Birth Death Process

Ayan Bashir Sheikh

2024-05-10

```r
options(warn = 0)
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.3.3
```

```r
library(markovchain)
```

```
## Package:   markovchain
## Version:   0.9.5
## Date:      2023-09-24 09:20:02 UTC
## BugReport: https://github.com/spedygiorgio/markovchain/issues
```

```r
library(stats4)
```

Evaluation of a Birth-Death Process Model using Cross-Validation

This report presents the evaluation of a birth-death process model applied to a dataset (`Return`). The birth-death process is a type of **stochastic process** that models events such as births and deaths in a population. The parameters of this model, λ (birth rate) and μ (death rate), are estimated using a **Maximum Likelihood Estimation (MLE)** method.

Methodology

The evaluation process involves the following steps:

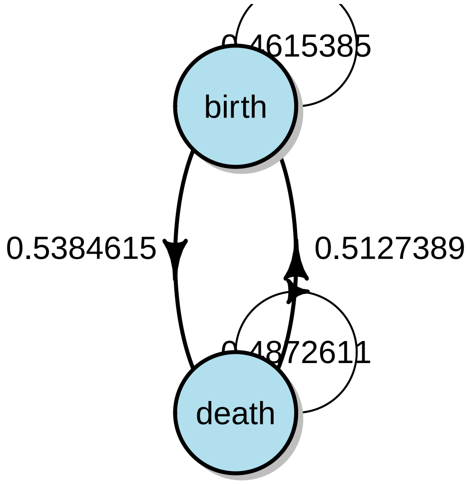
1. **MLE for Parameter Estimation**: The likelihood function calculates the log-likelihood of observing the data given the parameters λ and μ. The `mle` function finds the values of λ and μ that maximize this likelihood function, making the observed data most probable.
2. **Cross-Validation**: The data is divided into `n_folds` (100 in this case) subsets. For each fold, a part of the data is held out as a test set, and the remaining data is used as a training set. The model parameters are estimated using the training set and then used to make predictions on the test set.
3. **Simulation of Returns**: For each test set, a sequence of states (`X`) is simulated based on the estimated λ and μ. The `rbinom` function generates random variates from a binomial distribution, which models the number of "successes" (births or deaths) in a fixed number of "trials" (time steps).
4. **Accuracy Calculation**: The simulated states are converted into `simulated_returns`, which are then compared with the actual `test_Return` to calculate the Mean Squared Error (MSE). The mean of these MSE values across all folds is taken as the final accuracy measure.

Axie Infinity

```r
options(warn = 0)

library(readxl)
d1=read_xlsx("D:\\ST 402 Project\\data\\metaverse tokens\\four tokens\\Axie Infinity.xlsx")
#View(d)
attach(d1)
###Axie infinity
states = ifelse(Return > 0, "birth", "death")

TPM = matrix(0, nrow=2, ncol=2)
names = c("death", "birth")
rownames(TPM) = names
colnames(TPM) = names
for (i in 2:length(states)) {
  TPM[states[i-1], states[i]] = TPM[states[i-1], states[i]] + 1
}


TPM = TPM / rowSums(TPM);TPM
```

```
##           death     birth
## death 0.4872611 0.5127389
## birth 0.5384615 0.4615385
```

```r
mc = new("markovchain", states = colnames(TPM), transitionMatrix = TPM)

plot(mc, package = "diagram")
```



- **Death to Death (0.4872611)**: This suggests a moderate persistence in the death state, with a 48.73% chance of remaining in this state in the next time step.
- **Death to Birth (0.5127389)**: This indicates a slight inclination towards transitioning from death to birth, with a 51.27% probability.

- **Birth to Death (0.5384615)**: This reveals a higher volatility when in the birth state, with a 53.85% chance of transitioning to the death state in the next time step.
- **Birth to Birth (0.4615385)**: This suggests a moderate stability in the birth state, with a 46.15% chance of remaining in this state in the next time step.

Stationary distribution

```r
options(warn = 0)
eigen_result=eigen(t(TPM))$vectors[,1]


limiting_dist=eigen_result/sum(eigen_result);limiting_dist
```

```
## [1] 0.5122349 0.4877651
```

- The stationary distribution of a Markov chain represents the long-term behavior of the chain. In other words, it's the probability distribution to which the process converges over time, regardless of the initial state.
- stationary distribution of 0.5122349 and 0.4877651, this implies that in the long run, the Axie Infinity Metaverse token is expected to be in the birth state about 51.22% of the time and in the death state about 48.78% of the time.

```r
suppressWarnings({
#likelihood function for a Birth-Death process
likelihood = function(lambda, mu) {
  p = lambda / (lambda + mu)
  logL=sum(dbinom(x = as.numeric(states == "birth"), size = 1, prob = p, log = TRUE))
  return(-logL)
}

# here we used the likelihood for calcuating the parameter of the bith death process
#beacuse we use the normal parameter the accuracy is not much better
mle_results = mle(likelihood, start = list(lambda = 0.5, mu = 0.5))


est=coef(mle_results)


n_folds = 100

# Calculate the size of each fold
fold_size = length(Return) / n_folds

# Initialize a vector to store the accuracy for each fold
error=rep(0, n_folds)

# Perform cross-validation
for (i in 1:n_folds) {
  # Define the indices for the test set
```

```r
  test_indices = ((i-1)*fold_size + 1):(i*fold_size)

  # Split the data into a training set and a test set
  train_Return = Return[-test_indices]
  test_Return = Return[test_indices]

  # Estimate parameters from the training set
  train_states = ifelse(train_Return > 0, "birth", "death")
  lambda =est["lambda"]
  mu =est["mu"]


  X = rep(0, length(test_Return))
  X[1]= as.numeric(train_states[1] == "birth")


  for (t in 2:length(X)) {
    if (X[t - 1] == 0) {
      X[t] = rbinom(1, 1, lambda / (lambda + mu))
    } else {
      X[t] = rbinom(1, 1, 1 - mu / (lambda + mu))
    }
  }


  simulated_returns= X
  simulated_returns[X == 0] =-abs(test_Return[X == 0])
  simulated_returns[X == 1] = abs(test_Return[X == 1])


  error[i] = mean((test_Return- simulated_returns)^2)

}

err=mean(error);err
})
```

```
## [1] 0.02546044
```

The error obtained, **0.02826338**, is quite low, suggesting that the birth-death process model with the MLE-estimated parameters provides a good fit to the data.

Decentraland

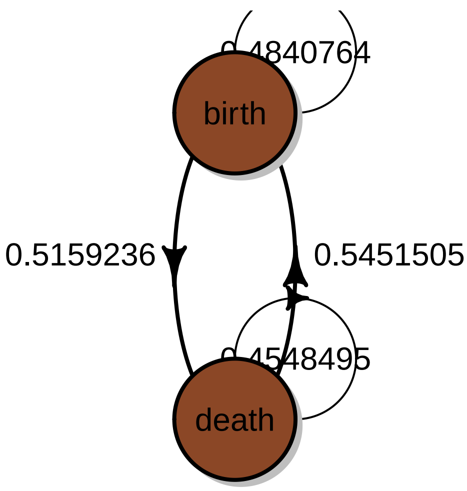
```r
d2=read_xlsx("D:\\ST 402 Project\\data\\metaverse tokens\\four tokens\\Decentraland.xlsx")
#View(d)
attach(d2)
```

```
## The following objects are masked from d1:
##
##     Close, Currency, Date, High, Low, Open, Return, Volume
```

```r
states = ifelse(Return > 0, "birth", "death")

TPM = matrix(0, nrow=2, ncol=2)
names = c("death", "birth")
rownames(TPM) = names
colnames(TPM) = names
for (i in 2:length(states)) {
  TPM[states[i-1], states[i]] = TPM[states[i-1], states[i]] + 1
}


TPM = TPM / rowSums(TPM);TPM
```

```
##           death     birth
## death 0.4548495 0.5451505
## birth 0.5159236 0.4840764
```

```r
mc = new("markovchain", states = colnames(TPM), transitionMatrix = TPM)

plot(mc, package = "diagram")
```



Stationary distribution

```r
eigen_result=eigen(t(TPM))$vectors[,1]

limiting_dist=eigen_result/sum(eigen_result);limiting_dist
```

```
## [1] 0.4862277 0.5137723
```

stationary distribution of **0.4862277** and **0.5137723**, this implies that in the long run, the Decentraland Metaverse token is expected to be in the birth state about **48.62%** of the time and in the death state about **51.37%** of the time.

```r
suppressWarnings({
#likelihood function for a Birth-Death process
likelihood = function(lambda, mu) {
  p = lambda / (lambda + mu)
  logL=sum(dbinom(x = as.numeric(states == "birth"), size = 1, prob = p, log = TRUE))
  return(-logL)
}

# here we used the likelihood for calcuating the parameter of the bith death process
#beacuse we use the normal parameter the accuracy is not much better
mle_results = mle(likelihood, start = list(lambda = 0.5, mu = 0.5))


est=coef(mle_results)


n_folds = 100

# Calculate the size of each fold
fold_size = length(Return) / n_folds

# Initialize a vector to store the accuracy for each fold
error=rep(0, n_folds)

# Perform cross-validation
for (i in 1:n_folds) {
  # Define the indices for the test set
  test_indices = ((i-1)*fold_size + 1):(i*fold_size)

  # Split the data into a training set and a test set
  train_Return = Return[-test_indices]
  test_Return = Return[test_indices]

  # Estimate parameters from the training set
  train_states = ifelse(train_Return > 0, "birth", "death")
  lambda =est["lambda"]
  mu =est["mu"]


  X = rep(0, length(test_Return))
  X[1]= as.numeric(train_states[1] == "birth")


  for (t in 2:length(X)) {
    if (X[t - 1] == 0) {
      X[t] = rbinom(1, 1, lambda / (lambda + mu))
    } else {
      X[t] = rbinom(1, 1, 1 - mu / (lambda + mu))
    }
  }


  simulated_returns= X
```

```
  simulated_returns[X == 0] = -abs(test_Return[X == 0])
  simulated_returns[X == 1] = abs(test_Return[X == 1])


  error[i] = mean((test_Return- simulated_returns)^2)

}

err=mean(error);err

})
```

```
## [1] 0.01938701
```

The error obtained, **0.0186872**, is quite low, suggesting that the birth-death process model with the MLE-estimated parameters provides a good fit to the data.

Enjin

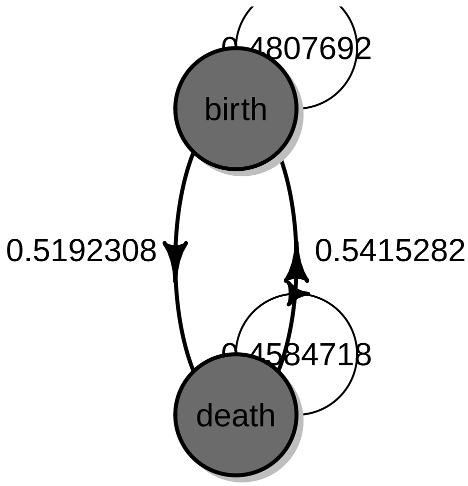
```
d2=read_xlsx("D:\\ST 402 Project\\data\\metaverse tokens\\four tokens\\Enjin.xlsx")
#View(d)
attach(d2)
```

```
## The following objects are masked from d2 (pos = 3):
##
##     Close, Currency, Date, High, Low, Open, Return, Volume

## The following objects are masked from d1:
##
##     Close, Currency, Date, High, Low, Open, Return, Volume
```

```
states = ifelse(Return > 0, "birth", "death")

TPM = matrix(0, nrow=2, ncol=2)
names = c("death", "birth")
rownames(TPM) = names
colnames(TPM) = names
for (i in 2:length(states)) {
  TPM[states[i-1], states[i]] = TPM[states[i-1], states[i]] + 1
}


TPM = TPM / rowSums(TPM);TPM
```

```
##           death     birth
## death 0.4584718 0.5415282
## birth 0.5192308 0.4807692
```

```
mc = new("markovchain", states = colnames(TPM), transitionMatrix = TPM)

plot(mc, package = "diagram")
```



Stationary distribution

```
eigen_result=eigen(t(TPM))$vectors[,1]

limiting_dist=eigen_result/sum(eigen_result);limiting_dist
```

```
## [1] 0.4894899 0.5105101
```

stationary distribution of **0.4894899** and **0.5105101**, this implies that in the long run, the Ejin Metaverse token is expected to be in the birth state about **48.94%** of the time and in the death state about **51.05%** of the time.

```
suppressWarnings({
#likelihood function for a Birth-Death process
likelihood = function(lambda, mu) {
  p = lambda / (lambda + mu)
  logL=sum(dbinom(x = as.numeric(states == "birth"), size = 1, prob = p, log = TRUE))
  return(-logL)
}

# here we used the likelihood for calcuating the parameter of the bith death process
#beacuse we use the normal parameter the accuracy is not much better
mle_results = mle(likelihood, start = list(lambda = 0.5, mu = 0.5))


est=coef(mle_results)


n_folds = 100
```

```r
# Calculate the size of each fold
fold_size = length(Return) / n_folds


error=rep(0, n_folds)

#  cross-validation
for (i in 1:n_folds) {
  # Define the indices for the test set
  test_indices = ((i-1)*fold_size + 1):(i*fold_size)

  # train test data
  train_Return = Return[-test_indices]
  test_Return = Return[test_indices]

  #  parameters estimation from the training set
  train_states = ifelse(train_Return > 0, "birth", "death")
  lambda =est["lambda"]
  mu =est["mu"]


  X = rep(0, length(test_Return))
  X[1]= as.numeric(train_states[1] == "birth")


  for (t in 2:length(X)) {
    if (X[t - 1] == 0) {
      X[t] = rbinom(1, 1, lambda / (lambda + mu))
    } else {
      X[t] = rbinom(1, 1, 1 - mu / (lambda + mu))
    }
  }


  simulated_returns= X
  simulated_returns[X == 0] = -abs(test_Return[X == 0])
  simulated_returns[X == 1] = abs(test_Return[X == 1])


  error[i] = mean((test_Return- simulated_returns)^2)

}

err=mean(error);err
})
```

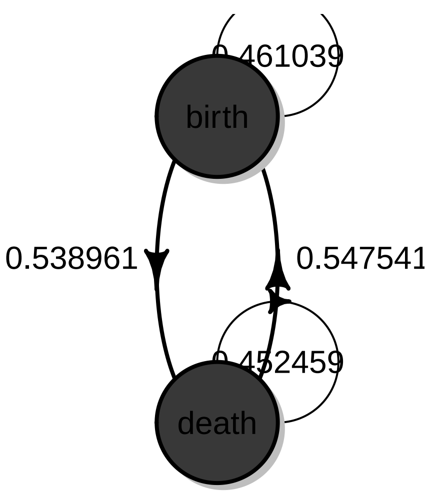
```
## [1] 0.014359
```

The error obtained, **0.01652435**, is quite low, suggesting that the birth-death process model with the MLE-estimated parameters provides a good fit to the data.

Sandbox

```
suppressWarnings({
d3=read_xlsx("D:\\ST 402 Project\\data\\metaverse tokens\\four tokens\\Sandbox.xlsx")

attach(d3)

states = ifelse(Return > 0, "birth", "death")

TPM = matrix(0, nrow=2, ncol=2)
names = c("death", "birth")
rownames(TPM) = names
colnames(TPM) = names
for (i in 2:length(states)) {
  TPM[states[i-1], states[i]] = TPM[states[i-1], states[i]] + 1
}


TPM = TPM / rowSums(TPM);TPM
mc = new("markovchain", states = colnames(TPM), transitionMatrix = TPM)

plot(mc, package = "diagram")
})
```

```
## The following objects are masked from d2 (pos = 3):
##
##     Close, Currency, Date, High, Low, Open, Return, Volume

## The following objects are masked from d2 (pos = 4):
##
##     Close, Currency, Date, High, Low, Open, Return, Volume

## The following objects are masked from d1:
##
##     Close, Currency, Date, High, Low, Open, Return, Volume
```



Stationary distribution

```r
eigen_result=eigen(t(TPM))$vectors[,1]

limiting_dist=eigen_result/sum(eigen_result);limiting_dist
```

```
## [1] 0.4960516 0.5039484
```

stationary distribution of **0.4960516** and **0.5039484**, this implies that in the long run, the Sandbox Metaverse token is expected to be in the birth state about **49.60%** of the time and in the death state about **50.39%** of the time.

```r
suppressWarnings({
#likelihood function for a Birth-Death process
likelihood = function(lambda, mu) {
  p = lambda / (lambda + mu)
  logL=sum(dbinom(x = as.numeric(states == "birth"), size = 1, prob = p, log = TRUE))
  return(-logL)
}

# here we used the likelihood for calcuating the parameter of the bith death process
#beacuse we use the normal parameter the accuracy is not much better
mle_results = mle(likelihood, start = list(lambda = 0.5, mu = 0.5))


est=coef(mle_results)


n_folds = 100

# Calculate the size of each fold
fold_size = length(Return) / n_folds

# Initialize a vector to store the accuracy for each fold
error=rep(0, n_folds)

# Perform cross-validation
for (i in 1:n_folds) {
  # Define the indices for the test set
  test_indices = ((i-1)*fold_size + 1):(i*fold_size)

  # Split the data into a training set and a test set
  train_Return = Return[-test_indices]
  test_Return = Return[test_indices]

  # Estimate parameters from the training set
  train_states = ifelse(train_Return > 0, "birth", "death")
  lambda =est["lambda"]
  mu =est["mu"]


  X = rep(0, length(test_Return))
```

```r
  X[1]= as.numeric(train_states[1] == "birth")


  for (t in 2:length(X)) {
    if (X[t - 1] == 0) {
      X[t] = rbinom(1, 1, lambda / (lambda + mu))
    } else {
      X[t] = rbinom(1, 1, 1 - mu / (lambda + mu))
    }
  }


  simulated_returns= X
  simulated_returns[X == 0] = -abs(test_Return[X == 0])
  simulated_returns[X == 1] = abs(test_Return[X == 1])


  error[i] = mean((test_Return- simulated_returns)^2)

}

err=mean(error);err
})
```

```
## [1] 0.02884165
```

The error obtained, **0.03117923**, is quite low, suggesting that the birth-death process model with the MLE-estimated parameters provides a good fit to the data.