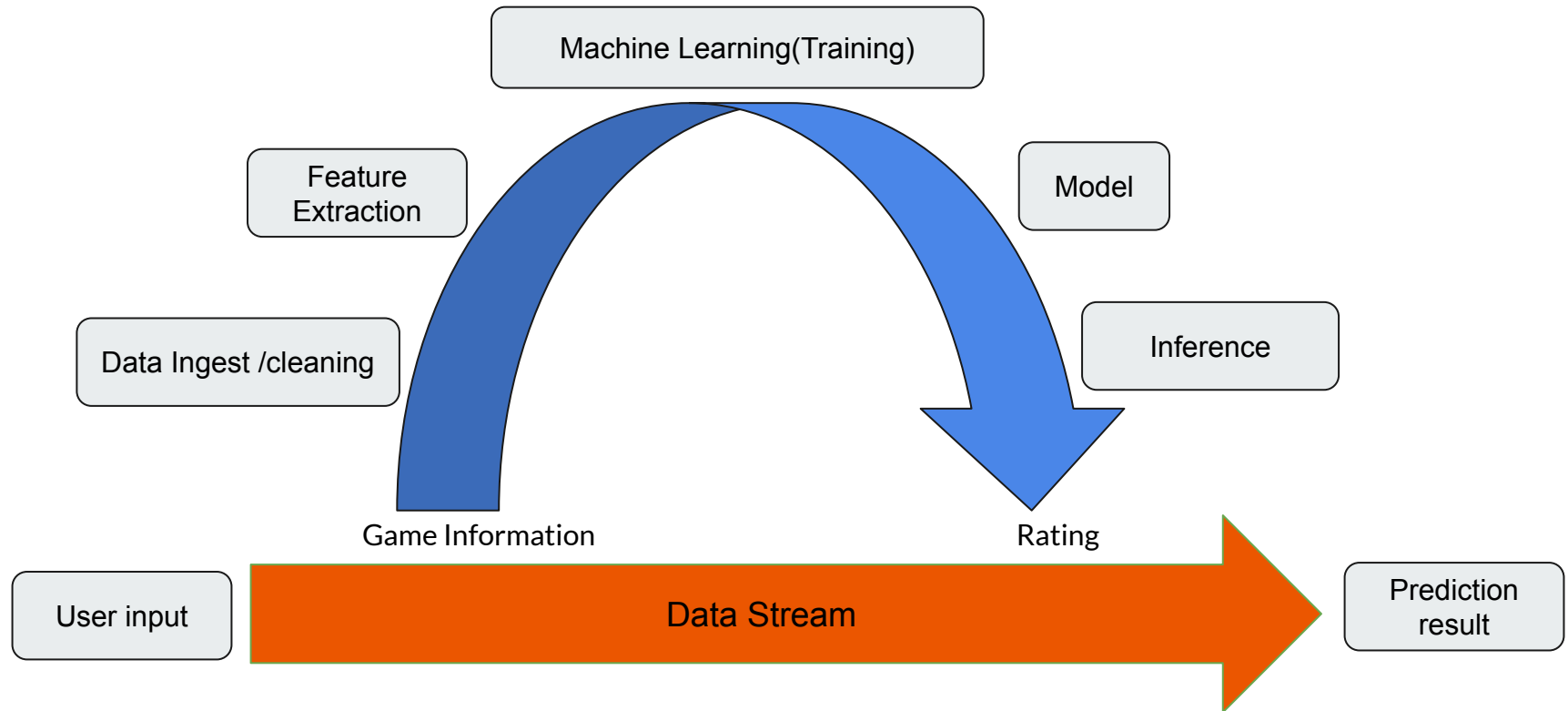


Data

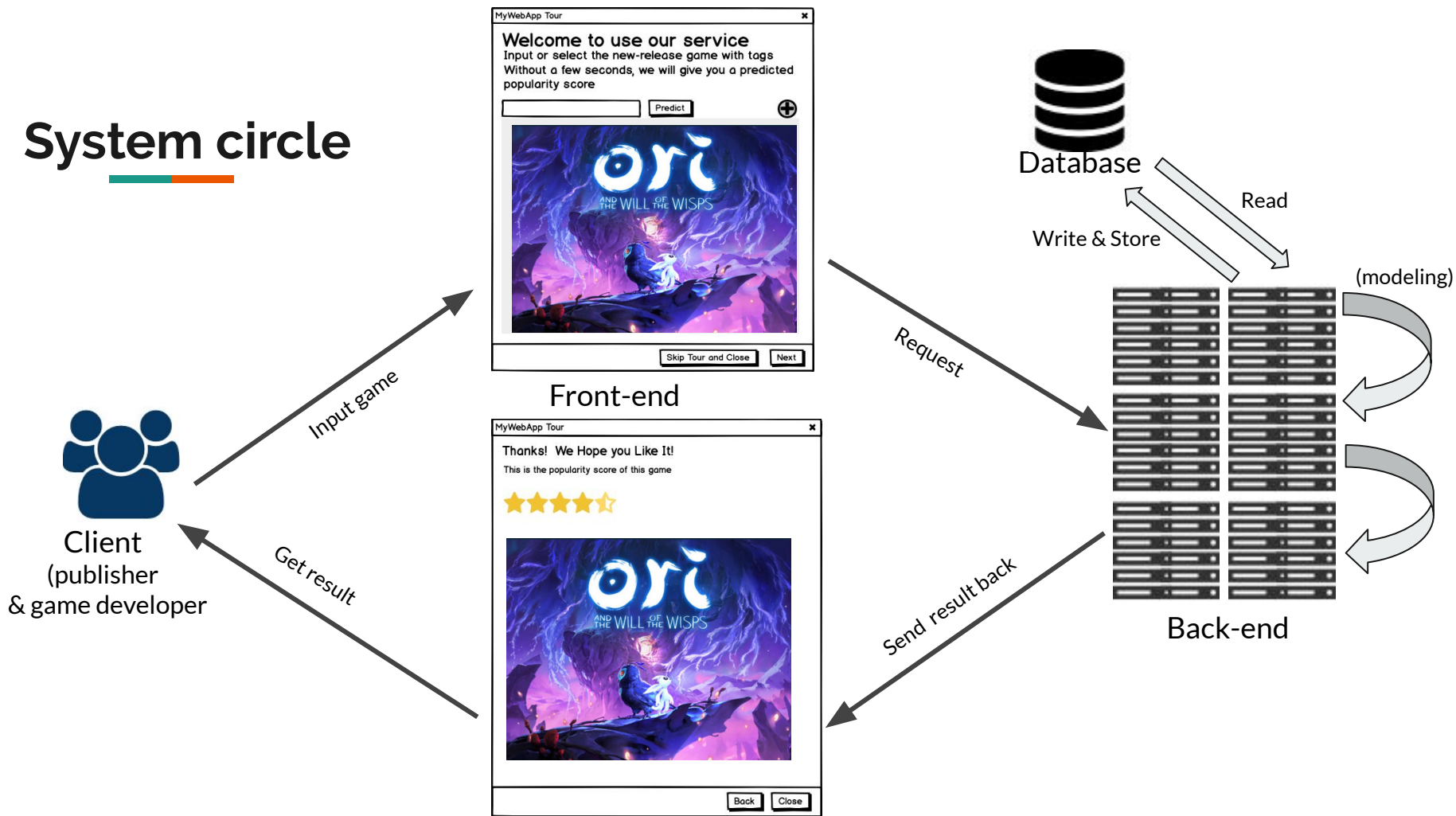
- Main Source [Steam Store Games \(Clean dataset\)](#)
- Combined Source [Steam Reviews Dataset](#)
- There are 27075 game id in the dataset

steam.csv	27075 rows	18 columns
steam_description.csv	27334 rows	4 columns
steam_media_data.csv	27332 rows	5 columns
steam_requirement.csv	27319 rows	6 columns
steam_support_info.csv	27319 rows	4 columns
steamspy_tag_data.csv	29022 rows	372 columns

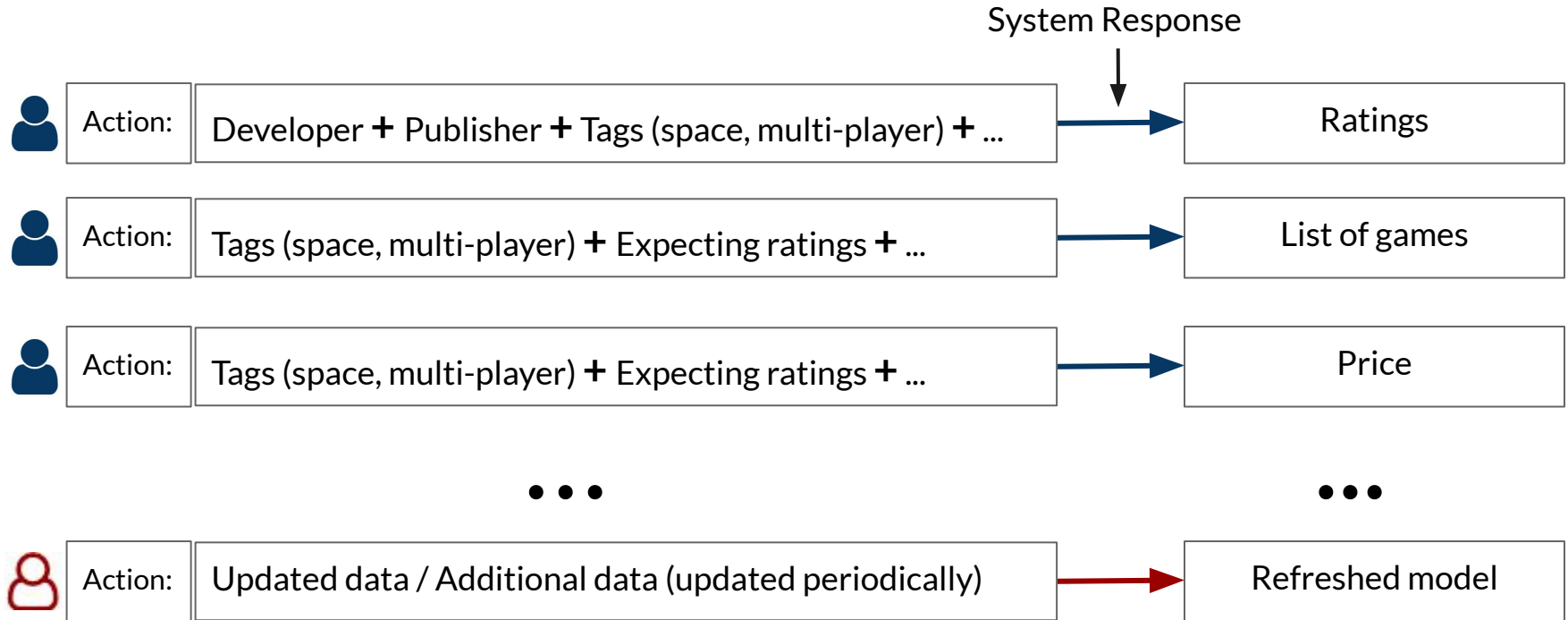
Big Data Cycle



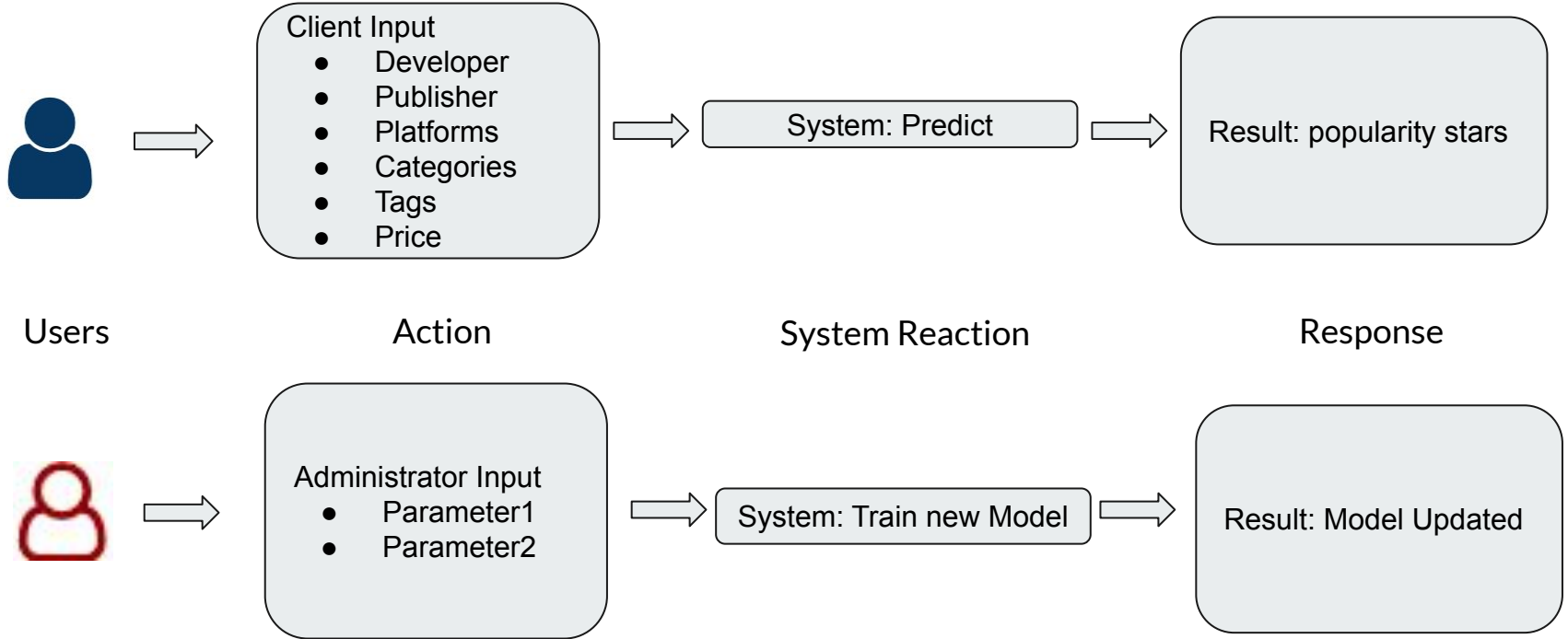
System circle



Use Cases



Use Cases



Methodology



STEPS

- **Data Ingest:** Using Spark Streaming to read from kaggle dataset website.
- **Preprocessing:** Clean and break down the text data using scala and Spark.
- **Feature extraction:** Using multiple transformers and estimators with pipeline to extract features from clean data.
- **Modeling:** Training several models to get a qualified one (LogisticRegression, Naive Bayes, Multilayer Perceptron, RandomForest).
- **Model persistent:** Using Pipeline to persist our model to do real-time prediction.
- **REST Web Application:** Using qualified models to predict, react with users.



Scala Utility

- Data cleaning & preprocess (Streaming, dataframe, spark sql)
- Pipeline setting(spark ml feature selector)
- Feature engineerings(spark pipeline, pipeline model, dataframe)
- Model Training (spark MLlib)
- Model analysis (Multi classification Evaluator)
- Model persistent (spark pipeline)
- Unit Test (scalatest, spark sql)

Other

- Other Machine learning skills(exploring)
- Using Pyspark & flask to build Back-End
- Using node.js to build Front-End
- Axios to handle Asynchronous http request and response



Github Repository

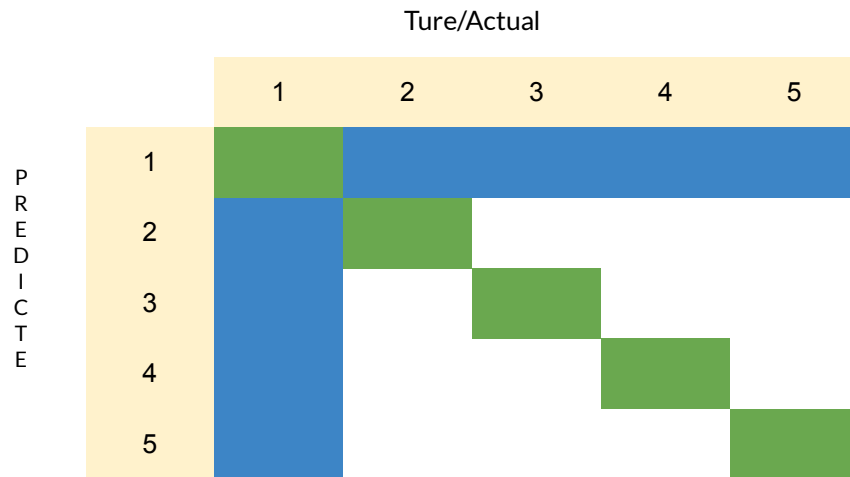
https://github.com/MinerYang/Team09_GamePopularity

Acceptance Criteria



	Precision	Recall	f1-score
LR	73%	76%	70%
MLP	55%	71%	61%
NB	66%	71%	65%
RF	48%	69%	57%

- Response time less than 2 seconds



confusion matrix for multi-class

- Average precision: > 65%
- Average recall: > 65%
- Average f1-score: > 65%
- Response time should less than 4 seconds



Part2

- Project structure
- Labeling methods
- Pipeline & feature selection
- Model training & Analysis
- Test & Real time prediction system



Project structure



Project structure

```
├── README.md
├── RatingModelTraining
│   ├── build.sbt
│   ├── nbymodel1
│   ├── predata2.parquet
│   ├── project
│   ├── selectionFile
│   ├── spark-warehouse
│   ├── src
│   └── target
├── RatingServer2.0
│   └── RatingServer2.0.ipynb
```

```
├── main
│   ├── resources
│   │   └── steam.csv
│   └── scala
│       ├── MachineLearning
│       │   ├── FeatureEngineering.scala
│       │   ├── PipelineTransfomer.scala
│       │   └── Training
│       │       ├── LR.scala
│       │       ├── MLP.scala
│       │       ├── NB.scala
│       │       └── RF.scala
│       ├── app
│       │   ├── DataCleaning.scala
│       │   ├── ML.scala
│       │   ├── ModelExport.scala
│       │   └── UserInput_Prediction.scala
│       └── schema
│           └── GameSchema.scala
└── test
```




Project structure

```
— UnitTest&Test\ sample
  |— Test\ Sample
  |— Test\ Server&Website
  |— UnitTest
```

```
— ratingWebsite2.0
  |— README.md
  |— babel.config.js
  |— jsconfig.json
  |— package.json
  |— quasar.conf.js
  |— src
  |— yarn.lock
```

backend

```
if __name__ == "__main__":
    app.run(debug=True, use_reloader=False)

* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

developer	publisher	platforms	categories	tags	price
[Valve]	[Valve]	[windows, mac, li...]	[Multi-player, On...]	[Action, FPS, Mul...]	7.19

```
127.0.0.1 - - [13/Apr/2020 17:47:58] "POST /predict HTTP/1.1" 200 -
result: 3.0, accuracy: 0.7
begin to train a new model
training completed

127.0.0.1 - - [13/Apr/2020 17:50:37] "POST /train HTTP/1.1" 200 -
new model accuracy: 0.6915876489180647
```

developer	publisher	platforms	categories	tags	price
[Valve]	[Valve]	[windows, mac, li...]	[Multi-player, On...]	[Action, FPS, Mul...]	7.19

```
127.0.0.1 - - [13/Apr/2020 17:51:28] "POST /predict HTTP/1.1" 200 -
result: 4.0, accuracy: 0.6915876489180647
```

frontend

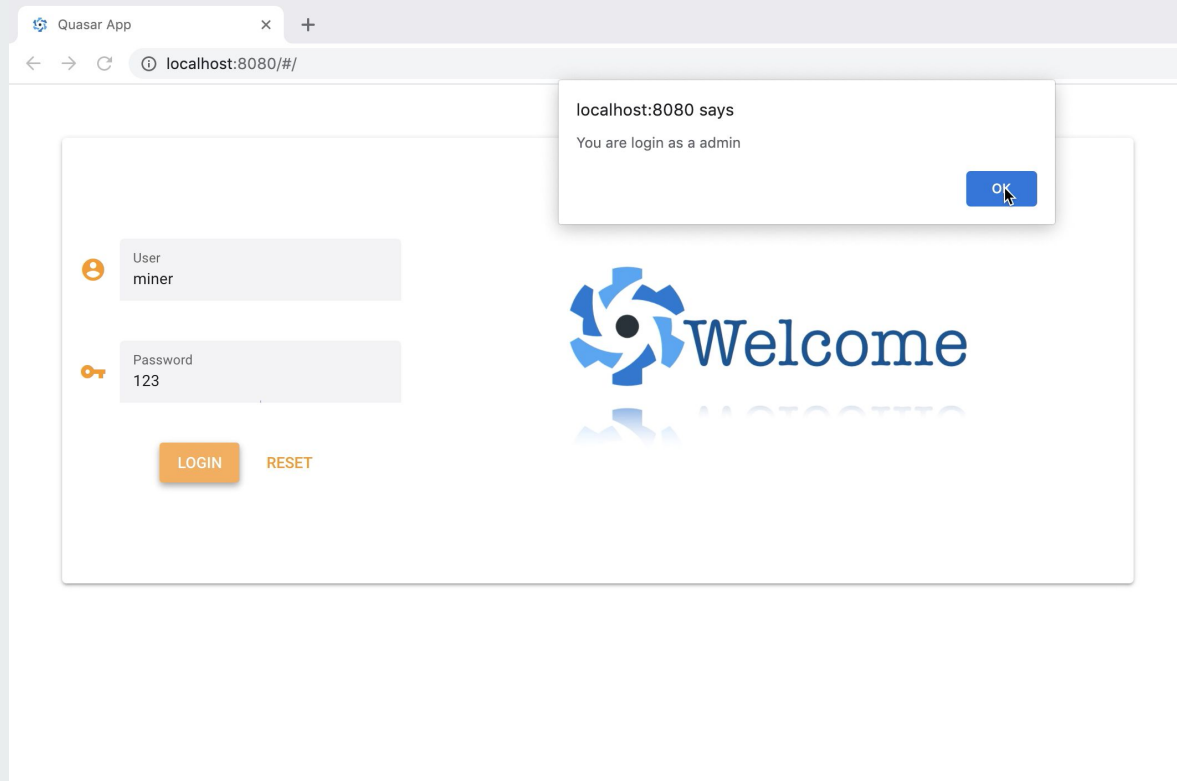
Front End

- Source: /RatingWebsite2.0
- Main job: REST web front end, client to get prediction result, administrator to manage prediction model
- tools: node.js, vue

run the front app

```
cd dist/spa  
python3 -m http.server --bind localhost 8080  
open http://localhost:8080
```

frontend





Labeling methods

Multiclass Classification



(Positive ratings, negative Ratings) => ratings (label) => popularity

How to sort and labeling?

#1 Score = (Positive ratings) – (Negative ratings) ???

#2 Score = Average rating = (Positive ratings) / (Total ratings) ???

item 1 has 2 positive ratings and 0 negative ratings. Suppose item 2 has 100 positive ratings and 1 negative rating.

Size should be considered

Multiclass Classification

(Positive ratings, negative Ratings) => ratings (label) => popularity

$$\left(\hat{p} + \frac{z_{\alpha/2}^2}{2n} \pm z_{\alpha/2} \sqrt{[\hat{p}(1 - \hat{p}) + z_{\alpha/2}^2/4n]/n} \right) / (1 + z_{\alpha/2}^2/n).$$

How to sort and labeling?

Score = Lower bound of Wilson score confidence interval for a Bernoulli parameter

```
val rank = df.sqlContext.sql("SELECT appid,name, " +  
  "((p + 1.9208) / (p + n) - 1.96 * SQRT((p * n) / (p + n) + 0.9604) / (p + n)) / (1 + 3.8416 / (p + n)) " +  
  "AS cilb FROM temp WHERE p + n > 0 ")
```

Multiclass Classification



(Positive ratings, negative Ratings) => ratings (label) => popularity

How to sort and labeling?

```
dfr.withColumn("ratings", when(dfr("cilb") >= percent95, 4.0)
  .when(dfr("cilb") >= percent90, 3.0)
  .when(dfr("cilb") >= percent80, 2.0)
  .when(dfr("cilb") >= percent70, 1.0)
  .otherwise(0.0))
```




Pipeline & feature selection

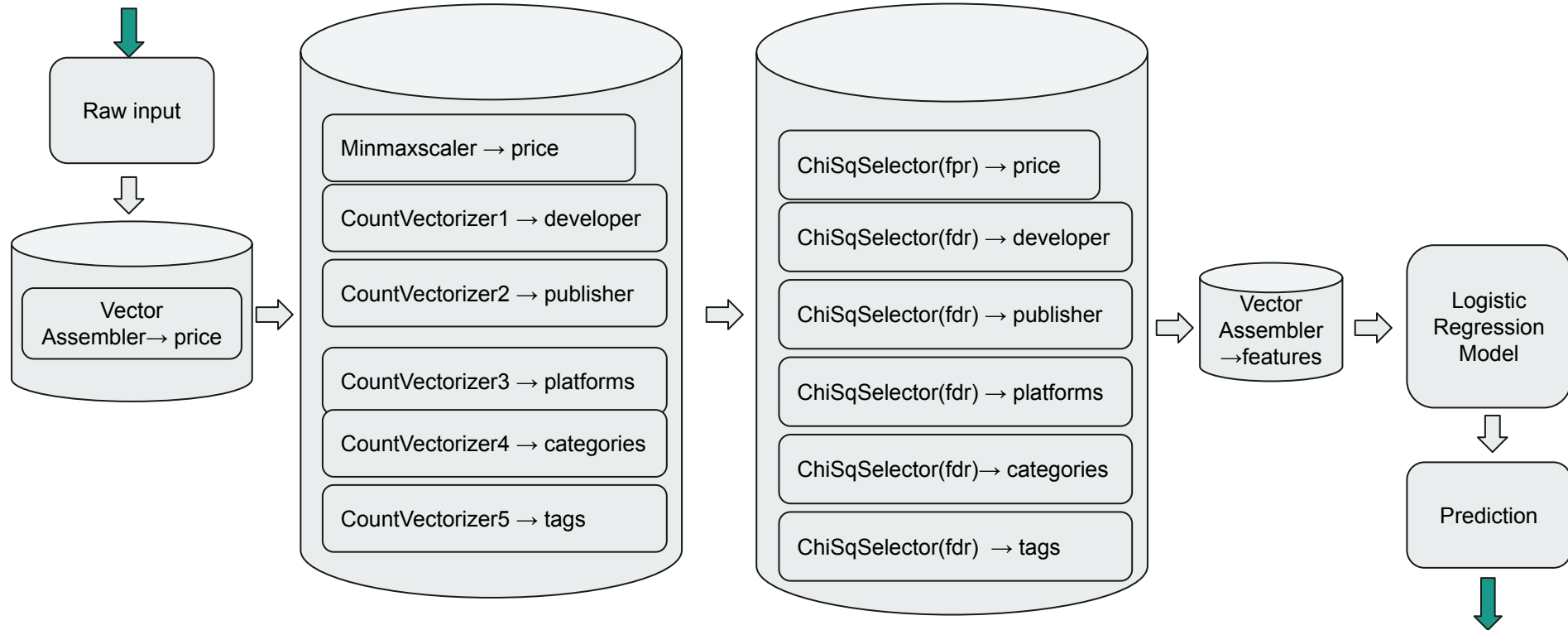
Pipeline & feature selection



Feature transformers & estimators

- ❖ **CountVectorizer:** Extracts a vocabulary from document collections
- ❖ **ChiSqSelector:** for feature selection, which selects categorical features to use for predicting a categorical label
- ❖ **MinMaxScaler:** Rescale each feature individually to a common range . Known as min-max normalization
- ❖ **VectorAssembler:** feature transformer that merges multiple columns into a vector column

Pipeline & feature selection



backend

Back End

- Source: /RatingServer2.0
- Main job: REST web back end, using pipeline to do real-time prediction, real-time model training
- tools: spark, Pyspark, python, flask
- required documents: my_pipeline, best_model, cleandata.parquet

run the back app

run on any python notebook

connection test

```
curl http://localhost:5000/predict  
curl http://localhost:5000/train
```



Model training & Analysis

Model Analysis



Logistic-regression model:

As we all know that LR model is mostly used for indicate the possibility of something happening. So when we use this model to predict. We try to predict a game is good game (with great rating) or not. That's why the precision is about 73%. But it cost less memory to store data.

Naïve Bayes model:

It has a **higher speed** for large numbers of training and queries and it also suitable for incremental training (that is, it can train new samples in real time). So this model would be **quite good** for our project when we need to improve our model.

Multilayer Perceptron Classifier model:

This model has strong self study and adaptation function but it has very slow learning speed. And it will be easy to sink into local extreme value. So that's why its precision is not good.

Random Forest model:

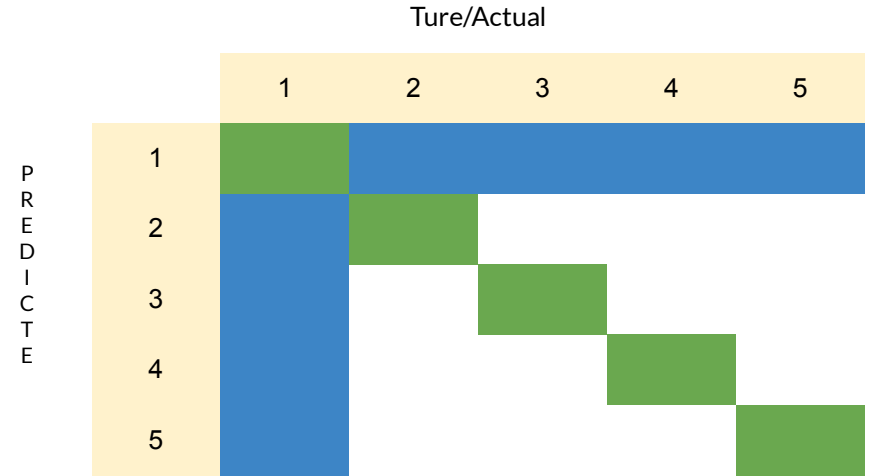
Since our game data are with different values, attributes with more values which will have a greater impact on random forests, so the attribute weights generated by random forests on such data are not credible. So this model has worst performance.

Acceptance Criteria



	Precision	Recall	f1-score
LR	73%	76%	70%
MLP	55%	71%	61%
NB	66%	71%	65%
RF	48%	69%	57%

- Response time less than 2 seconds
- NB has both **good response time** with web and **accuracy criteria**



confusion matrix for multi-class

- Average precision: > 65%
- Average recall: > 65%
- Average f1-score: > 65%
- Response time should less than 4 seconds

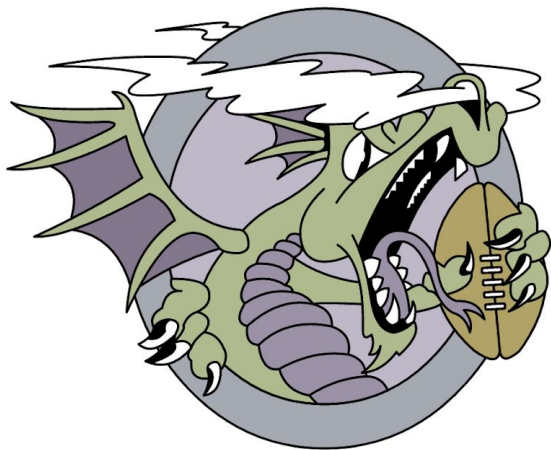


Test & Real time prediction System

Test sample(test inside dataset / new released game on steam)

appid	label	prediction	developer	publisher	platforms	categories
30	3	3	Valve	Valve	windows;mac;linux	Multi-player;Valve .
40	2	3	Valve	Valve	windows;mac;linux	Multi-player;Online
80	3	4	Valve	Valve	windows;mac;linux	Single-player;Multi
130	3	3	Gearbox Software	Valve	windows;mac;linux	Single-player
220	4	4	Valve	Valve	windows;mac;linux	Single-player;Stea
240	4	4	Valve	Valve	windows;mac;linux	Multi-player;Cross
320	3	4	Valve	Valve	windows;mac;linux	Multi-player;Valve .
360	1	4	Valve	Valve	windows;mac;linux	Multi-player;Valve .
380	4	4	Valve	Valve	windows;mac;linux	Single-player;Stea
400	4	4	Valve	Valve	windows;mac;linux	Single-player;Stea
420	4	4	Valve	Valve	windows;mac;linux	Single-player;Stea
440	4	4	Valve	Valve	windows;mac;linux	Multi-player;Cross
500	4	4	Valve	Valve	windows;mac	Single-player;Multi
570	3	3	Valve	Valve	windows;mac;linux	Multi-player;Co-op
730	3	3	Valve;Hidden Path Entertainment	Valve	windows;mac;linux	Multi-player;Stearn
1002	0	0	Mark Healey	Mark Healey	windows	Single-player;Multi

Test sample



developer

CREATIVE ASSEMBLY



publisher

SEGA



platforms

windows

mac

linux



categories

Single-player

Multi-player

Steam Achievements

Steam Trading Cards

Steam Cloud



tags

Strategy

Historical

Turn-Based Strategy

Grand Strategy

RTS

Military

Multiplayer

Action

Great Soundtrack

Turn-Based

Co-op

Simulation

Violent

Sandbox

Fantasy



price

49.99

SUBMIT


CLEAR

Test sample

Validation : <https://steamdb.info/app/261550/>



developer
CREATIVE ASSEMBLY



Prediction result ✓ stars:5
★★★★★


Accuracy i about
0.6981

Popularity Level i about
Overwhelmingly Positive

Team9 production
This prediction result is for reference only & all rights reserved.

GOT IT

Test sample

 Manager LOGOUT

Model Logs

Training history

id	type	accuracy	time
0	Bernoulli Naive Bayes	0.7018	2020-4-12 12:3:26

Records per page: 5 1-1 of 1 < >

Training New Model

Bernoulli Naive Bayes

algorithm to train our model

p1

0.6

split dataset

p2


0.4

split dataset

RESET

START

Test sample

 Manager LOGOUT

Model Logs

Training history

id	type	accuracy	time
0	Bernoulli Naive Bayes	0.7018	2020-4-12 12:3:26
1	Bernoulli Naive Bayes	0.6981	2020-4-14 16:29:19

Records per page: 5 1-2 of 2 < >

Training New Model

Bernoulli Naive Bayes

algorithm to train our model

p1
0.7

split dataset

p2
0.3

split dataset

RESET

START

Thanks!

