Classifier Algorithms

And Swift implementations

> whoami

- Software Engineer at Softserve Poland for 1½ year
- 5 years of experience with mobile development Android & iOS
- 2 years as Data Science researcher at Campinas State University (Brazil)

Agenda

Chapter 1 (25 minutes)

Introduction to AI → ML → Classifiers → Supervised Methods → Trees

Chapter 2 (35 minutes)

Challenges of implementing your algorithms in Swift

Chapter 1

A brief history of ML

1.1 What is Artificial Intelligence?

Has to be this guy, right?

No. It's a general name to label a few Computer Science research fields:

- Natural Language Processing
- Computer Vision
- Machine Learning
- ....

1.2 Then what is Machine Learning?

For sure it's this guy

Algorithms that use large data and statistics to make predictions

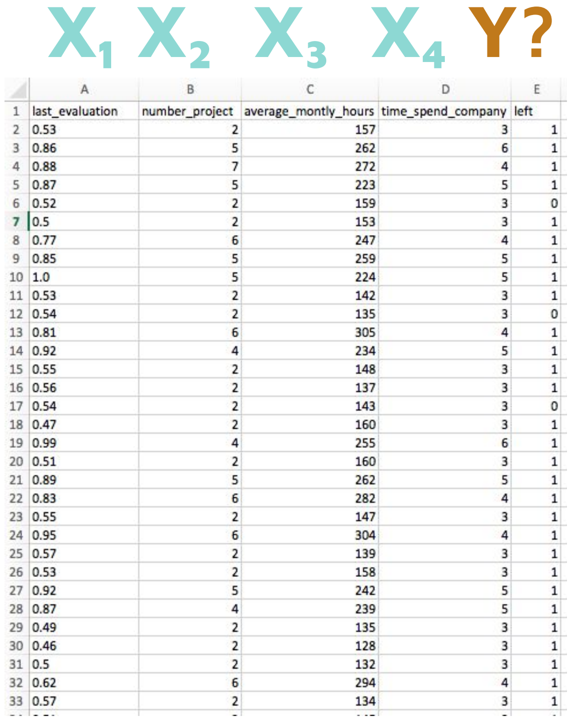
Data + ~Mythical Black box Creature~ = Predictions

1.3 Let's talk about Data

1.3 Let's talk about Data

1. Labeled vs Unlabeled
2. Feature Scaling
3. Sparse
4. Curse of dimensionality

1.3.1 Labeled Data versus Unlabeled data



	A	B	C	D	E
1	last_evaluation	number_project	average_montly_hours	time_spend_company	left
2	0.53	2	157	3	1
3	0.86	5	262	6	1
4	0.88	7	272	4	1
5	0.87	5	223	5	1
6	0.52	2	159	3	0
7	0.5	2	153	3	1
8	0.77	6	247	4	1
9	0.85	5	259	5	1
10	1.0	5	224	5	1
11	0.53	2	142	3	1
12	0.54	2	135	3	0
13	0.81	6	305	4	1
14	0.92	4	234	5	1
15	0.55	2	148	3	1
16	0.56	2	137	3	1
17	0.54	2	143	3	0
18	0.47	2	160	3	1
19	0.99	4	255	6	1
20	0.51	2	160	3	1
21	0.89	5	262	5	1
22	0.83	6	282	4	1
23	0.55	2	147	3	1
24	0.95	6	304	4	1
25	0.57	2	139	3	1
26	0.53	2	158	3	1
27	0.92	5	242	5	1
28	0.87	4	239	5	1
29	0.49	2	135	3	1
30	0.46	2	128	3	1
31	0.5	2	132	3	1
32	0.62	6	294	4	1
33	0.57	2	134	3	1

Labeled = we know Y, so we should use a **supervised algorithm**

Unlabeled = we do not Y, so we should use an **unsupervised algorithm***

*** OR PAY SOMEONE TO MANUALLY LABEL THEM**

1.3.2 Feature scaling

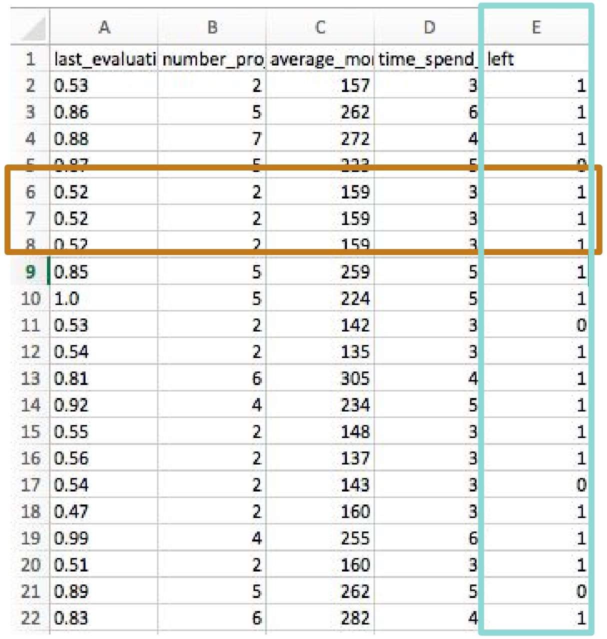
(-124.3, 12512.4) → (-1, 1)

[true, false] → [0,1]

["Apple", "Pear", "Banana"] → [0,1,2]

- Data should be normalized when your algorithm uses calculations with numbers, like **euclidean distance** or **stochastic gradient descent**
- Tree **ensemble** methods, for example, do not need feature scaling
- There are many types of normalization methods, the most common is **min-max**
- **Data types matter** - Unsigned Int, Int8, Int16, Float, Double, ...

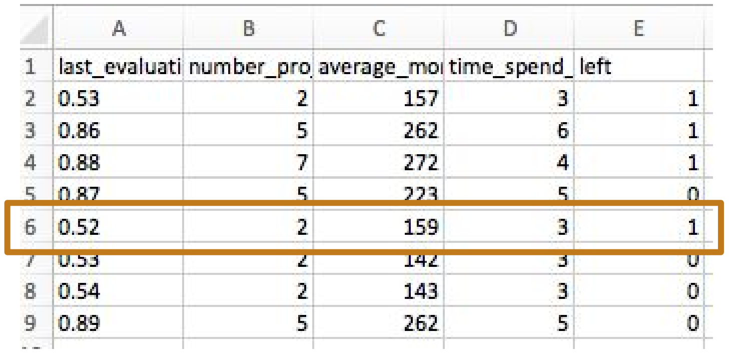
1.3.3 Sparse Data



	A	B	C	D	E
1	last_evaluati	number_pro	average_mo	time_spend_	left
2	0.53	2	157	3	1
3	0.86	5	262	6	1
4	0.88	7	272	4	1
5	0.87	5	223	5	0
6	0.52	2	159	3	1
7	0.52	2	159	3	1
8	0.52	2	159	3	1
9	0.85	5	259	5	1
10	1.0	5	224	5	1
11	0.53	2	142	3	0
12	0.54	2	135	3	1
13	0.81	6	305	4	1
14	0.92	4	234	5	1
15	0.55	2	148	3	1
16	0.56	2	137	3	1
17	0.54	2	143	3	0
18	0.47	2	160	3	1
19	0.99	4	255	6	1
20	0.51	2	160	3	1
21	0.89	5	262	5	0
22	0.83	6	282	4	1

17 1's versus 4 0's

Duplicates removed



	A	B	C	D	E
1	last_evaluati	number_pro	average_mo	time_spend_	left
2	0.53	2	157	3	1
3	0.86	5	262	6	1
4	0.88	7	272	4	1
5	0.87	5	223	5	0
6	0.52	2	159	3	1
7	0.53	2	142	3	0
8	0.54	2	143	3	0
9	0.89	5	262	5	0

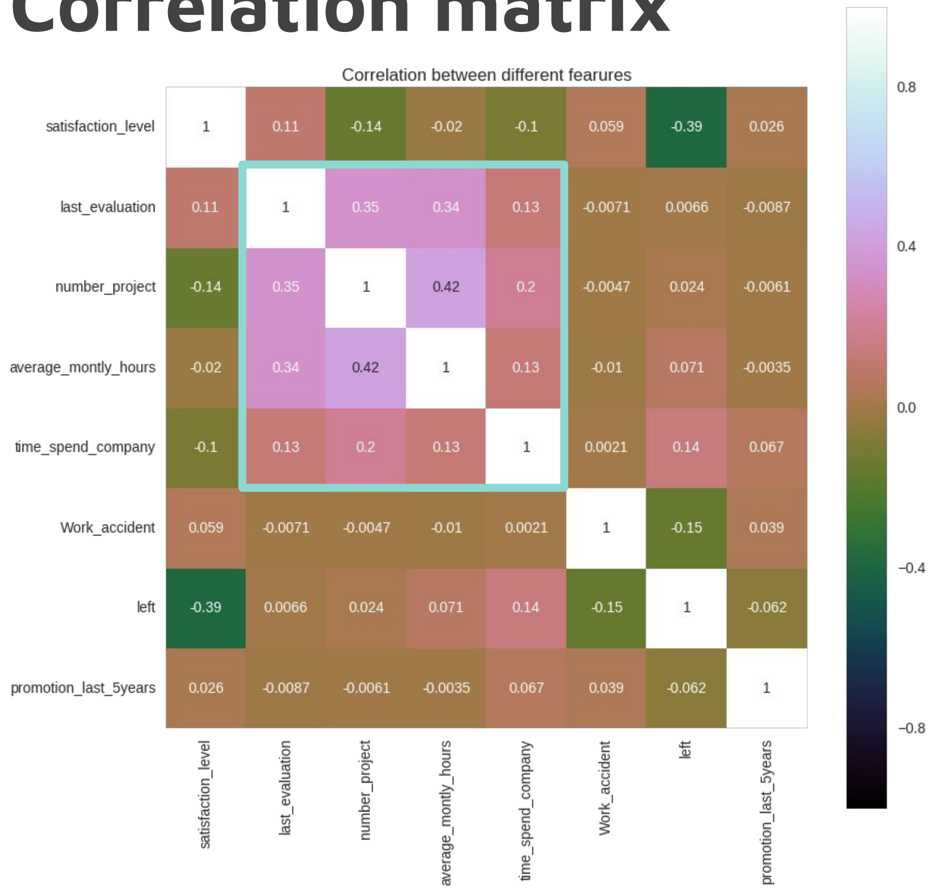
Balanced outputs

*What to do with Missing Data?

1.3.4 Curse of dimensionality

- A lot of dimensions brings Sparsity (quick tip: BAD)
- Examine the correlation of the dimensions against the output, and kick insignificant dimensions
- Principal component analysis
- Feature Engineering

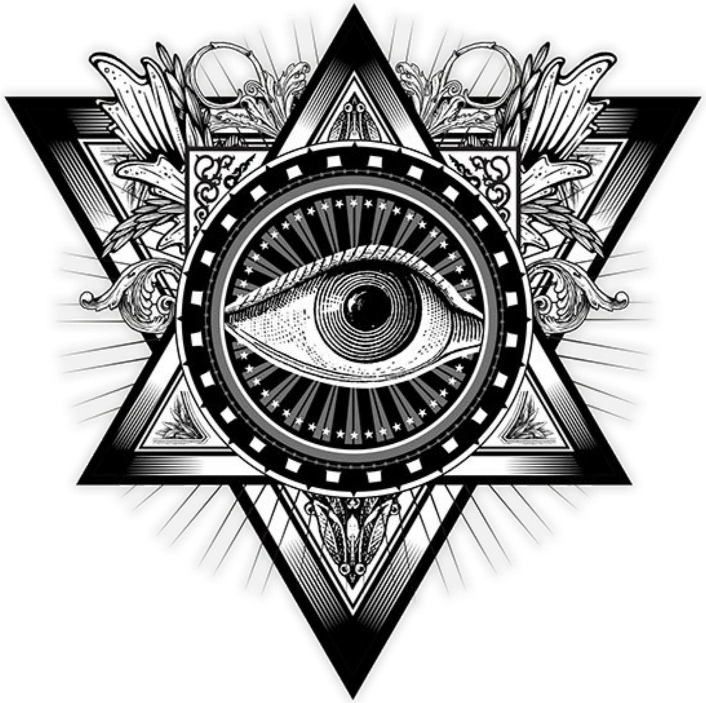
1.3.4.1 Correlation matrix



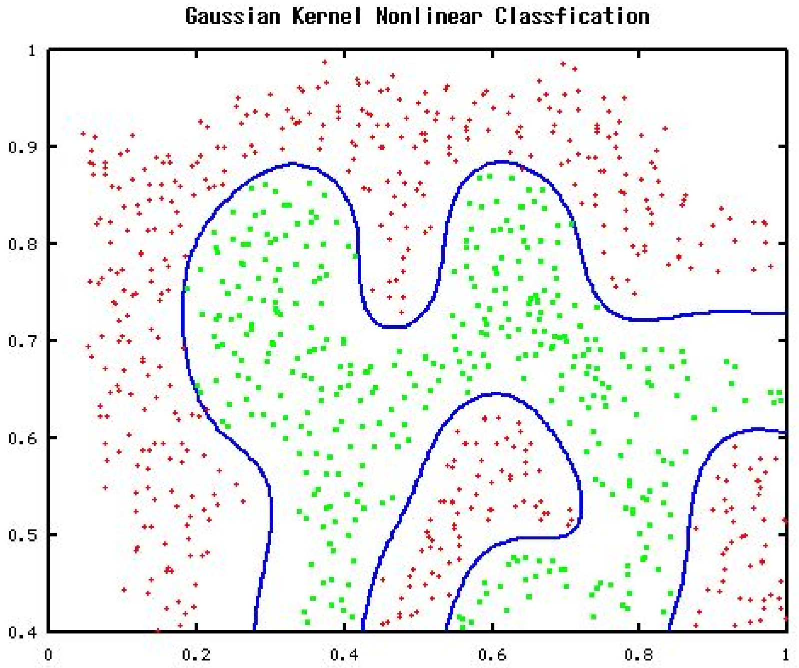
	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years
satisfaction_level	1	0.11	-0.14	-0.02	-0.1	0.059	-0.39	0.026
last_evaluation	0.11	1	0.35	0.34	0.13	-0.0071	0.0066	-0.0087
number_project	-0.14	0.35	1	0.42	0.2	-0.0047	0.024	-0.0061
average_montly_hours	-0.02	0.34	0.42	1	0.13	-0.01	0.071	-0.0035
time_spend_company	-0.1	0.13	0.2	0.13	1	0.0021	0.14	0.067
Work_accident	0.059	-0.0071	-0.0047	-0.01	0.0021	1	-0.15	0.039
left	-0.39	0.0066	0.024	0.071	0.14	-0.15	1	-0.062
promotion_last_5years	0.026	-0.0087	-0.0061	-0.0035	0.067	0.039	-0.062	1

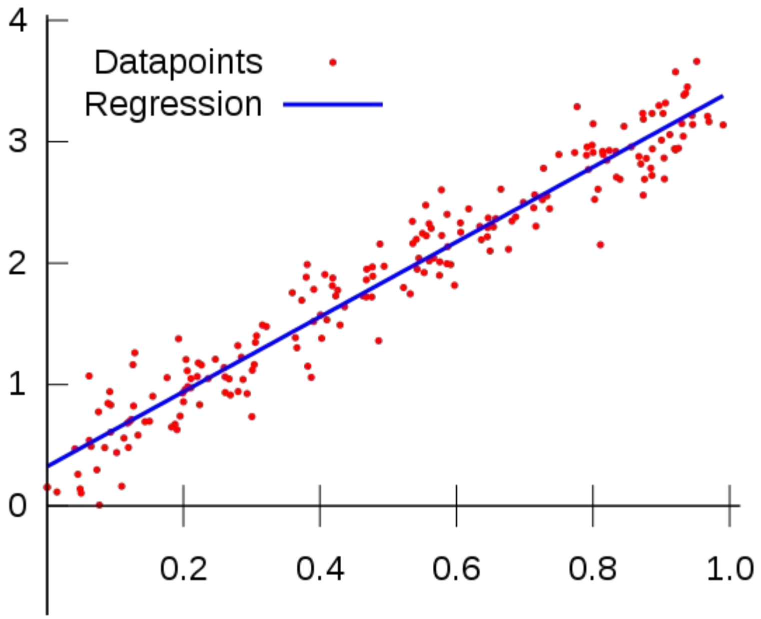
1.4 Let's talk about predictions

1. Classification versus Regression
2. Data splitting
3. Metrics



1.4.1 Classification versus Regression



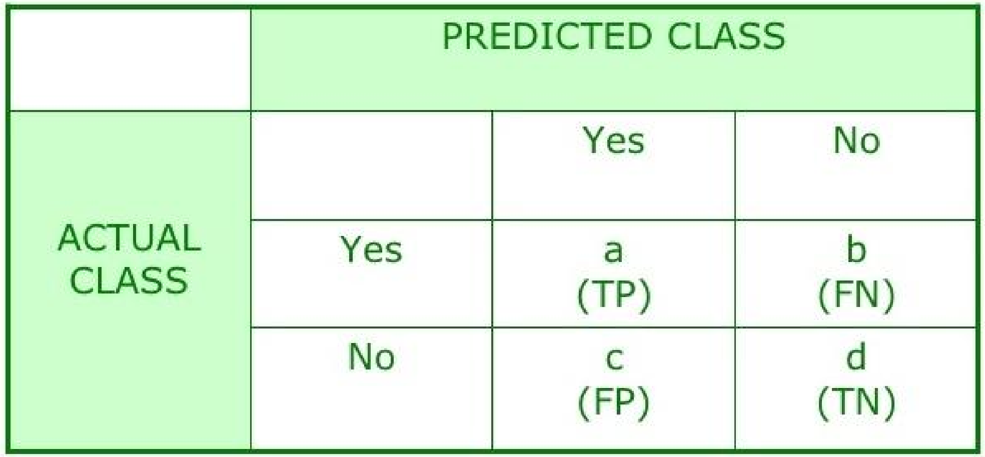


1.4.2 Data Splitting

Training set: use to train your model (~60%)
Test set: use to measure and optimize your models (~20%)
Validation set: don't touch this guy. Ever* (~20%)

"The training set is used to fit the models; the validation set is used to estimate prediction error for model selection; the test set is used for assessment of the generalization error of the final chosen model. Ideally, the test set should be kept in a "vault," and be brought out only at the end of the data analysis." - Elements of Statistical Learning, Travor Hastie

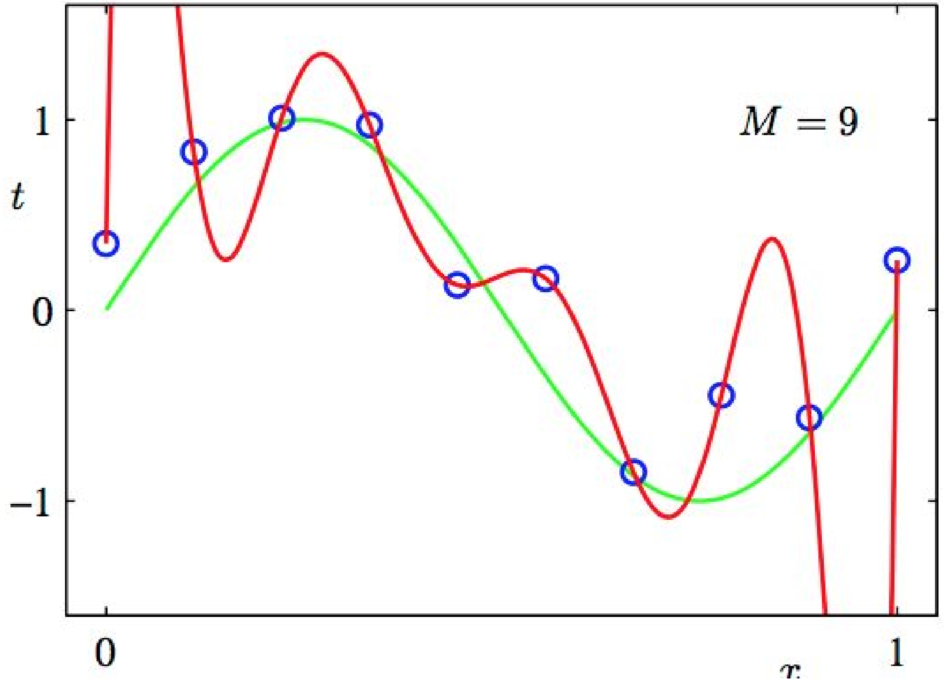
1.4.3 Metrics



		PREDICTED CLASS	
		Yes	No
ACTUAL CLASS	Yes	a (TP)	b (FN)
	No	c (FP)	d (TN)

Ways to measure how good your classifier is.
Accuracy = TP+TN / ALL
Precision, Recall, Specificity, F1, Kapa, ...

1.4.4 Overfitting



1.5 Supervised Classifiers

- Random Forest
- Gradient Boosting Machines
- Naive Bayes
- Support Vector Machines
- …

1.5.1 Why Random Forest?

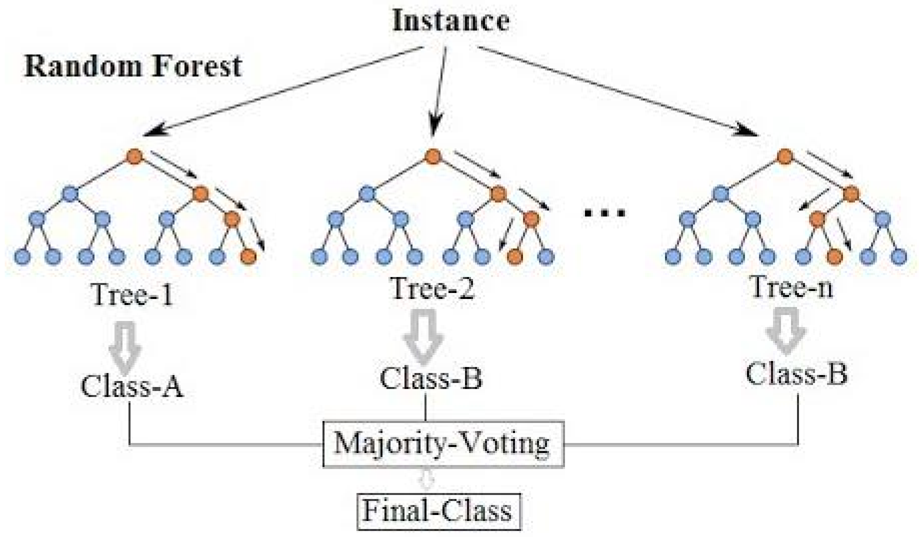
Widely used - Almost half of data mining competition are won by using some variants of tree ensemble methods

Invariant to scaling of inputs, so you do not need to do careful features normalization

Learn higher order interaction between features

Can be scaled, and are used in the industry

1.5.1.1 The algorightm

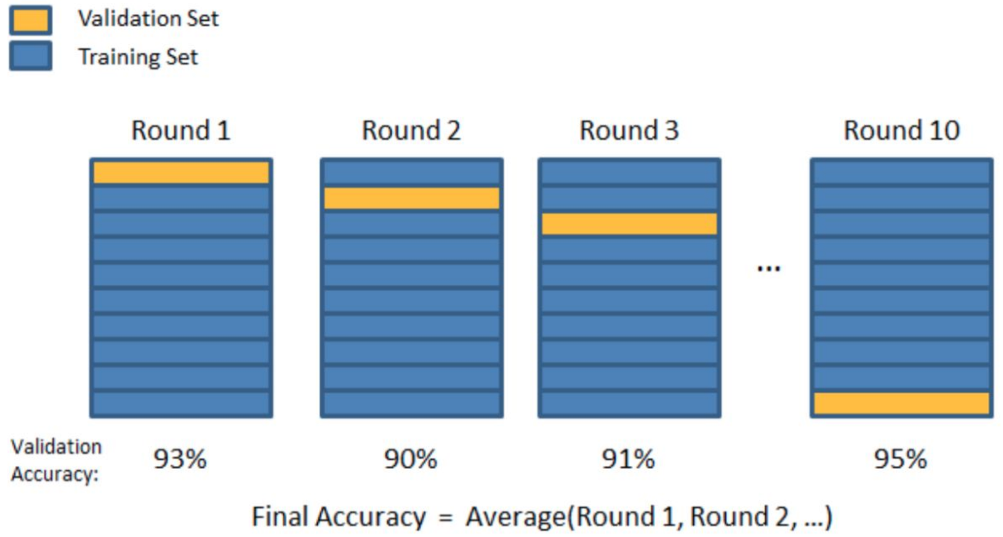


- Not decision trees: each tree handles a random subset of the features (mtry parameter in python, usually sqrt)
- Leaf nodes can have a minimum number of instances
- Features can have different weights
- Each tree is built with a random subset of the database

1.5.2 Cross Validation

In order to validate algorightms, researches usually apply K-fold cross validation.

Validating a random Kth part of the dataset against the trained rest, K times.

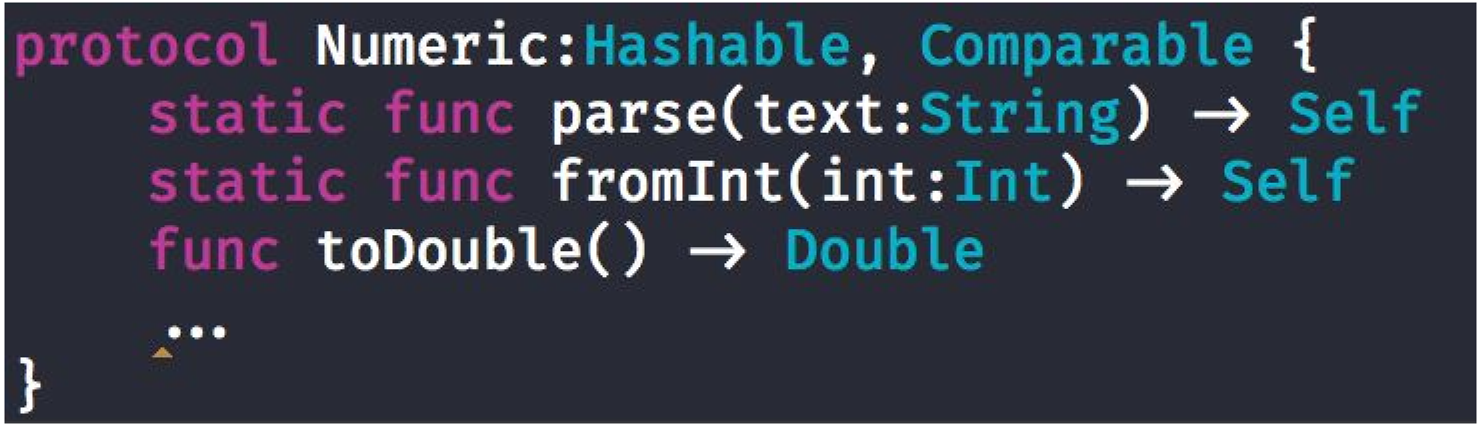


Chapter II:
Swift implementations

2 - Biggest challenges in Swift

- Generics
- Matrixes
- Optional parameters

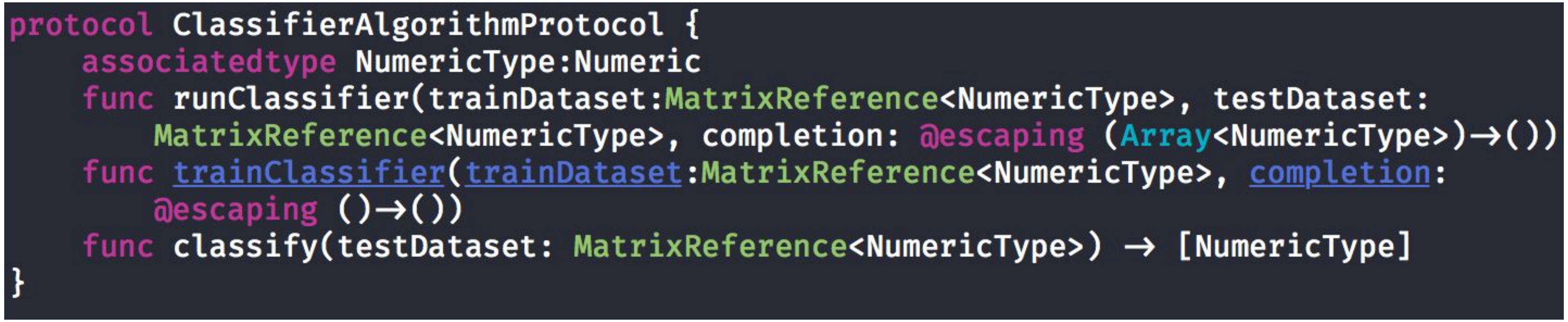
2.1.1 - Having a standard data type



```
protocol Numeric:Hashable, Comparable {
    static func parse(text:String) → Self
    static func fromInt(int:Int) → Self
    func toDouble() → Double
    ...
}
```

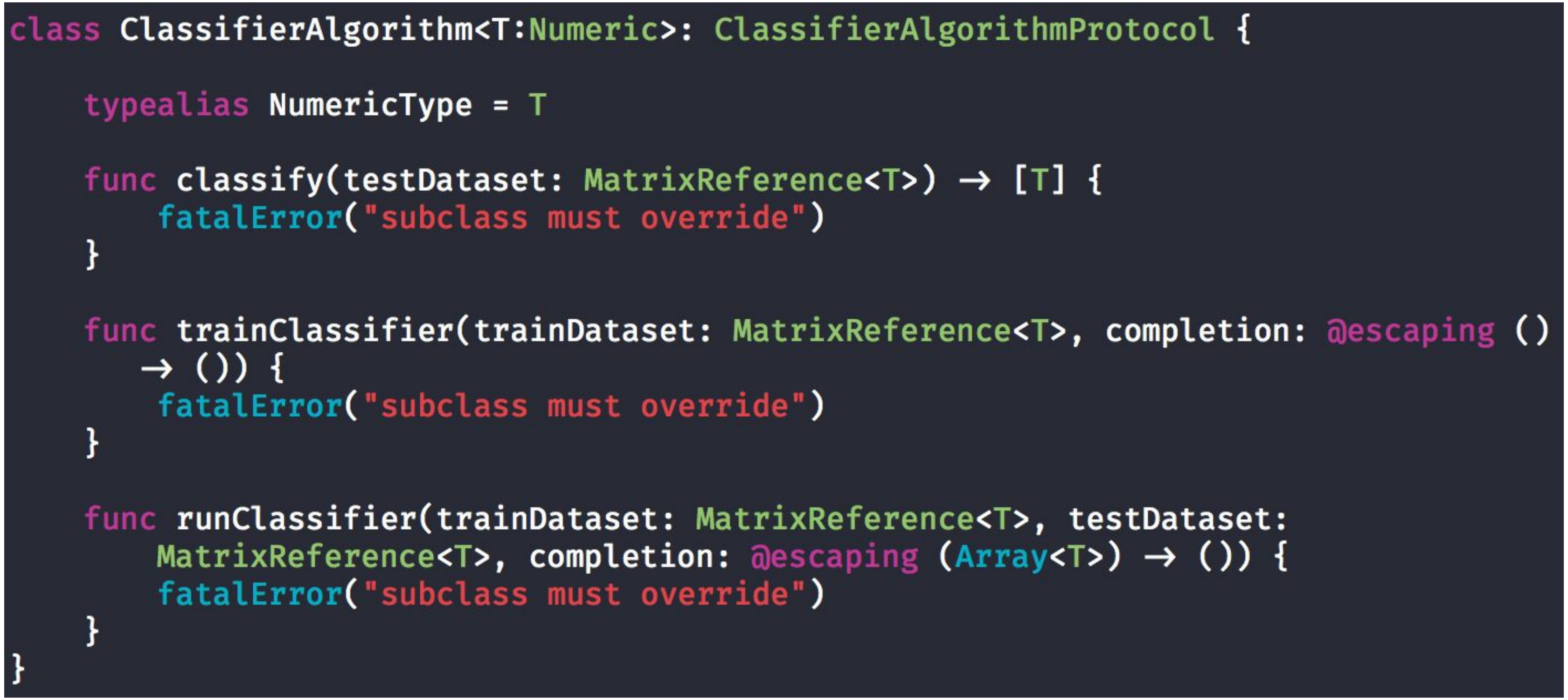
Create extensions of this protocol, for the types you want

2.1.2 - A generic classifier



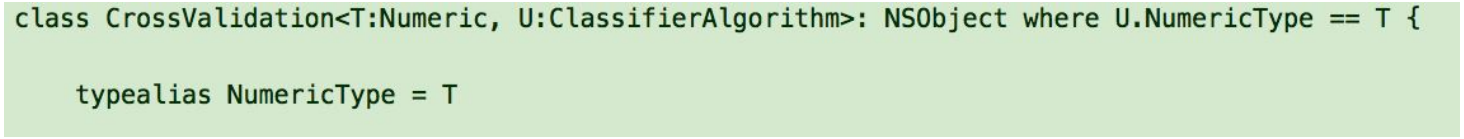
```
protocol ClassifierAlgorithmProtocol {
    associatedtype NumericType:Numeric
    func runClassifier(trainDataset:MatrixReference<NumericType>, testDataset:
        MatrixReference<NumericType>, completion: @escaping (Array<NumericType>)->())
    func trainClassifier(trainDataset:MatrixReference<NumericType>, completion:
        @escaping ()->())
    func classify(testDataset: MatrixReference<NumericType>) -> [NumericType]
}
```

2.1.3 - A concrete implementation (Why?)



```
class ClassifierAlgorithm<T:Numeric>: ClassifierAlgorithmProtocol {

    typealias NumericType = T

    func classify(testDataset: MatrixReference<T>) → [T] {
        fatalError("subclass must override")
    }

    func trainClassifier(trainDataset: MatrixReference<T>, completion: @escaping ()
       → ()) {
        fatalError("subclass must override")
    }

    func runClassifier(trainDataset: MatrixReference<T>, testDataset:
        MatrixReference<T>, completion: @escaping (Array<T>) → ()) {
        fatalError("subclass must override")
    }
}
```

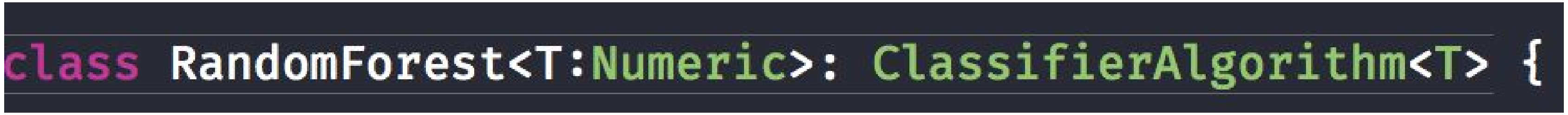
2.1.3.1 Problems of generic types



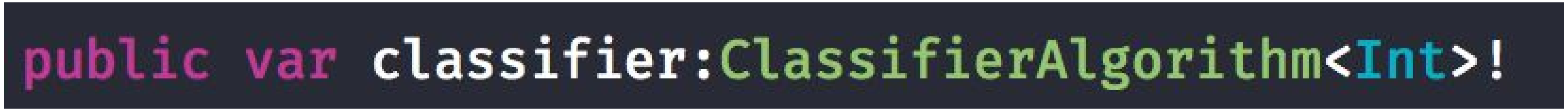
```
class CrossValidation<T:Numeric, U:ClassifierAlgorithm>: NSObject where U.NumericType == T {

    typealias NumericType = T
```

ViewControllers cannot be generic!! They never load from storyboard, and the app crashes

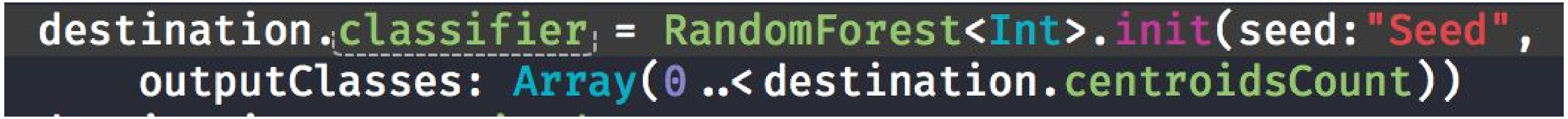
2.1.4 - An algorithm



```
class RandomForest<T:Numeric>: ClassifierAlgorithm<T> {
```



```
public var classifier:ClassifierAlgorithm<Int>!
```



```
destination.classifier = RandomForest<Int>.init(seed:"Seed",
     outputClasses: Array(0 ..< destination.centroidsCount))
```

2.2 - Matrixes

The Swift Array implementation have **super slow** allocation/deallocation

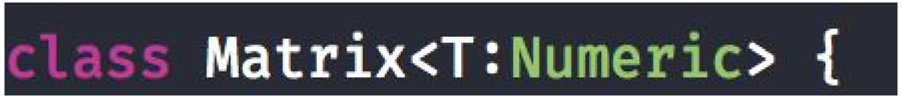
NSArray is faster, but still very slow

C pointers are the fastest, but memory management is up to you.

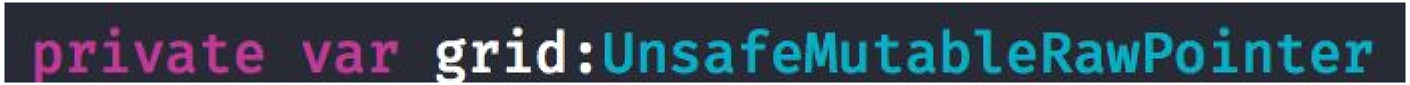
UnsafeMutablePointers do not have most of the C memory functions, like realloc, memcpy, memmove, (...), and will wield unexpected results

UnsafeMutableRawPointers should be the best option. Bonus, they work with Accelerate framework for advanced matrix operations (inverse, transpose, multiply, eigen values, ...)

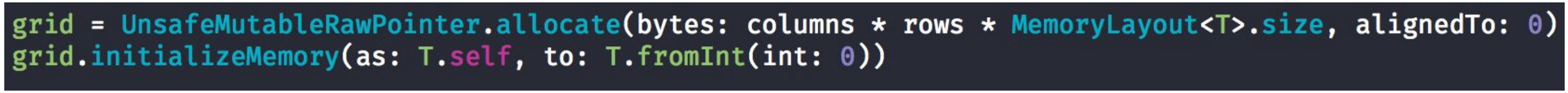
2.2.1 - Pointer examples



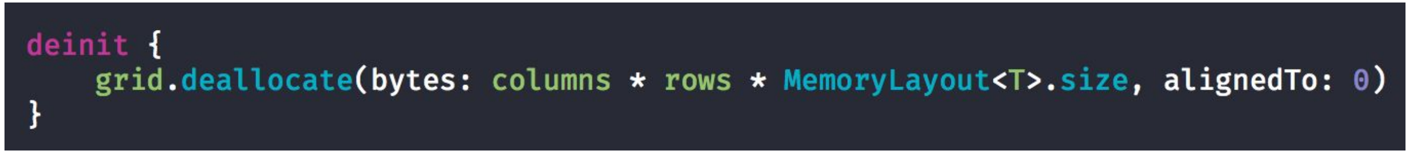
```swift
class Matrix<T:Numeric> {
```



```swift
 private var grid:UnsafeMutableRawPointer
```

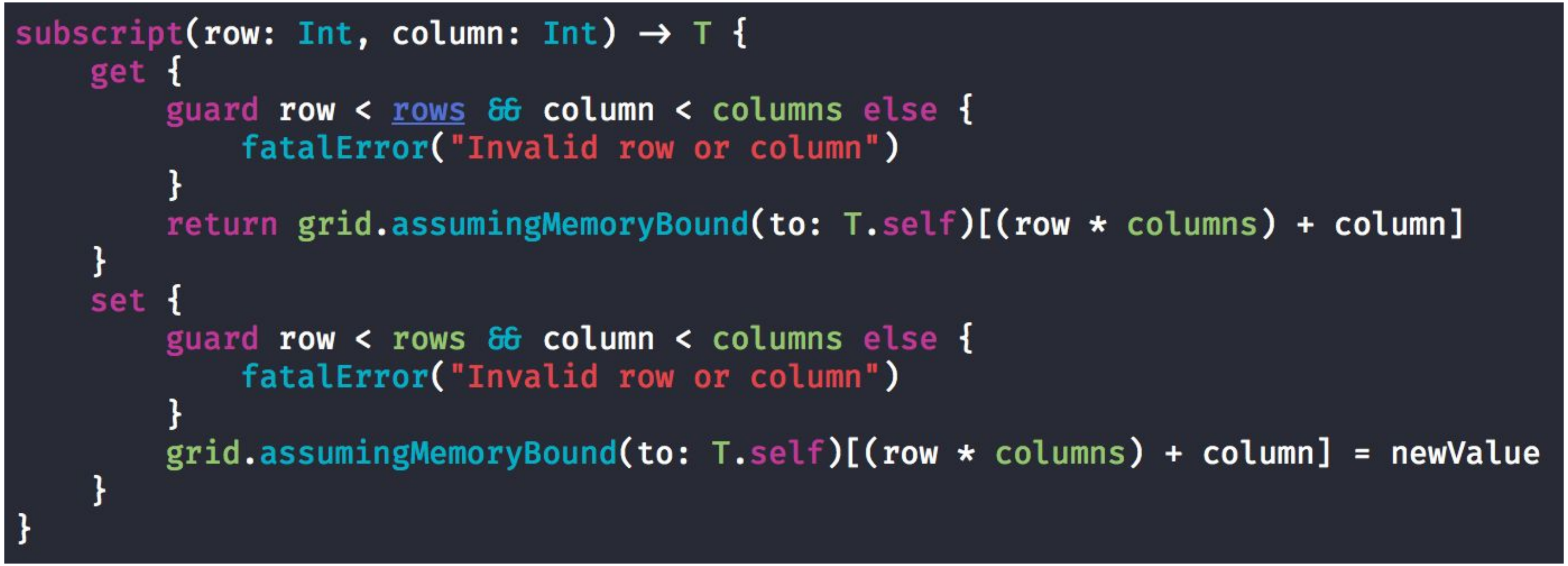


```swift
grid = UnsafeMutableRawPointer.allocate(bytes: columns * rows * MemoryLayout<T>.size, alignedTo: 0)
grid.initializeMemory(as: T.self, to: T.fromInt(int: 0))
```



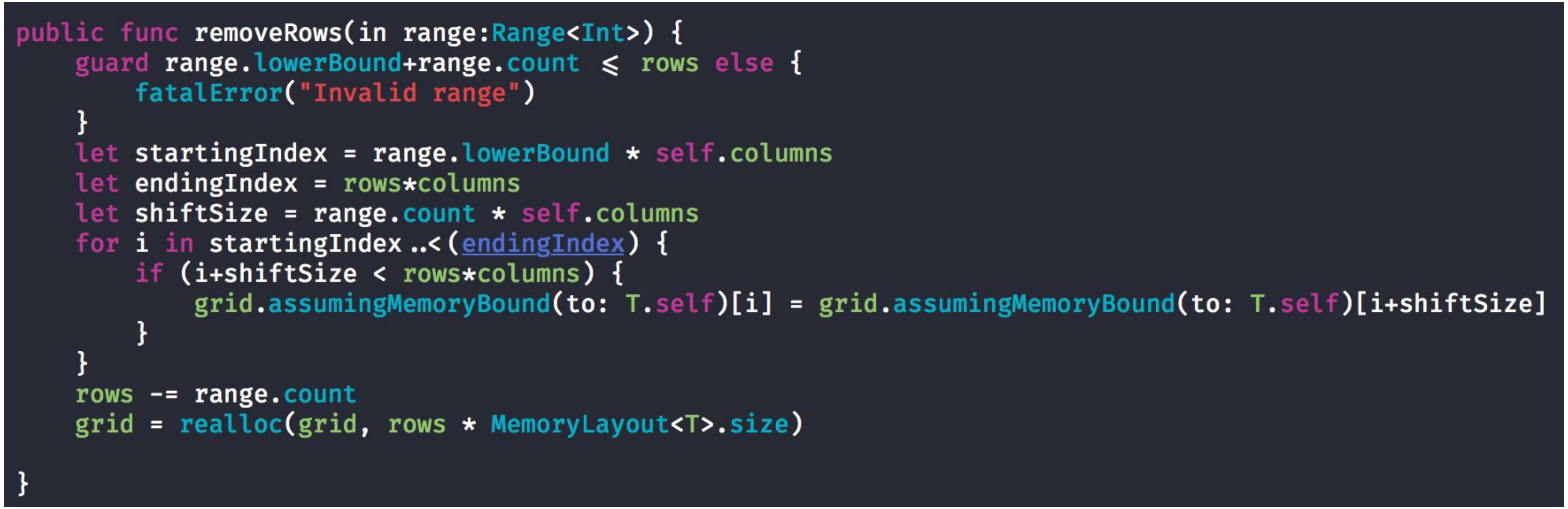
```swift
deinit {
    grid.deallocate(bytes: columns * rows * MemoryLayout<T>.size, alignedTo: 0)
}
```

2.2.2 - Element get/set



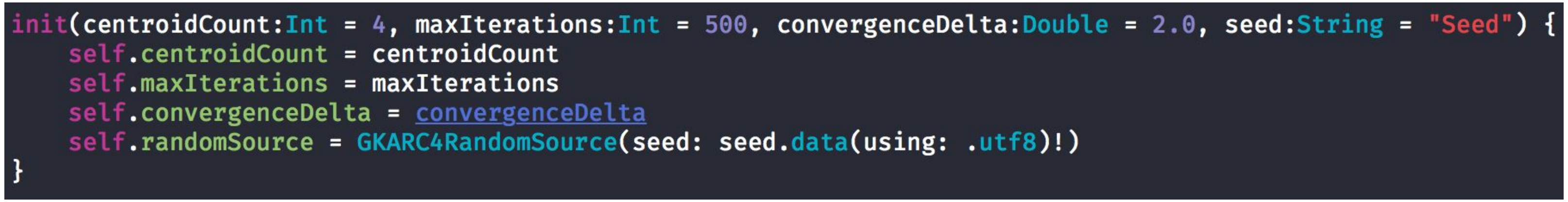
```swift
subscript(row: Int, column: Int) -> T {
    get {
        guard row < rows && column < columns else {
            fatalError("Invalid row or column")
        }
        return grid.assumingMemoryBound(to: T.self)[(row * columns) + column]
    }
    set {
        guard row < rows && column < columns else {
            fatalError("Invalid row or column")
        }
        grid.assumingMemoryBound(to: T.self)[(row * columns) + column] = newValue
    }
}
```

2.2.3 - Delete routine



```swift
public func removeRows(in range:Range<Int>) {
    guard range.lowerBound+range.count ⩽ rows else {
        fatalError("Invalid range")
    }
    let startingIndex = range.lowerBound * self.columns
    let endingIndex = rows*columns
    let shiftSize = range.count * self.columns
    for i in startingIndex ..<(endingIndex) {
        if (i+shiftSize < rows*columns) {
            grid.assumingMemoryBound(to: T.self)[i] = grid.assumingMemoryBound(to: T.self)[i+shiftSize]
        }
    }
    rows -= range.count
    grid = realloc(grid, rows * MemoryLayout<T>.size)

}
```

2.3 - Optional parameters



```swift
init(centroidCount:Int = 4, maxIterations:Int = 500, convergenceDelta:Double = 2.0, seed:String = "Seed") {
    self.centroidCount = centroidCount
    self.maxIterations = maxIterations
    self.convergenceDelta = convergenceDelta
    self.randomSource = GKARC4RandomSource(seed: seed.data(using: .utf8)!)
}
```

Questions & Answers

Thank you very much for your participation.

Lucas Farris - lfarris@softserveinc.com