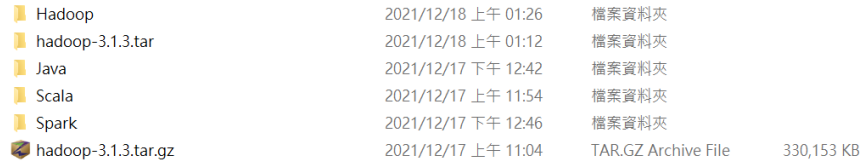
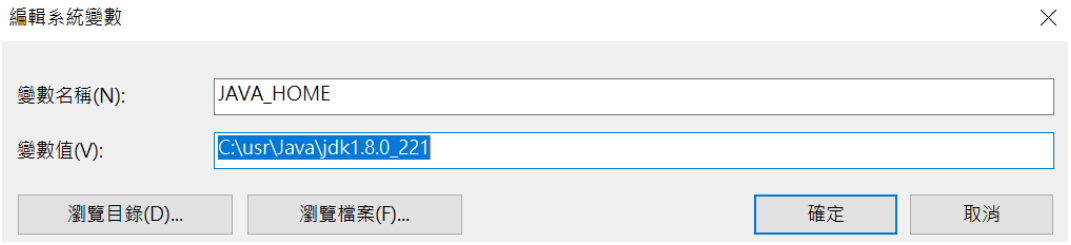
一、安裝部分

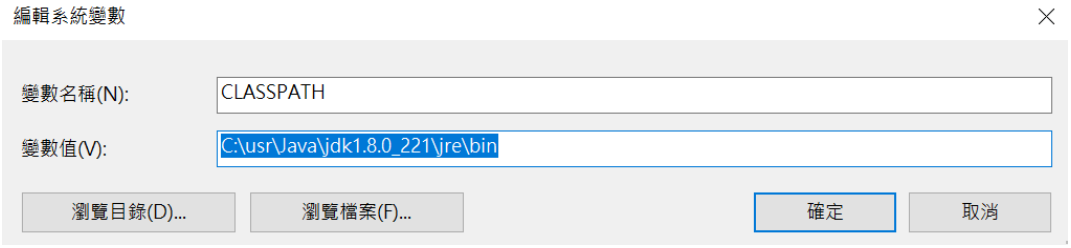
- 安裝 java, scala, hadoop, spark 到自己設定的路徑中



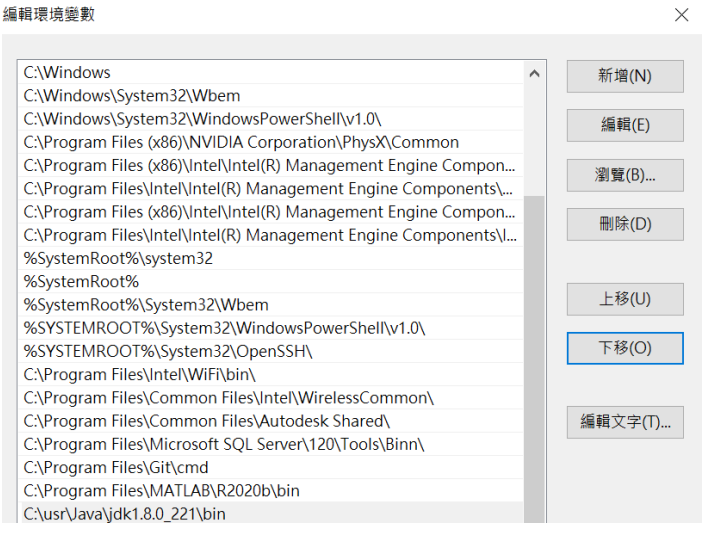
- Java 版本需為 1.8
- 新增環境變數 JAVA_HOME，輸入剛剛安裝 Java 中的 jdk 資料夾的路徑



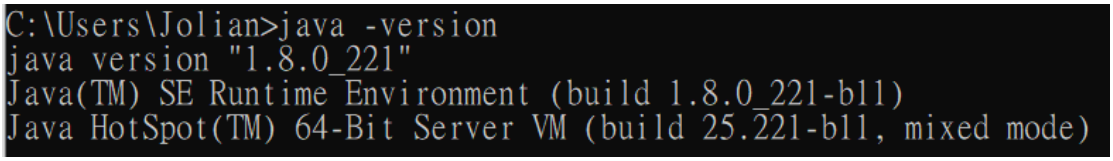
- 新增環境變數 CLASSPATH，輸入剛剛安裝 Java 中的 jre\bin 的路徑



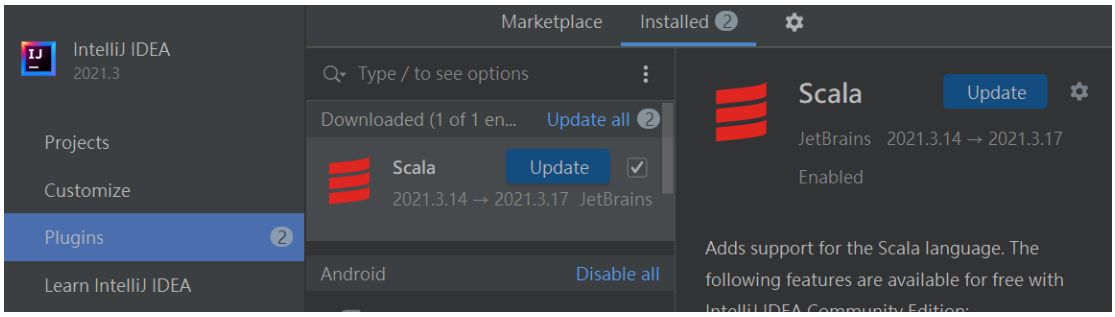
- 環境變數 Path 中，加入 Java 中 bin 資料夾的路徑



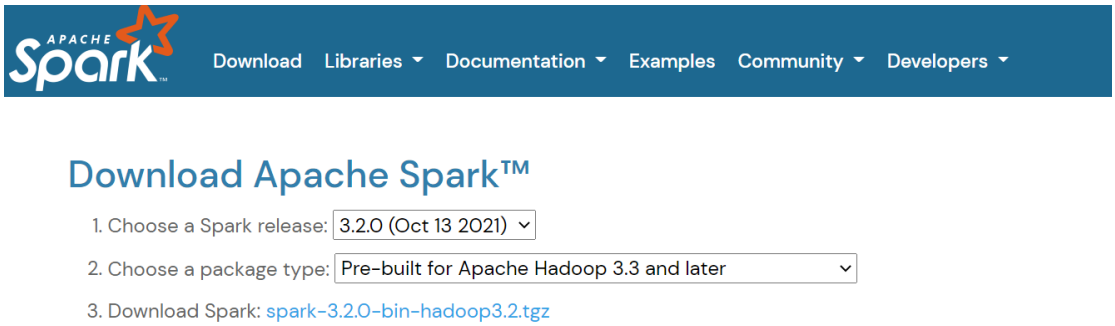
- 輸入 java -version 測試是否安裝完成



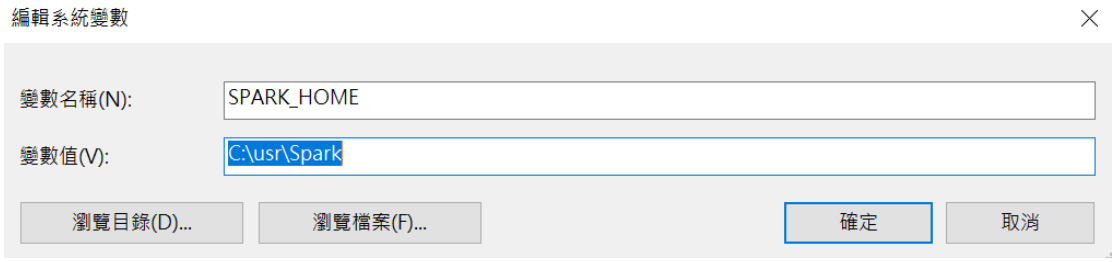
- 安裝 IntelliJ IDEA，完成後開啟，找到 Plugins 中的 Scala 選項點選安裝



- 到 Apache Spark 下載適合版本的 Spark，解壓縮後丟入自己設定的路徑裡

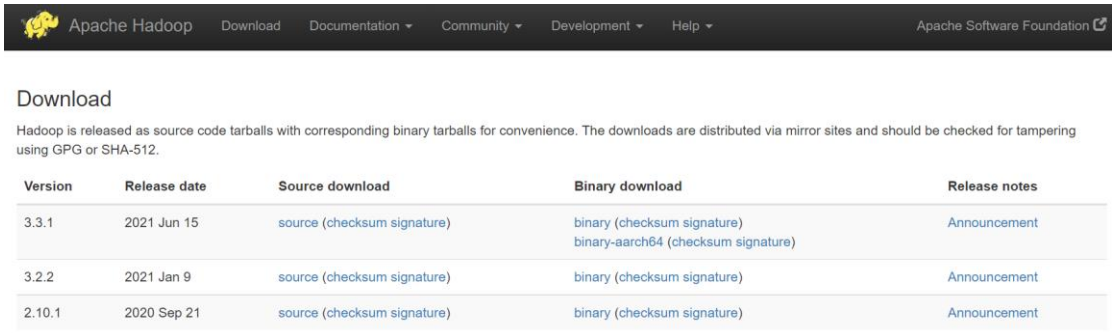


- 新增環境變數 SPARK_HOME，輸入剛剛安裝 Spark 的路徑



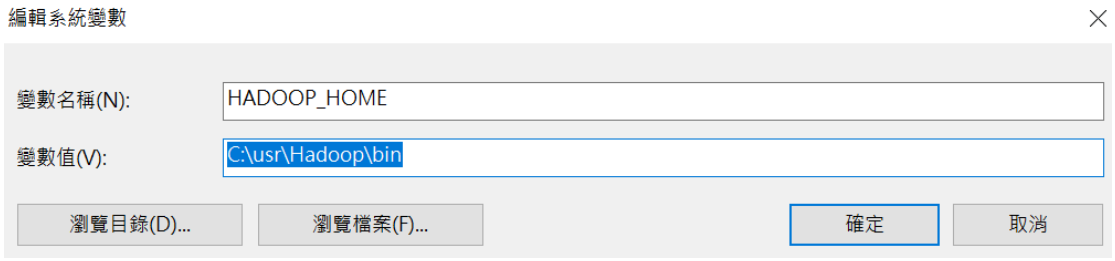
- 環境變數 Path 中，加入 Spark 中 bin 資料夾的路徑

- 到 Apache Hadoop 下載 Hadoop (Binary download)，Hadoop 的版本要和 Spark 的版本互相匹配，同樣解壓縮後丟入自己設定的路徑

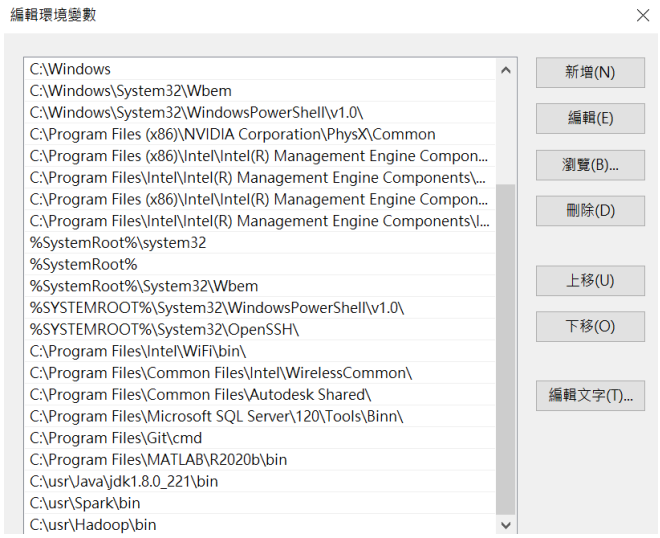


- 到 github (https://github.com/steveloughran/winutils)下載 windows 二進制文件包 winutils，選擇與自己安裝的 hadoop 版本符合的 winutils.exe，下載下來放到 Hadoop 中的 bin 資料夾中

- 新增環境變數 HADOOP_HOME，輸入剛剛安裝 Hadoop\bin 的路徑



- 環境變數 Path 中，加入 Hadoop 中 bin 資料夾的路徑



- 輸入 spark-shell 測試是否安裝完成(有 Spark 和 Scala 的 version)

```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://192.168.0.14:4040
Spark context available as 'sc' (master = local[*], app id = local-1640111042223).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.2
      /_/

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_221)
Type in expressions to have them evaluated.
Type :help for more information.
```

- 輸入 hadoop version 確定安裝成功

```
C:\Users\Jolian>hadoop version
Hadoop 2.7.1
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r 15ecc87ccf4a0228f35af08fc56de536e6ce657a
Compiled by jenkins on 2015-06-29T06:04Z
Compiled with protoc 2.5.0
From source with checksum fc0a1a23fc1868e4d5ee7fa2b28a58a
This command was run using /C:/usr/Hadoop/share/hadoop/common/hadoop-common-2.7.1.jar
```

- 配置 hadoop 偽分布式，在單機上配置 hadoop 節點，模擬分散式計算。
- hadoop\etc 中找到 core-site.xml，加入 configuration 如下(value 路徑要改)

```
<!-- Put site-specific property overrides in this file. -->
<configuration>
    <property>
      <name>hadoop.tmp.dir</name>
      <value>/C:/hadoop/hadoop-3.1.3/data</value>
    </property>
    <property>
      <name>fs.defaultFS</name>
      <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

- hadoop\etc 中找到 hdfs-site.xml，加入 configuration 如下(value 路徑要改)

```
<!-- Put site-specific property overrides in this file. -->
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>/C:/usr/Hadoop/data/datanode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>/C:/usr/Hadoop/data/datanode</value>
    </property>
</configuration>
```

- hadoop\etc 中找到 mapred-site.xml.template，檔名改為 mapred-site.xml，並加入 configuration 如下

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

- hadoop\etc 中找到 yarn-site.xml，加入 configuration 如下

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hahoop.mapred.ShuffleHandler</value>
    </property>
</configuration>
```

- 到 cmd 中執行 hdfs namenode -format。執行完(看到 SHUTDOWN_MSG 出現)後，到 hadoop\data 下，找到新增的 current 資料夾即代表設置偽分布節點成功

二、程式碼部分

- [1]：import packages，用 findspark 找到 SPARK_HOME 路徑啟用，之後可以 import spark，SparkConf 設定集合生成 RDD，使用 local 執行

In [1]:
```
import pandas as pd
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
conf = pyspark.SparkConf().setAppName('sparkApp').setMaster('local')
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)
print(sc)
numeric_val = sc.parallelize([1,2,3,4])
numeric_val.map(lambda x:x*x*x).collect()
sc.stop
import mllib
```

```
<SparkContext master=local appName=sparkApp>
```

- [2]：從 pyspark.ml.recommendation 中 import ALS 演算法，並 create session 供後續進行 movieRating 的預測

In [2]:
```
#import module
from pyspark.ml.recommendation import ALS

#create session
appName = "Recommender system in Spark"
spark = SparkSession \
    .builder \
    .appName(appName) \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

- [3]：讀取資料(data_orin)，將 TrainId 這個 column 去掉，並使用 pyspark.sql.DataFrame.randomSplit 將資料 shuffle 並隨機按比例分成訓練集(train)與測試集(test)。將 train 和 test 的 movie Rating 欄分別改名為 Label 和 trueLabel。資料如下，可看到 train 有 719625 tuples，test 有 180248 tuples

In [3]:
```
data_orin = spark.read.csv('./data/movieRating.csv', inferSchema=True, header=True).select("UserId", "MovieId", "Rating")
splits = data_orin.randomSplit([0.8, 0.2], seed=10)
train = splits[0].withColumnRenamed("Rating", "Label")
test = splits[1].withColumnRenamed("Rating", "trueLabel")
# aaa.printSchema()
train.show(5)
test.show(5)
train_rows = train.count()
test_rows = test.count()
print ("number of training data rows:", train_rows,
       ", number of testing data rows:", test_rows)
```

```
+------+-------+-----+
|UserId|MovieId|Label|
+------+-------+-----+
|     1|     11|    5|
|     1|     85|    3|
|     1|    249|    4|
|     1|    260|    5|
|     1|    265|    4|
+------+-------+-----+
only showing top 5 rows

+------+-------+---------+
|UserId|MovieId|trueLabel|
+------+-------+---------+
|     1|     36|        5|
|     1|    150|        5|
|     1|    261|        4|
|     1|    356|        5|
|     1|    531|        5|
+------+-------+---------+
only showing top 5 rows

number of training data rows: 719625 , number of testing data rows: 180248
```

- [4]：宣告一個 ALS 推薦模型，ALS 演算法屬於 User-Item CF (協同過濾)，每筆資料包含<User,Item,Rating>。假設有 m 個 User 和 n 個 Item，則定義 Rating 矩陣 R，其元素 R_{ui} 表示第 u 個 User 對第 i 個 Item 的評分。但因為 user 不可能對每個商品都有評分，所以矩陣 R 為一稀疏矩陣，用傳統的矩陣方式計算會耗費不必要的資源，缺乏效率。在 ALS 方法中，假定 User 與 Item 中會有某些關聯性(例如：User 的年齡、性別、教育程度等會影響選擇 Item 的類別)，這種關聯可以看成一種空間投射，這個空間稱作 Latent factor，維度 k 的取值一般是 20～200，則原本的維度 m*n 的 R 矩陣可以分解成$(m*k)*(n*k)^T$，這種作法稱為概率矩陣分解演算法(probabilistic matrix factorization，PMF)。一般情況下，k 的值會遠小於 n 和 m 的值，從而達到了資料降維的目的。ALS 雖增加運算效率與準確度，但無法準確評估新加入的用戶或商品(即 Cold-start Problem)，但在本次實作中，train 和 test 是 shuffle 後再分開成訓練、測試集，所以應該比較不會受到此問題的影響
- 使用 pyspark.ml.recommendation 的 ALS 套件，其中參數 maxIter 表示 ALS 的 iteration 次數，regParam 表示 regularization parameter，userCol 為欲推薦用戶的特徵資料，itemCol 為欲推薦項目的特徵資料，ratingCol 則為 actual label，用以與模型預測的 rating (predict)結果進行比對，並根據 Loss function 回推修正參數，直到模型訓練完畢。model 為使用 als 宣告模型對 train 資料進行訓練後，得出的模型。

```
In [4]: #define ALS (Alternating Least Square) as our recommender system
als = ALS(maxIter=19, regParam=0.01, userCol="UserId",
          itemCol="MovieId", ratingCol="Label")
#train our ALS model
model = als.fit(train)
print("Training is done!")
```

```
Training is done!
```

- [5]：prediction 為使用訓練完畢的模型 model 去預測 test 資料 label 的結果

```
In [5]: prediction = model.transform(test)
print("testing is done!")
```

```
testing is done!
```

- [6]：用 pyspark.ml 的 RegressionEvaluator 套件計算 MAE，宣告一個 evaluator，參數中 labelCol 為資料真正的 label (對該電影的評分)，predictionCol 為剛剛用 model 預測的 label 結果，metricName 設定成計算 MAE，將 prediction 放入 evaluator，但得到 nan，猜測因為 prediction 中有缺值，導致無法計算。

```
In [6]: #import RegressionEvaluator since we also want to calculate RMSE (Root Mean Square Error)
from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(
    labelCol="trueLabel", predictionCol="prediction", metricName="mae")
mae = evaluator.evaluate(prediction)
print ("MAE:", mae)
```

```
MAE: nan
```

- [7]：用 dropna 去掉 prediction 中的缺失值，並 print 出原 prediction 與去掉缺值得 cleanPred 的資料筆數，可以看到原 prediction 有 180248 筆資料，cleanPred 只有 180226 筆，共有 22 筆資料含缺失值，將其刪除

```
In [7]: a = prediction.count()
        print("number of original data rows: ", a)
        #drop rows with any missing data
        cleanPred = prediction.dropna(how="any", subset=["prediction"])
        b = cleanPred.count()
        print("number of rows after dropping data with missing value: ", b)
        print("number of missing data: ", a-b)

        number of original data rows:  180248
        number of rows after dropping data with missing value:  180226
        number of missing data:  22
```

- [13]：用刪除缺失值的預測結果 cleanPred 去重新計算 MAE，得到 MAE 約為 0.7

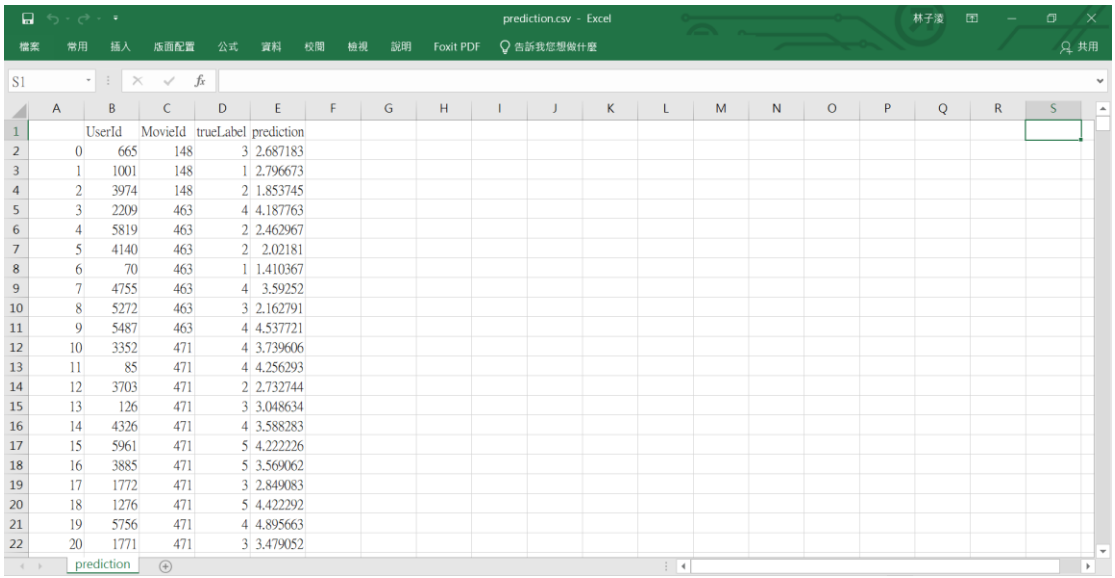
```
In [13]: mae = evaluator.evaluate(cleanPred)
         print ("mae:", mae)

         mae: 0.6972219239848941
```

- [14]：將預測結果轉為 pandas dataframe，並使用 to_csv 將結果存入./result/prediction.csv 中

```
In [14]: pred = prediction.toPandas()
         pred.to_csv('./result/prediction.csv')
```

- ./result/prediction.csv 結果，可以看到資料包含<User, Item, Label, prediction> trueLabel 為測試集原本 user 對該 item 的 rating，prediction 則為 ALS 模型預測出的 rating.



prediction.csv - Excel

	A	UserId	MovieId	trueLabel	prediction
2	0	665	148	3	2.687183
3	1	1001	148	1	2.796673
4	2	3974	148	2	1.853745
5	3	2209	463	4	4.187763
6	4	5819	463	2	2.462967
7	5	4140	463	2	2.02181
8	6	70	463	1	1.410367
9	7	4755	463	4	3.59252
10	8	5272	463	3	2.162791
11	9	5487	463	4	4.537721
12	10	3352	471	4	3.739606
13	11	85	471	4	4.256293
14	12	3703	471	2	2.732744
15	13	126	471	3	3.048634
16	14	4326	471	4	3.588283
17	15	5961	471	5	4.222226
18	16	3885	471	5	3.569062
19	17	1772	471	3	2.849083
20	18	1276	471	5	4.422292
21	19	5756	471	4	4.895663
22	20	1771	471	3	3.479052