Estimating Financial Risk through Monte Carlo Simulation

Modeling Value at Risk (VaR) with Linear Regression Under Normal Distribution Assumption

Outline

- What Are We Getting Into?
- Basic Terms
- Monte Carlo Risk Modeling
- Results / Evaluations

What Are We Getting Into?

- Train a linear regression model on stock data
- Calculate the risk by running the trained model on virtual markets produced by Monte Carlo Simulation
- We will assume normal distribution for features (market factors) and use multivariate normal distribution for the simulation
- Monte Carlo Simulation is massively parellelizable and Spark is very useful for this!

Basic Terms

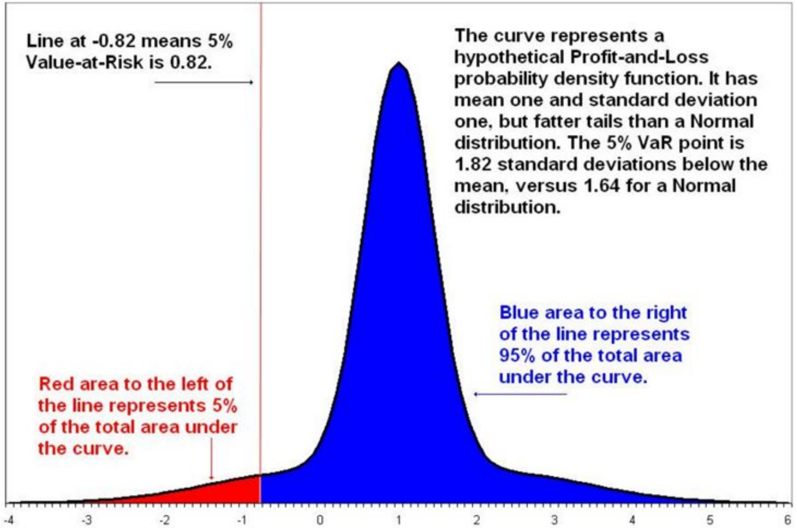
1. Value at Risk (VaR)

A simple measure of investment risk that tries to provide a reasonable estimate of maximum probable loss in value of an investment over the particular period

e.g.) A VaR of 1 mil dollars with a 5% p-value and two weeks -> your investment stands 5% chance of losing more than 1 mil dollars over two weeks

Basic Terms

1. 5% VaR



Basic Terms

1. Conditional Value at Risk (CVaR)

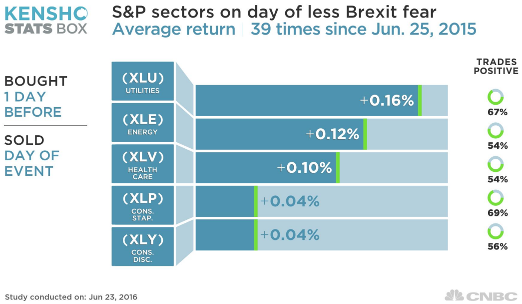
Expected Shortfall (average of VaR values)

e.g.) A CVaR of 5 million dollars with a 5% q-value and two weeks indicates the belief that the average loss in the worst 5% of outcomes is 5 million dollars.

Basic Terms

2. Market Factors

A value that can be used as an indicator of macro aspects of the financial climate at a particular time



Basic Terms

3. Resilient Distributed Datasets (RDDs)

Spark revolves around the concept of a resilient distributed dataset (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel.

Basic Terms

3. Resilient Distributed Datasets (RDDs)

It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Basic Terms

4. Linear Regression

- Try to fit the model with a linear assumption

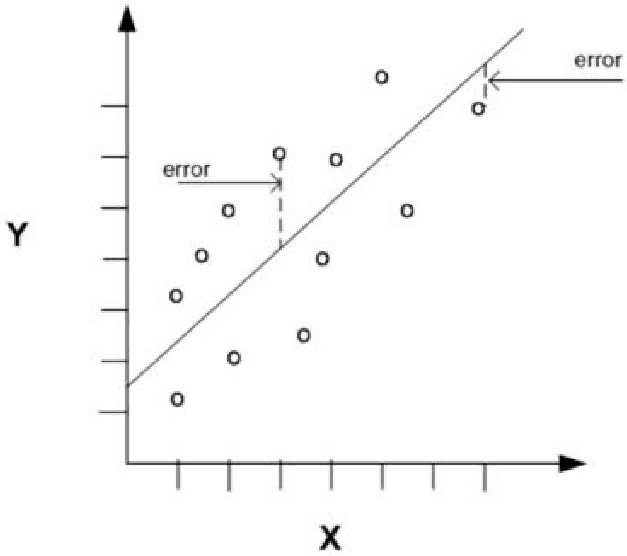
$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$

- Find parameters which minimize errors

$$\text{Find } \min_{\alpha,\, \beta} Q(\alpha, \beta), \qquad \text{for } Q(\alpha, \beta) = \sum_{i=1}^{n} \varepsilon_i^{\,2} = \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2$$

Basic Terms

4. Linear Regression



$$y = \theta_2 x + \theta_1 + e$$

- The **slope** of the line (θ_2) — the angle between a data point and the regression line
- The ***y* intercept** (θ_1) — the point where **x** crosses the *y* axis (**x** = 0)

Basic Terms

5. Monte Carlo Simulation

Monte Carlo simulation performs risk analysis by building models of possible results by substituting a range of values—a probability distribution—for any factor that has inherent uncertainty. It then calculates results over and over, each time using a different set of random values from the probability functions.

Methods for Calculating VaR

1. Variance-Covariance
2. Historical Simulation
3. Monte Carlo Simulation

Monte Carlo Risk Modeling

Our Approach

- Time interval: two weeks
- Model: Linear Regression
- Features (x): four market factors
- Dataset (y): historical data of 3,000 stocks. Returns (change of stock values)
- Objective: Calculate VaR and CVaR of stocks with Monte Carlo Simulation

Dataset

- Stock History Data from Yahoo (GOOGL.csv)

```
Date,Open,High,Low,Close,Volume,Adj Close
2014-10-24,554.98,555.00,545.16,548.90,2175400,548.90
2014-10-23,548.28,557.40,545.50,553.65,2151300,553.65
2014-10-22,541.05,550.76,540.23,542.69,2973700,542.69
2014-10-21,537.27,538.77,530.20,538.03,2459500,538.03
2014-10-20,520.45,533.16,519.14,532.38,2748200,532.38
```

Dataset

- Stock History Data from investing.com (CrudeOil.tsv)

Oct 24, 2014	81.01	81.95	81.95	80.36	272.51K	-1.32%
Oct 23, 2014	82.09	80.42	82.37	80.05	354.84K	1.95%
Oct 22, 2014	80.52	82.55	83.15	80.22	352.22K	-2.39%
Oct 21, 2014	82.49	81.86	83.26	81.57	297.52K	0.71%
Oct 20, 2014	81.91	82.39	82.73	80.78	301.04K	-0.93%
Oct 19, 2014	82.67	82.39	82.72	82.39	-	0.75%

Preprocessing

- Data Point Generation (Two-week interval)

(price on day A - price 14 days later [= 10 rows below]) / (price on day A)

```
def twoWeekReturns(history: Array[(DateTime, Double)]): Array[Double] = {
  history.sliding(10).map { window =>
    val next = window.last._2
    val prev = window.head._2
    (next - prev) / prev
  }.toArray
}
```

Preprocessing

—

- Trimming Data Matrix (no need for details)

Set the start date and the end date for factors/stocks

```
def trimToRegion(history: Array[(DateTime, Double)], start: DateTime, end: DateTime)
: Array[(DateTime, Double)] = {
  var trimmed = history.dropWhile(_._1 < start).takeWhile(_._1 <= end)
  if (trimmed.head._1 != start) {
    trimmed = Array((start, trimmed.head._2)) ++ trimmed
  }
  if (trimmed.last._1 != end) {
    trimmed = trimmed ++ Array((end, trimmed.last._2))
  }
  trimmed
}
```

Preprocessing

- Trimming Data Matrix (no need for details)

Fill in the missing values with the value at the closest date

```scala
def fillInHistory(history: Array[(DateTime, Double)], start: DateTime, end: DateTime)
: Array[(DateTime, Double)] = {
  var cur = history
  val filled = new ArrayBuffer[(DateTime, Double)]()
  var curDate = start
  while (curDate < end) {
    if (cur.tail.nonEmpty && cur.tail.head._1 == curDate) {
      cur = cur.tail
    }

    filled += ((curDate, cur.head._2))

    curDate += 1.days
    // Skip weekends
    if (curDate.dayOfWeek().get > 5) curDate += 2.days
  }
  filled.toArray
}
```

Calculation for Parameters of Linear Regression

A Monte Carlo risk model typically phrases each instrument's return (the change of stock price over a time period) in terms of a set of market factors.

$$r_{it} = c_i + \sum_{j=1}^{|w_i|} w_{ij} * f_{tj}$$

Calculation for Parameters of Linear Regression

Feature Vector with Market Factors

- NASDAQ
- S&P 500
- Crude Oil Price
- US 30-year Treasury Bonds

$$f_t = \phi(m_t)$$

Calculation for Parameters of Linear Regression

Feature vector from the sample code (x: stock value change, sign of the value is preserved)

$$[x^2 \quad \sqrt{x} \quad x]$$

```
def featurize(factorReturns: Array[Double]): Array[Double] = {
  val squaredReturns = factorReturns.map(x => math.signum(x) * x * x)
  val squareRootedReturns = factorReturns.map(x => math.signum(x) * math.sqrt(math.abs(x)))
  squaredReturns ++ squareRootedReturns ++ factorReturns
}
```

Calculation for Parameters of Linear Regression

Linear Regression Model

w: weights for features, f: feature, c: intercept, r: return,

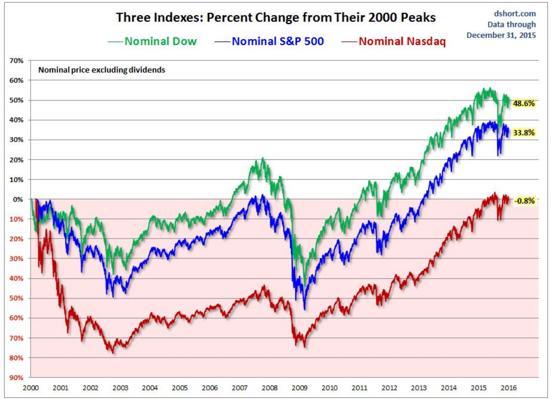
r: return, i: stock, j: feature factor, t: trials

$$r_{it} = c_i + \sum_{j=1}^{|w_i|} w_{ij} * f_{tj}$$

Monte Carlo Simulation

- Calculate Covariance matrix of four market factors

```
val factorMat = factorMatrix(factorsReturns)
val factorCov = new Covariance(factorMat).getCovarianceMatrix().getData()
val factorMeans = factorsReturns.map(factor => factor.sum / factor.size).toArray
```



Closer to the reality! (comparing to independence assumptions)

Monte Carlo Simulation

- Generate samples of market factor values following multivariate normal distribution

```
def trialReturns(
    seed: Long,
    numTrials: Int,
    instruments: Seq[Array[Double]],
    factorMeans: Array[Double],
    factorCovariances: Array[Array[Double]]): Seq[Double] = {
  val rand = new MersenneTwister(seed)
  val multivariateNormal = new MultivariateNormalDistribution(rand, factorMeans,
    factorCovariances)

  val trialReturns = new Array[Double](numTrials)
  for (i <- 0 until numTrials) {
    val trialFactorReturns = multivariateNormal.sample()
    val trialFeatures = RunRisk.featurize(trialFactorReturns)
    trialReturns(i) = trialReturn(trialFeatures, instruments)
  }
  trialReturns
}
```

Parallel Computations with RDDs

- # of trials: 10,000,000
- # of RDDs: 1,000
- Use different seed for Mersenne Twister random generator and feed it to multivariate normal sample for each trial

```
// Generate different seeds so that our simulations don't all end up with the same results
val seeds = (baseSeed until baseSeed + parallelism)
val seedRdd = sc.parallelize(seeds, parallelism)

// Main computation: run simulations and compute aggregate return for each
seedRdd.flatMap(
  trialReturns(_, numTrials / parallelism, bInstruments.value, factorMeans, factorCov))
```

One RDD for One Trial

- One trial simulates one virtual market situation
- Each market situation is simulated by features sampled by multivariate normal distribution of four market factors and the trained Linear Regression model parameters
- For each market situation, we calculate the average of VaRs of all stock prices (increase/decrease)

One RDD for One Trial

```scala
def computeTrialReturns(
    stocksReturns: Seq[Array[Double]],
    factorsReturns: Seq[Array[Double]],
    sc: SparkContext,
    baseSeed: Long,
    numTrials: Int,
    parallelism: Int): RDD[Double] = {
  val factorMat = factorMatrix(factorsReturns)
  val factorCov = new Covariance(factorMat).getCovarianceMatrix().getData()
  val factorMeans = factorsReturns.map(factor => factor.sum / factor.size).toArray
  val factorFeatures = factorMat.map(featurize)
  val factorWeights = computeFactorWeights(stocksReturns, factorFeatures)

  val bInstruments = sc.broadcast(factorWeights)

  // Generate different seeds so that our simulations don't all end up with the same results
  val seeds = (baseSeed until baseSeed + parallelism)
  val seedRdd = sc.parallelize(seeds, parallelism)

  // Main computation: run simulations and compute aggregate return for each
  seedRdd.flatMap(
    trialReturns(_, numTrials / parallelism, bInstruments.value, factorMeans, factorCov))
}
```

One RDD for One Trial

```scala
/**
 * Calculate the full return of the portfolio under particular trial conditions.
 */
def trialReturn(trial: Array[Double], instruments: Seq[Array[Double]]): Double = {
  var totalReturn = 0.0
  for (instrument <- instruments) {
    totalReturn += instrumentTrialReturn(instrument, trial)
  }
  totalReturn / instruments.size
}

/**
 * Calculate the return of a particular instrument under particular trial conditions.
 */
def instrumentTrialReturn(instrument: Array[Double], trial: Array[Double]): Double = {
  var instrumentTrialReturn = instrument(0)
  var i = 0
  while (i < trial.length) {
    instrumentTrialReturn += trial(i) * instrument(i+1)
    i += 1
  }
  instrumentTrialReturn
}
```

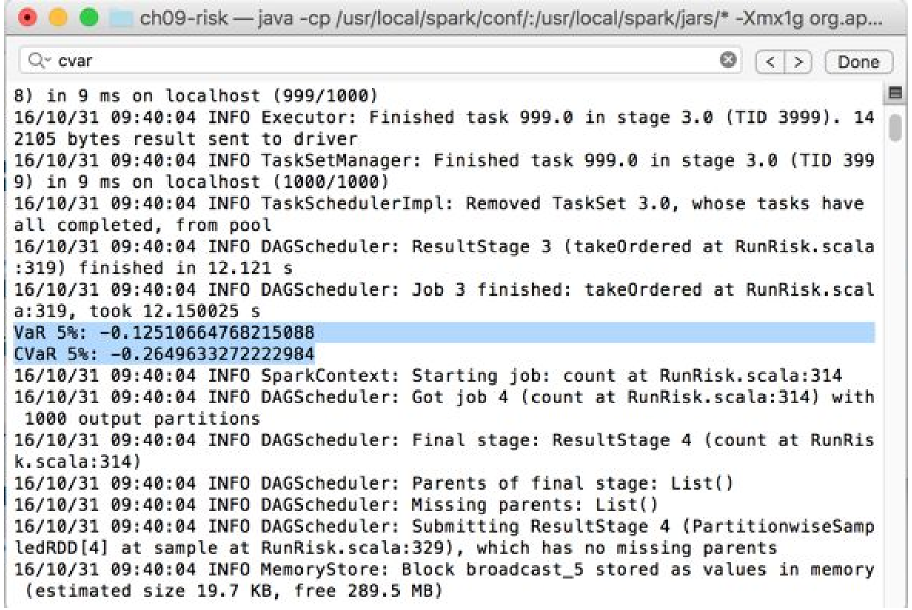
Finally, VaR and CVaR

- Aggregate all trial results

```
def fivePercentVaR(trials: RDD[Double]): Double = {
  val topLosses = trials.takeOrdered(math.max(trials.count().toInt / 20, 1))
  topLosses.last
}

def fivePercentCVaR(trials: RDD[Double]): Double = {
  val topLosses = trials.takeOrdered(math.max(trials.count().toInt / 20, 1))
  topLosses.sum / topLosses.length
}
```

Results & Evaluation



Results & Evaluation

- Confidence Interval (95%)

We are 95% confident to say that the VaR would fall into this interval.

- Bootstrapping

Resample from the subset of VaRs resulted from trials

Results & Evaluation

- Bootstrapped Confidence Interval (95%)

Get the confidence interval from bootstrapped dataset.

```
def bootstrappedConfidenceInterval(
    trials: RDD[Double],
    computeStatistic: RDD[Double] => Double,
    numResamples: Int,
    pValue: Double): (Double, Double) = {
  val stats = (0 until numResamples).map { i =>
    val resample = trials.sample(true, 1.0)
    computeStatistic(resample)
  }.sorted
  val lowerIndex = (numResamples * pValue / 2 - 1).toInt
  val upperIndex = math.ceil(numResamples * (1 - pValue / 2)).toInt
  (stats(lowerIndex), stats(upperIndex))
}
```

Results & Evaluation

- Kupiec’s proportion-of-failures (POF) test

Counts the number of times that the losses exceeded the VaR. The null hypothesis is that the VaR is reasonable, and a sufficiently extreme test statistic means that the VaR estimate does not accurately describe the data.

Results & Evaluation

- Kupiec's proportion-of-failures (POF) test

PortfolioID	VaRID	VaRLevel	POF	LRatioPOF	PValuePOF	Observations	Failures
"Equity"	"Normal95"	0.95	accept	0.46147	0.49694	1043	57
"Equity"	"Normal99"	0.99	reject	3.5118	0.060933	1043	17
"Equity"	"Historical95"	0.95	accept	0.91023	0.34005	1043	59
"Equity"	"Historical99"	0.99	accept	0.22768	0.63325	1043	12
"Equity"	"EWMA95"	0.95	accept	0.91023	0.34005	1043	59
"Equity"	"EWMA99"	0.99	reject	9.8298	0.0017171	1043	22

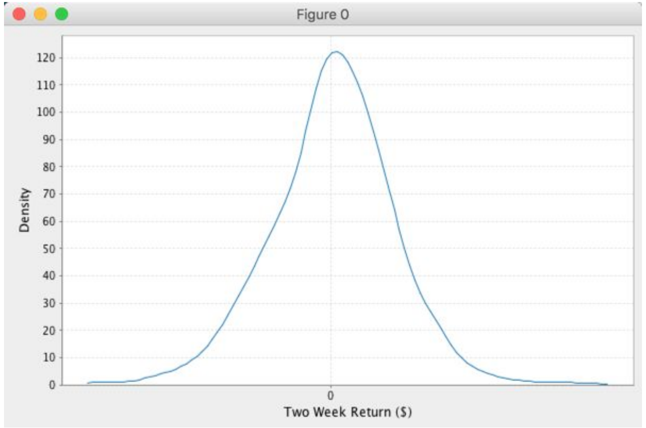
Results & Evaluation

Kupiec test says that this VaR model is not reasonable...

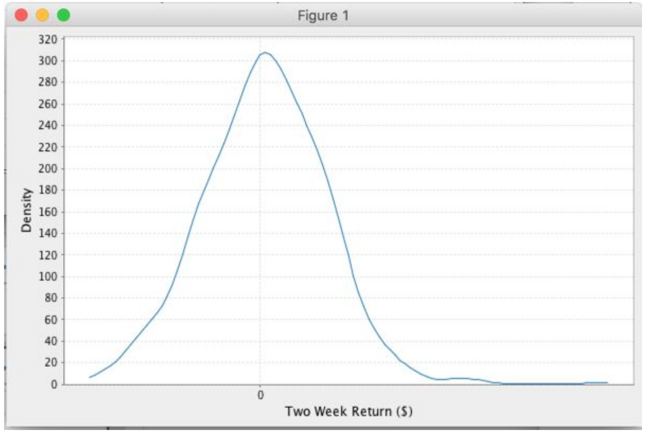
```
ch09-risk — java -cp /usr/local/spark/conf/:/usr/local/spark/jars/* -Xmx1g org....
16/10/31 10:23:26 INFO Executor: Finished task 999.0 in stage 403.0 (TID 403999)
. 109625 bytes result sent to driver
16/10/31 10:23:26 INFO TaskSetManager: Finished task 999.0 in stage 403.0 (TID 4
03999) in 8 ms on localhost (1000/1000)
16/10/31 10:23:26 INFO TaskSchedulerImpl: Removed TaskSet 403.0, whose tasks hav
e all completed, from pool
16/10/31 10:23:26 INFO DAGScheduler: ResultStage 403 (takeOrdered at RunRisk.sca
la:319) finished in 11.257 s
16/10/31 10:23:26 INFO DAGScheduler: Job 403 finished: takeOrdered at RunRisk.sc
ala:319, took 11.262985 s
VaR confidence interval: (-0.12549004584482001,-0.12468834434968425)
CVaR confidence interval: (-0.2655703019054329,-0.26444242296872705)
Kupiec test p-value: 0.0
16/10/31 10:23:26 INFO SparkContext: Starting job: stats at RunRisk.scala:299
16/10/31 10:23:26 INFO DAGScheduler: Got job 404 (stats at RunRisk.scala:299) wi
th 1000 output partitions
16/10/31 10:23:26 INFO DAGScheduler: Final stage: ResultStage 404 (stats at RunR
isk.scala:299)
16/10/31 10:23:26 INFO DAGScheduler: Parents of final stage: List()
16/10/31 10:23:26 INFO DAGScheduler: Missing parents: List()
16/10/31 10:23:26 INFO DAGScheduler: Submitting ResultStage 404 (MapPartitionsRD
D[404] at stats at RunRisk.scala:299), which has no missing parents
16/10/31 10:23:26 INFO MemoryStore: Block broadcast_405 stored as values in memo
ry (estimated size 3.2 KB, free 289.5 MB)
```

Results & Evaluation

Market Factor Distributions



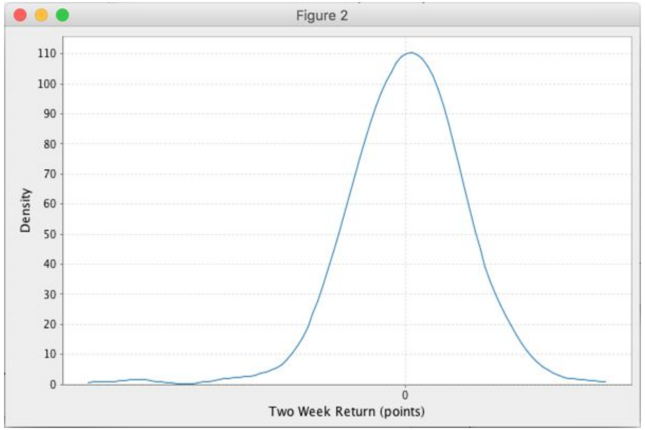
Crude Oil



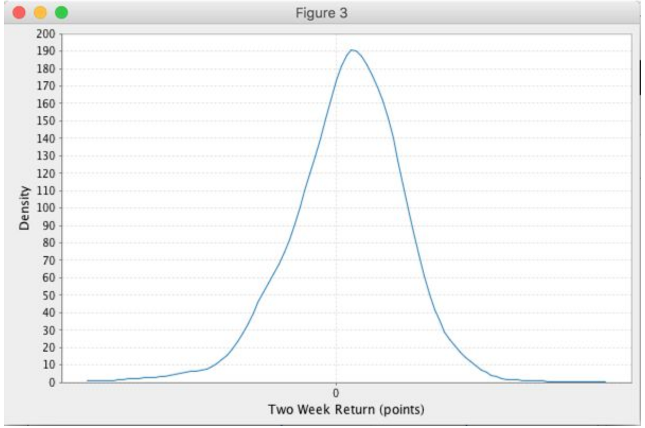
US 30-Year Treasury

Results & Evaluation

Market Factor Distributions



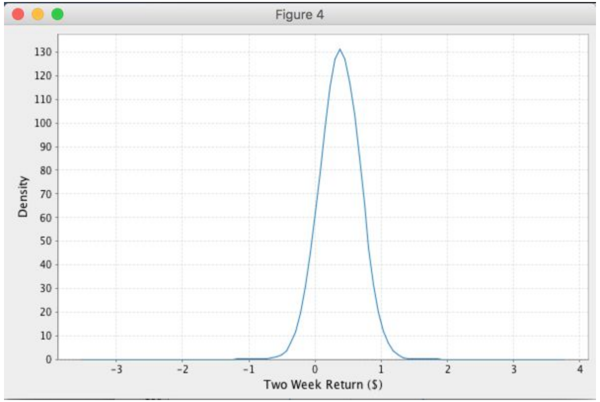
S&P 500



NASDAQ

Results & Evaluation

Monte Carlo Simulation



3,000 stocks

References

http://spark.apache.org/docs/latest/programming-guide.html

https://github.com/sryza/aas

https://www.mathworks.com/help/risk/pof.html

https://en.wikipedia.org/wiki/Linear_regression

http://www.palisade.com/risk/monte_carlo_simulation.asp

Advanced Analytics with Spark: Patterns for Learning from Data at Scale (2015) - Josh Wills, Sandy Ryza, Sean Owen, and Uri Laserson

Image Resources

http://sakiicelimbekardas.blogspot.com/2016/02/stock.html

http://www.cnbc.com/2016/06/23/sp-500-sectors-in-the-brexit-crosshairs.html

http://www.cnbc.com/2015/07/17/5-tech-trades-on-nasdaqs-record-close.html

http://www.investing.com/analysis/the-s-p-500,-dow-and-nasdaq-since-their-2000-highs-378646