人工智能與財務應用 期末小組報告

班別：財金三甲

2021/01/01

主題：台積電股價模擬測試

組別：第 **12**組

組長：黃婉華**A107230045**

組員：

岳昱均A107230073

彭辰瑄A107260067

汪家彣A107230209

1、 分析目的及資料取樣期間

I. 分析目的

以台積電(2330)之2018年7月1日至2019年9月底之每日收盤價轉換成依變數，若該日股價大於前一日股價則以Y=1表示，反之則以Y=0表示，因爲股價受市場供需關係影響，而其中技術指標最能反應市場狀態，因此我們自變數採取技術指標，其中我們旨在處理短線操作預測，故我們以MA5、MA10作短期預測的目的性指標，另外採取業界最主要應用的前三個特徵指標，包括KD指標、MACD、RSI作為投資行為人的預測指數，最後以業界常用手法之量先價行的觀念，將成交量作為我們第一個指標基準。

II. 資料取樣期間

經由XQ全球贏家中抓取2018年7月1日至2019年10月底之台積電股價與技術指標，我們將2018年7月1日至2019年8月底之資料作為訓練資料，而2019年9月之資料作為測試資料，另外抓取2019年10月之資料作為新測試資料。

二、解釋變數的定義及目前業界應用面或報告使用原因

I. 成交量(Trading Volume)

量先價行是技術分析裡面常用到的一個觀念，是運用成交量領先股價反應的基礎，做為預測未來股價走勢的一個方法，因為通常股價無論是要上漲或是下跌，都會透過市場的成交量來做反應。

當股價位於低檔盤整，在起漲發動以前 都會出現連續性的遞增量 或是3-5次不等的相對大量，或是股票在連續下跌之後，也會出現1-2次的相對大量，尋求支撐或是止跌，這都代表著市場籌碼經過了換手和沉澱。

以下為簡易量價型態：

a. 量增價漲、量縮下跌，股價隨著趨勢走
   1. 一般型：上漲持續量增直到A點量失控(超大量)。
   2. 複雜型：A點量縮B點放量突破頸線，為股價上漲的訊號。
b. 多頭反轉型(轉空)：量與價背離量縮價漲
   1. 一般型：量價連續背離。
   2. 複雜型：股價高檔震盪，破線出大量後量縮，反轉點在A點。
c. 空頭反轉(轉多)：量與價背離量增價跌
   1. 一般型：築底出量，A點與B點是技術面之一二三法則(N型)
   2. 複雜型：急跌出量，但在最後最低點量放至最大則出現反轉。若配合「缺口」有機會出現V型態反轉

II. 5日均線(MA5)

為過去 5 日均量。

為過去 10 日均量

成交量為人氣聚集與否的指標。當一檔個股成交量突然上升，代表某些事情的發生而吸引了先知先覺的人參與市場交易，所以觀察成交量的變化，可以更明確的掌握每檔個股的變化。即所謂的量是價的先行指標。
而成交量的多寡代表著流動性的意義，成交量的多寡經常是法人或大戶決定是否進出某股票的重要考量。

IV. 震盪量指標(OSC)，也稱為變動速率線

利用快線(DIF)與慢線(DEM)的差額，為MACD指標圖形中的柱狀線(MACD bar / OSC)，亦為震盪量指標OSC、變動速率線。

$$OSC = DIF - DEM = DIF - MACD$$

或簡寫為 $OSC = D - M$

柱狀線(OSC)由負轉正及由正轉負，與黃金交叉、死亡交叉有著相同的意涵，且柱狀線正負向之長度越長，代表近期股價偏向多方或空方的力道越強，而MACD柱狀線在盤整走勢中於零軸附近上下徘徊，則此時指標不具太大參考價值。此外，在漲多回檔的格局中，MACD柱狀線亦可能出現短暫跌破零軸又再次翻正的干擾訊號(跌深反彈時亦同)。

其中，差離值(DIF)為利用收盤價的指數移動平均值(12日/ 26日)計算出差離值。

$$DIF = EMA_{(close,12)} - EMA_{(close,26)}$$

而訊號線(DEM，又稱MACD[1]線)則是在計算出DIF後，再將DIF線再作一次指數移動平均，通常是DIF的9日指數移動平均值畫一條訊號線。

$$DEM = EMA_{(DIF,9)}$$

快線及慢線的分別為DIF線只經過一次指數移動平均平滑處理，對股價變動的反應較為迅速，因此我們把它視為「快線」，而經過兩次指數移動平均平滑處理的DEM線，對股價變動反應較為遲緩，我們將其視為「慢線」。

V. D值

D值(稱為慢線)，將該日加權平均後的K值和前日D值再平均一次，而經過兩次平滑、該日最新股價對它的影響會較小，因此對當前股價變動的反應會比較慢。

$$D(day, weight) = D(9, 3)$$

VI. K值

[1] MACD指標

平滑異同移動平均線指標(Moving Average Convergence Div[illegible]gence，MACD)，利用快慢[illegible]的交錯來判斷股[illegible]走勢[illegible]的轉折，
快線由下向上突破慢線時，即為黃金交叉，代表股價後市看漲；[illegible]當快[illegible]由上而[illegible]慢線[illegible]死[illegible]叉，代[illegible]市

K值(稱為快線)，將該日股價算出的RSV和前日K值取加權平均，對當前股價變動的反應會比較快速。

$$K(day,\ weight)\ =\ K(9,3)$$

❖

超買：KD值>80時股價表現較強勢，有較高機率將會繼續上漲。

超賣：KD值<20時股價表現較弱勢，有較高機率將會繼續下跌。

黃金交叉：當K值從低檔往上突破D值時，短期可能轉折從低點向上漲。

死亡交叉：當K值從高檔往下跌破D值時，短期可能轉折從高檔向下跌。

VII. 相對強弱指數(Relative Strength Index , RSI)

主要是用來評估股市中買賣盤雙方力道的強弱，衡量近期價格變化的幅度，以評估股價超買或超賣情況。

三、分析流程及使用之AI模型(若有參數請詳述)

■ 決策樹(decision tree)

將演算法組織成一顆樹的形式(可以是二叉樹或非二叉樹)。其每個非葉節點表示一個特徵屬性上的測試，每個分支代表這個特徵屬性在某個值域上的輸出，而每個葉節點存放一個類別。使用決策樹進行決策的過程就是從根節點開始，測試待分類項中相應的特徵屬性，並按照其值選擇輸出分支，直到到達葉子節點，將葉子節點存放的類別作為決策結果。

決策樹

```
tree= DecisionTreeClassifier(criterion='gini',max_depth=5) #以Gini不純度指標切割，而每次切割都是基於Gini不純度指標下降最多，且max_depth=5指示最多到五層 # 對knn模型進行訓練，並傳入樣本特徵和樣本標簽
tree.fit(train_X, train_Y) # 構建函數原型、構建損失函數、求損失函數之最優解
pred_Y_tree = tree.predict(test_X) # 使用predict函數對數據進行預測
pred_acc_tree = accuracy_score(test_Y,pred_Y_tree) #test_Y是真實的資料 #看樣本外的能力
loss_Y_tree = tree.predict(train_X) # 使用predict函數對數據進行預測
loss_acc_tree = accuracy_score(train_Y,loss_Y_tree) #看樣本內的能力
print(pred_acc_tree, loss_acc_tree) #(樣本外預測力，樣本內解釋力)
```

參數定義

使用DecisionTreeClassifier 之Sklearn工具套件中的參數：

Criterion：使用Gini係數切割也等同於使用Gini不純度指標分割，而每次的切割都是基於Gini不純度指標下降最多。

max_depth：決策樹的最大深度，而max_depth=5代表此決策樹深度最多到五層。

■ 單純貝氏分類器(Naive Bayes Classifier)

單純貝氏分類器為透過貝氏定理的計算，我們可得知在已知的資料下發生最大機率之目標為何，並由此結果作為分類標準(預測類別)，且模型中假設所有的特徵都是獨立的。

```
#貝氏分類器
bayes = GaussianNB()
bayes.fit(train_X, train_Y)
pred_Y_bayes = bayes.predict(test_X)
pred_acc_bayes = accuracy_score(test_Y,pred_Y_bayes)
loss_Y_bayes = bayes.predict(train_X)
loss_acc_bayes = accuracy_score(train_Y,loss_Y_bayes)
print(pred_acc_bayes, loss_acc_bayes)
```

■ 支持向量機support vector machine, SVM(linear)

SVM是一種監督式的學習方法，用統計風險最小化的原則來估計一個分類的超平面(hyperplane)，其基礎的概念是找到一個決策邊界(decision boundary)讓兩類之間的邊界(margins)最大化，使其可以完美區隔開來，如果能用一條直線就把紅色與藍色的點分開，則代表我們蒐集的訓練集合是線性可分的SVM(linear)，若沒辦法只用一條直線就把紅色與藍色的點分開，則代表我們蒐集的訓練集合是線性不可分的SVM(nonlinear)。

```
#SVM(linear)
param_list1 = {'C': [0.1, 0.25, 0.5 ,0.75, 1, 3, 5, 7, 9, 15, 20, 100, 500]}
cv_svm1 = GridSearchCV(SVC(kernel='linear'), param_list1)
cv_svm1.fit(train_X, train_Y)
best_C_svm1=cv_svm1.best_estimator_.C
best_C_svm1

svm1 = SVC(kernel='linear',C=best_C_svm1)
svm1.fit(train_X,train_Y)
pred_Y_svm1 = svm1.predict(test_X)
pred_acc_svm1 = accuracy_score(test_Y,pred_Y_svm1)
loss_Y_svm1 = svm1.predict(train_X)
loss_acc_svm1 = accuracy_score(train_Y,loss_Y_svm1)
print(pred_acc_svm1, loss_acc_svm1)
```

參數定義

使用SVC之Sklearn工具套件中的參數：

C懲罰係數：為對誤差的寬容度，以控制懲罰程度強弱。C越高越不能容忍出現誤差，因為容易過擬合，其表現與Hard SVM相似。C越小則容易欠擬合，所以C過[illegible]或過[illegible]，其分[illegible]力[illegible]在此[illegible]們[illegible]建立C[illegible]

[0.1, 0.25, 0.5 ,0.75, 1, 3, 5, 7, 9, 15, 20, 100, 500]，以GridSearchCV (Cross Validation,CV交叉驗證)從中挑選最佳的C，而我們藉此得知C為 100 (亦為best_C_svm1=100)。

Kernel function核函數：當不同類別的資料在原始空間中無法被線性分類器區隔開來時，經由非線性投影後的資料，為映射關係的內積，但映射函數只是一種映射關係，並沒有增加維度的特性，不過可以利用核函數的特性，構造可以增加維度的核函數，能在更高維度的空間中更容易被區隔。此時，我們希望SVM是利用線性關係切割類別，因此 kernel = linear。

- SVM(nonlinear)

```
#SVM(nonlinear)
param_list2 = {'C': [0.1, 0.5, 1, 5, 10, 15, 20, 50, 100],'gamma': [0.01, 0.1 ,0.25, 0.5,0.75, 1, 3, 5, 10, 20]}
cv_svm2 = GridSearchCV(SVC(kernel='rbf'), param_list2)
cv_svm2.fit(train_X, train_Y)
best_C_svm2=cv_svm2.best_estimator_.C
best_gamma_svm2=cv_svm2.best_estimator_.gamma
svm2 = SVC(kernel='rbf',C=best_C_svm2, gamma=best_gamma_svm2)
svm2.fit(train_X,train_Y)
pred_Y_svm2 = svm2.predict(test_X)
pred_acc_svm2 = accuracy_score(test_Y,pred_Y_svm2)
loss_Y_svm2 = svm2.predict(train_X)
loss_acc_svm2 = accuracy_score(train_Y,loss_Y_svm2)
print(pred_acc_svm2, loss_acc_svm2)
```

參數定義

使用SVC之Sklearn工具套件中的參數：

Radial Based Function，RBF函數(徑向核函數)：表示存在無限多維空間。

Gamma：是選擇RBF函數作為kernel後，該函數自帶的一個參數，決定數據映射到新的特徵空間後的分布，gamma越大其支持向量越少，gamma值越小其支持向量越多，而支持向量的個數影響訓練與預測的速度。

- Logistic Regression (羅吉斯回歸、邏輯回歸)

邏輯迴歸可用於了解兩個或多個變數間是否相關、相關方向與強度關係，應用於離散變數的分類，其輸出值y表示屬於某一類的概率，亦其目的為找出一條最能夠代表所有觀測資料的函數，而一個單獨的邏輯迴歸函式只能判別兩個類別並以0和1表示，邏輯迴歸的結果會給出一個概率p，表示屬於類別1的概率。

```
#Logistic Regression
```

```
pred_Y_logist = logist.predict(test_X)
pred_acc_logist = accuracy_score(test_Y,pred_Y_logist)
loss_Y_logist = logist.predict(train_X)
loss_acc_logist = accuracy_score(train_Y,loss_Y_logist)
print(pred_acc_logist, loss_acc_logist)
```

參數定義

使用LogisticRegression之Sklearn工具套件

■ 隨機森林(Random Forest)

是用隨機挑選決策樹的方式以建立一個森林，因此每棵決策樹之間皆無關聯。而在建立完隨機森林後，當新的資料進入時，每一棵決策樹將分別進行判斷，以測試此資料應屬於的類別為何，其中隨機森林中的每棵樹在產生過程中，接已考慮到須避免共線性、避免過擬合，以達分類的完整性。

```
# 隨機森林(Random Forest)
forest = RandomForestClassifier(n_estimators=500) #可以挑幾棵樹 100~500
forest.fit(train_X,train_Y)
pred_Y_forest = forest.predict(test_X)
pred_acc_forest = accuracy_score(test_Y,pred_Y_forest)
loss_Y_forest = logist.predict(train_X)
loss_acc_forest = accuracy_score(train_Y,loss_Y_forest)
print(pred_acc_forest, loss_acc_forest)
```

參數定義

使用RandomForestClassifier 之Sklearn工具套件中的參數：

n_estimators：隨機森林中的決策樹數量，也為弱學習器的最大迭代次數，亦可稱為最大的弱學習器的個數。另外n_estimators太小會容易欠擬合，而n_estimators太大則容易過擬合，預設值是100~500，但若是需要調整參數時，n_estimators則需要和引數learning_rate同時考慮。

■ KNN (K-Nearest Neighbor最近鄰居法)

KNN對於預測的樣本數據（尚未分類的資料）將與訓練資料進行比較，而樣本集中的每個樣本都要與待測樣本的進行歐基里德距離 (Euclidean distance)的計算，然後在其中取k個最近鄰居(訓練資料)，並以這K個訓練樣本中出現最多的分類標籤作為最終新樣本數據的預測標籤。

K值的挑選

預期n_neighbors <= n_samples，因為n_neigbors=n_samples，每個點的距離度量計算，總數為組合C(n_samples, n_neigbors)是1，但若是以 n_neighbors [illegible]

因為K值太大可能包含許多並不相關的樣本點，故我們選擇K為3，而K值太小則容易受到噪音的影響。依據經驗法則要盡量讓K值低於樣本數的平方根，但依據實務方面我們查到也可以透過不斷拆分訓練資料與測試資料的交叉驗證，以找出較穩定的K值。

歐基里德距離 (Euclidean distance)

$$d(x_k, x_t) = \sqrt{(x_{k1} - x_{t1})^2 + (x_{k2} - x_{t2})^2 + \ldots + (x_{kp} - x_{tp})^2}$$

```
# KNN
knn = KNeighborsClassifier(n_neighbors=3)  #鄰居數量
knn.fit(train_X,train_Y)
pred_Y_knn = knn.predict(test_X)
pred_acc_knn = accuracy_score(test_Y,pred_Y_knn)
loss_Y_knn = knn.predict(train_X)
loss_acc_knn = accuracy_score(train_Y,loss_Y_knn)
print(pred_acc_knn, loss_acc_knn)
```

參數定義

使用KNeighborsClassifier 之Sklearn工具套件中的參數：

n_neighbors：為K值，代表指定搜尋最近的樣本數據的數量。

四、每種模型之結果(分樣本內及外) 及最終模型之選擇依據

```
pred_res=[pred_acc_tree,pred_acc_bayes,pred_acc_svm1,pred_acc_svm2,pred_acc_logist,pred_acc_forest,pred_acc_knn]
loss_res=[loss_acc_tree,loss_acc_bayes,loss_acc_svm1,loss_acc_svm2,loss_acc_logist,loss_acc_forest,loss_acc_knn]
print(pred_res)
print(loss_res)
```

```
[0.7058823529411765, 0.6470588235294118, 0.6470588235294118, 0.6470588235294118, 0.7058823529411765, 0.7647058823529411, 0.529
4117647058824]
[0.8384879725085911, 0.5979381443298969, 0.5189003436426117, 0.5154639175257731, 0.563573883161512, 0.563573883161512, 0.73539
51890034365]
```

- 決策樹(decision tree)

樣本內解釋能力為0.8384879725085911約為0.84

樣本外預測能力為0.7058823529411765約為0.71

- 單純貝氏分類器(Naive Bayes Classifier)

樣本內解釋能力為0.5979381443298969約為0.60

樣本外預測能力為0.6470588235294118約為0.65

- 支持向量機support vector machine, SVM(linear)

樣本內解釋能力為0.5189003436426117約為0.52

- SVM(nonlinear)

樣本內解釋能力為0.5154639175257731約為0.52

樣本外預測能力為0.6470588235294118約為0.65

- Logistic Regression (羅吉斯回歸、邏輯回歸)

樣本內解釋能力為0.563573883161512約為0.56

樣本外預測能力為0.7058823529411765約為0.71

- 隨機森林(Random Forest)

樣本內解釋能力為0.563573883161512約為0.56

樣本外預測能力為0.7647058823529411約為0.76

- KNN (K-Nearest Neighbor最近鄰居法)

樣本內解釋能力為0.7353951890034365約為0.74

樣本外預測能力為0.5294117647058824約為0.53

- 最終以預估能力與解釋能力相比其餘模型有著較好結果，為我們小組選擇模型的標準，而決策樹(decision tree) 樣本內解釋能力為0.8384879725085911約為0.84樣本外預測能力為0.7058823529411765約為0.71，因此我們挑選由決策樹(decision tree)所分類的模型為最佳。

• Code

```
import os
os.getcwd()
```

```
'C:\\Users\\sweet\\Desktop'
```

```
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold,StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
import random
random.seed(1)
data0=pd.read_csv('2330.csv')
data0_train=data0.iloc[0:291,:]
data0_test=data0.iloc[292:309,:]
train_X=data0_train[['Volume','MA5','MA10','OSC','D(9,3)','K(9,3)','RSI_6']]
train_Y=data0_train['LABAL']
test_X=data0_test[['Volume','MA5','MA10','OSC','D(9,3)','K(9,3)','RSI_6']]
test_Y=data0_test['LABAL']

#決策樹
tree= DecisionTreeClassifier(criterion='gini',max_depth=5) #以gini部純度切割，max_depth=5只最多到五層
tree.fit(train_X, train_Y)
pred_Y_tree = tree.predict(test_X)
pred_acc_tree = accuracy_score(test_Y,pred_Y_tree) #test_Y是真實得資料
loss_Y_tree = tree.predict(train_X) #看樣本外的能力
loss_acc_tree = accuracy_score(train_Y,loss_Y_tree)
print(pred_acc_tree, loss_acc_tree) #看樣本內的能力
```

```
0.7058823529411765 0.8384879725085911
```

```
#貝氏分類器
bayes = GaussianNB()
bayes.fit(train_X, train_Y)
pred_Y_bayes = bayes.predict(test_X)
pred_acc_bayes = accuracy_score(test_Y,pred_Y_bayes)
loss_Y_bayes = bayes.predict(train_X)
loss_acc_bayes = accuracy_score(train_Y,loss_Y_bayes)
print(pred_acc_bayes, loss_acc_bayes)
```

```
0.6470588235294118 0.5979381443298969
```

```
#SVM(linear)
param_list1 = {'C': [0.1, 0.25, 0.5 ,0.75, 1, 3, 5, 7, 9, 15, 20, 100, 500]}
cv_svm1 = GridSearchCV(SVC(kernel='linear'), param_list1)
cv_svm1.fit(train_X, train_Y)
best_C_svm1=cv_svm1.best_estimator_.C
best_C_svm1
```

```
C:\Users\sweet\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:605: Warning: The least populated class in y has only 1 members, which is too few. The minimum number of members in any class cannot be less than n_splits=3.
  % (min_groups, self.n_splits)), Warning)
```

```
100
```

```
svm1 = SVC(kernel='linear',C=best_C_svm1)
svm1.fit(train_X,train_Y)
pred_Y_svm1 = svm1.predict(test_X)
pred_acc_svm1 = accuracy_score(test_Y,pred_Y_svm1)
loss_Y_svm1 = svm1.predict(train_X)
loss_acc_svm1 = accuracy_score(train_Y,loss_Y_svm1)
print(pred_acc_svm1, loss_acc_svm1)
```

```
0.6470588235294118 0.5189003436426117
```

```
#SVM(nonlinear)
param_list2 = {'C': [0.1, 0.5, 1, 5, 10, 15, 20, 50, 100],'gamma': [0.01, 0.1 ,0.25, 0.5,0.75, 1, 3, 5, 10, 20]}
cv_svm2 = GridSearchCV(SVC(kernel='rbf'), param_list2)
cv_svm2.fit(train_X, train_Y)
best_C_svm2=cv_svm2.best_estimator_.C
best_gamma_svm2=cv_svm2.best_estimator_.gamma
svm2 = SVC(kernel='rbf',C=best_C_svm2, gamma=best_gamma_svm2)
svm2.fit(train_X,train_Y)
pred_Y_svm2 = svm2.predict(test_X)
pred_acc_svm2 = accuracy_score(test_Y,pred_Y_svm2)
loss_Y_svm2 = svm2.predict(train_X)
loss_acc_svm2 = accuracy_score(train_Y,loss_Y_svm2)
print(pred_acc_svm2, loss_acc_svm2)
```

```
C:\Users\sweet\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:605: Warning: The least populated class in y has
only 1 members, which is too few. The minimum number of members in any class cannot be less than n_splits=3.
  % (min_groups, self.n_splits)), Warning)
```

```
0.6470588235294118 0.5154639175257731
```

```
#Logistic Regression
logist = LogisticRegression()
logist.fit(train_X, train_Y)
pred_Y_logist = logist.predict(test_X)
pred_acc_logist = accuracy_score(test_Y,pred_Y_logist)
loss_Y_logist = logist.predict(train_X)
loss_acc_logist = accuracy_score(train_Y,loss_Y_logist)
print(pred_acc_logist, loss_acc_logist)
```

```
0.7058823529411765 0.563573883161512
```

```
#隨機森林
forest = RandomForestClassifier(n_estimators=500) #可以挑幾棵樹 100~500
forest.fit(train_X,train_Y)
pred_Y_forest = forest.predict(test_X)
pred_acc_forest = accuracy_score(test_Y,pred_Y_forest)
loss_Y_forest = logist.predict(train_X)
loss_acc_forest = accuracy_score(train_Y,loss_Y_forest)
print(pred_acc_forest, loss_acc_forest)
```

```
0.7647058823529411 0.563573883161512
```

```
# KNN
knn = KNeighborsClassifier(n_neighbors=3) #可以挑幾棵樹 100~500
knn.fit(train_X,train_Y)
pred_Y_knn = knn.predict(test_X)
pred_acc_knn = accuracy_score(test_Y,pred_Y_knn)
loss_Y_knn = knn.predict(train_X)
loss_acc_knn = accuracy_score(train_Y,loss_Y_knn)
print(pred_acc_knn, loss_acc_knn)
```

```
0.5294117647058824 0.7353951890034365
```

```
pred_res=[pred_acc_tree,pred_acc_bayes,pred_acc_svm1,pred_acc_svm2,pred_acc_logist,pred_acc_forest,pred_acc_knn]
loss_res=[loss_acc_tree,loss_acc_bayes,loss_acc_svm1,loss_acc_svm2,loss_acc_logist,loss_acc_forest,loss_acc_knn]
print(pred_res)
print(loss_res)
```

```
[0.7058823529411765, 0.6470588235294118, 0.6470588235294118, 0.6470588235294118, 0.7058823529411765, 0.7647058823529411, 0.529
4117647058824]
[0.8384879725085911, 0.5979381443298969, 0.5189003436426117, 0.5154639175257731, 0.563573883161512, 0.563573883161512, 0.73539
51890034365]
```