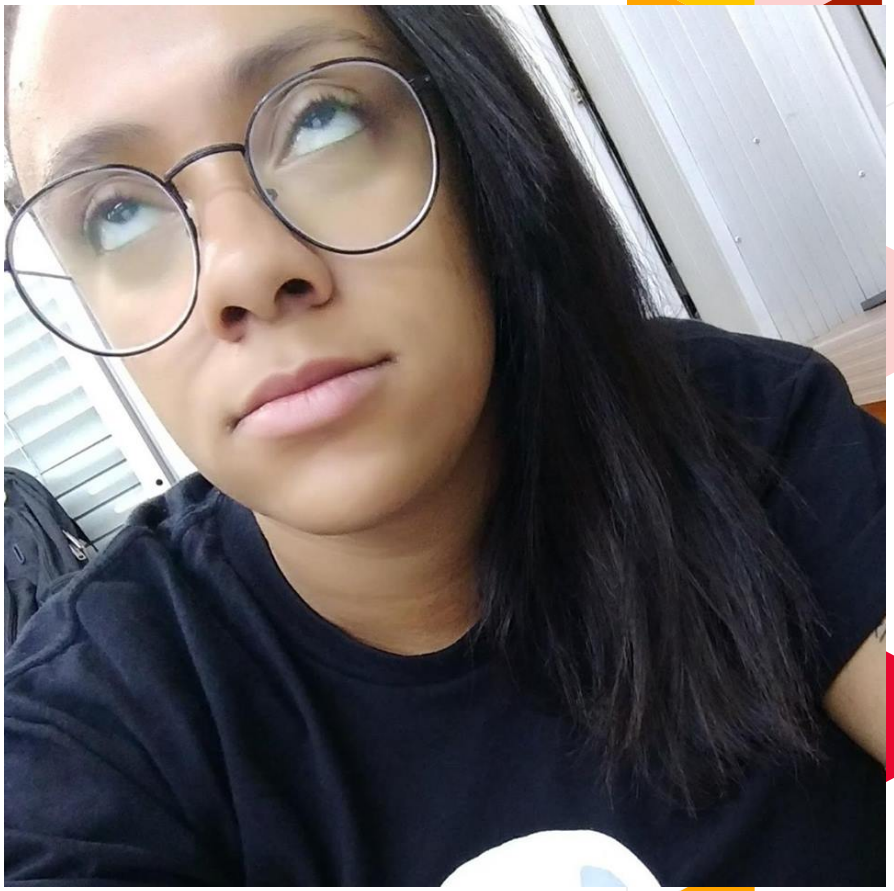
Artificial Intelligence with Ruby?

Hello!

I am Natalia Raythz | Nat

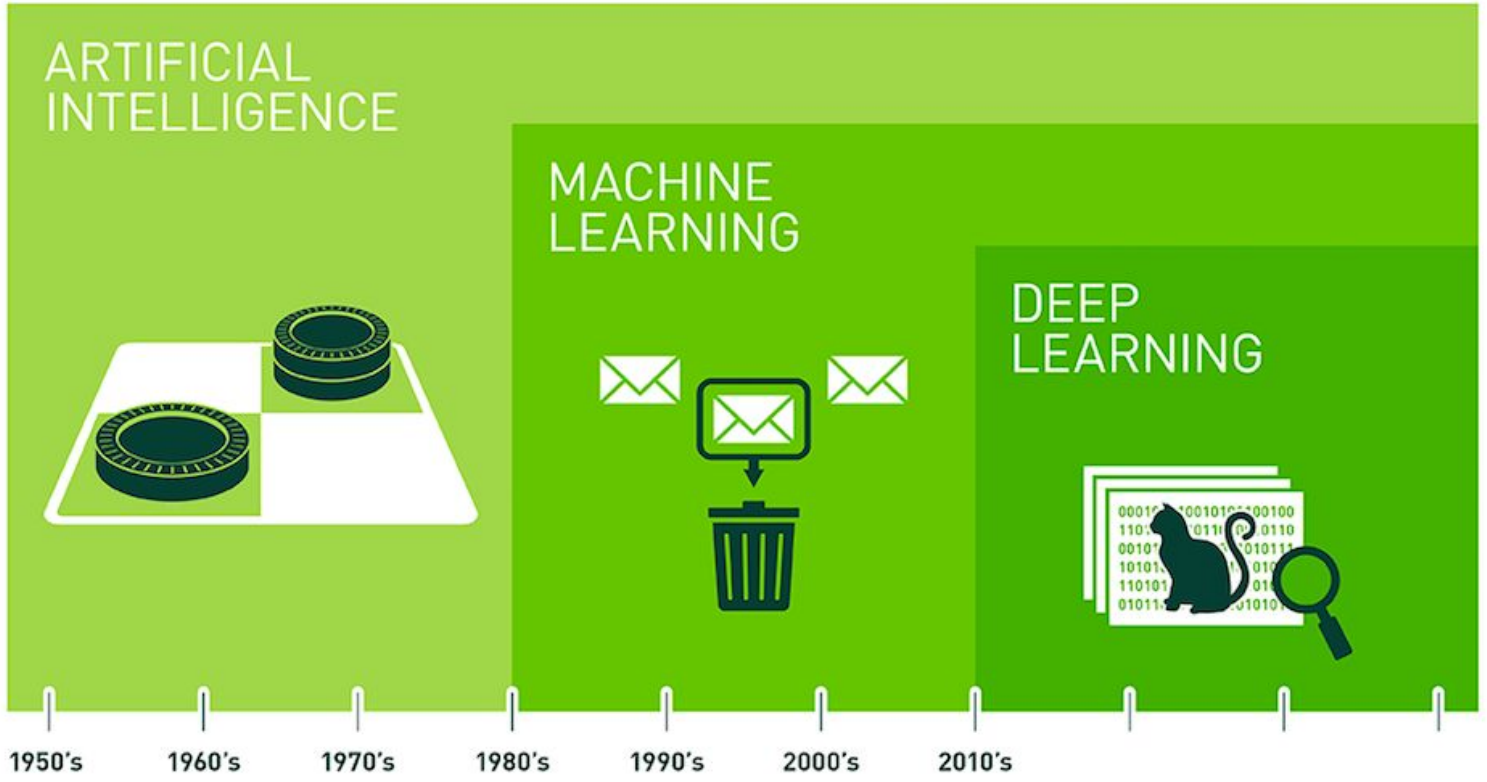
AIOps at ADP Brazil Labs

You can find me at
@guriadeprograma

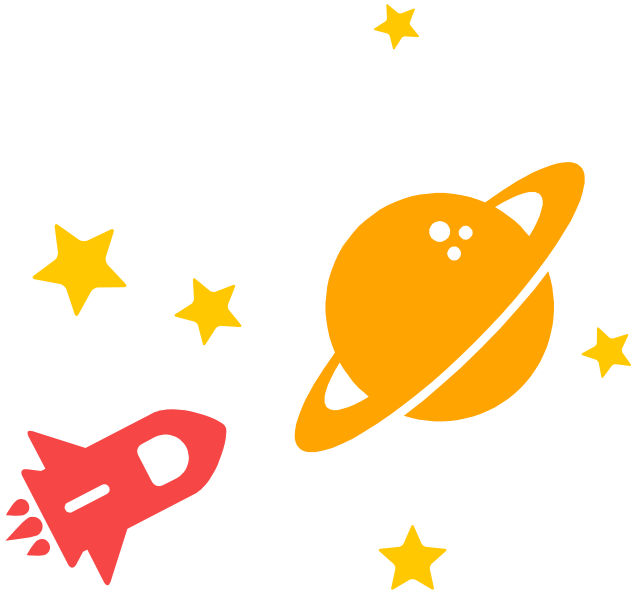


The data determines the output.

What is Artificial Intelligence?


ARTIFICIAL INTELLIGENCE
MACHINE LEARNING
DEEP LEARNING
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Where does Ruby get into this story?



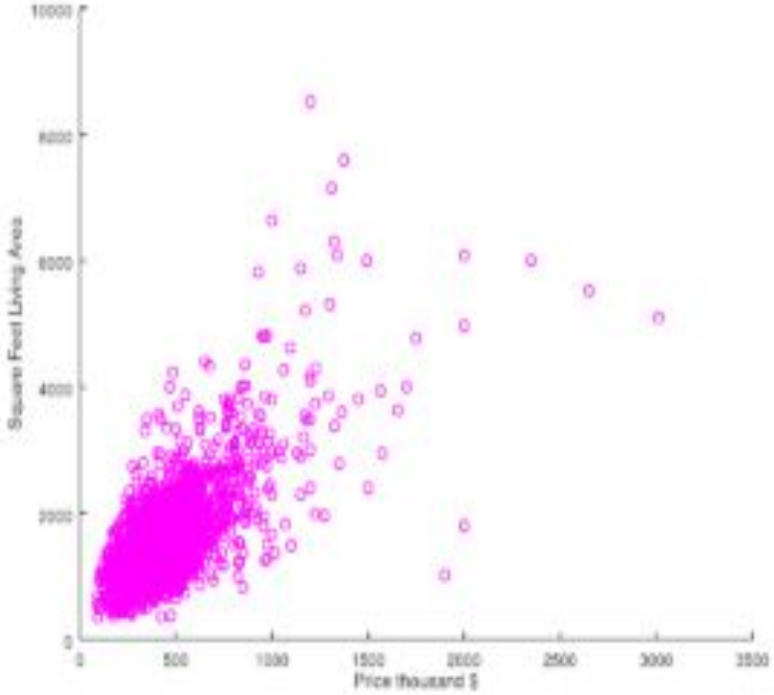
Why Ruby Isn’t typically used for Machine Learning.

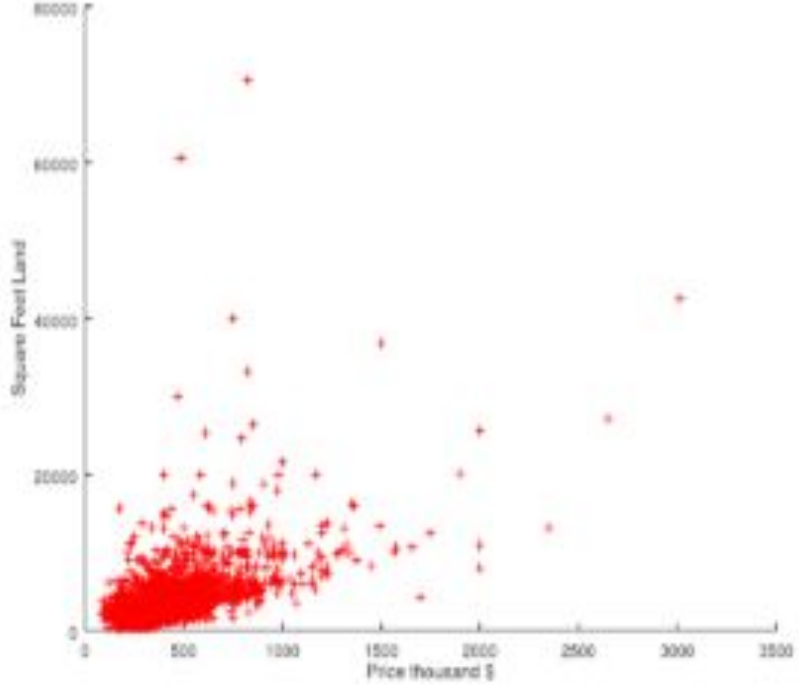
- Misconceived notions about speed
- Lack of libraries
- Ease of passing responsibilities to others services

Some examples

Linear regression

```
1.  LAND SQUARE FEET,GROSS SQUARE FEET,SALE PRICE,BOROUGH,NEIGHBORHOOD,TAX CLASS AT
    PRESENT,BLOCK,LOT,EASE-MENT,BUILDING CLASS AT PRESENT,ZIP CODE,YEAR BUILT,TAX
    CLASS AT TIME OF SALE,BUILDING CLASS AT TIME OF SALE,SALE DATE
2.  13390,5994,1495000,5,ANNADALE ,1,6475,85, ,A3,10312,2002,1, A3 ,7/28/2015
3.  6180,4808,975000,5,ANNADALE ,1,6370,4, ,A3,10312,1990,1, A3 ,11/20/2015
4.  13406,4180,1199000,5,ANNADALE ,1,5394,4, ,A2,10312,1982,1, A2 ,8/26/2015
5.  8000,4011,865000,5,ANNADALE ,1,6222,54, ,A1,10312,2000,1, A1 ,1/12/2015
6.  30000,4000,470000,5,ANNADALE ,1,6499,40, ,A1,10312,1985,1, A1 ,4/30/2015
7.
8.  ......
```


Square Feet Living Area
Price thousand $


Square Feet Land
Price thousand $

How?

Installing prerequisites

```
gem install ruby_linear_regression
```

Implementing linear regression

```
1.  require 'csv'
2.  require 'ruby_linear_regression'
```

```
1.  x_data = []
2.  y_data = []
3.  # Load data from CSV file into two arrays - one for independent variables
    X and one for the dependent variable Y
4.  # Each row contains square feet for property and living area like this:
5.  # [ SQ FEET PROPERTY, SQ FEET HOUSE ]
6.  CSV.foreach("./data/staten-island-single-family-home-sales-2015.csv" ,
    :headers => true) do |row|
7.  x_data.push( [row[0].to_i, row[1].to_i] )
8.  y_data.push( row[2].to_i )
9.  end
```

```ruby
# Create regression model
linear_regression = RubyLinearRegression.new
# Load training data
linear_regression.load_training_data(x_data, y_data)
```

```ruby
# Train the model using the normal equation
linear_regression.train_normal_equation
```

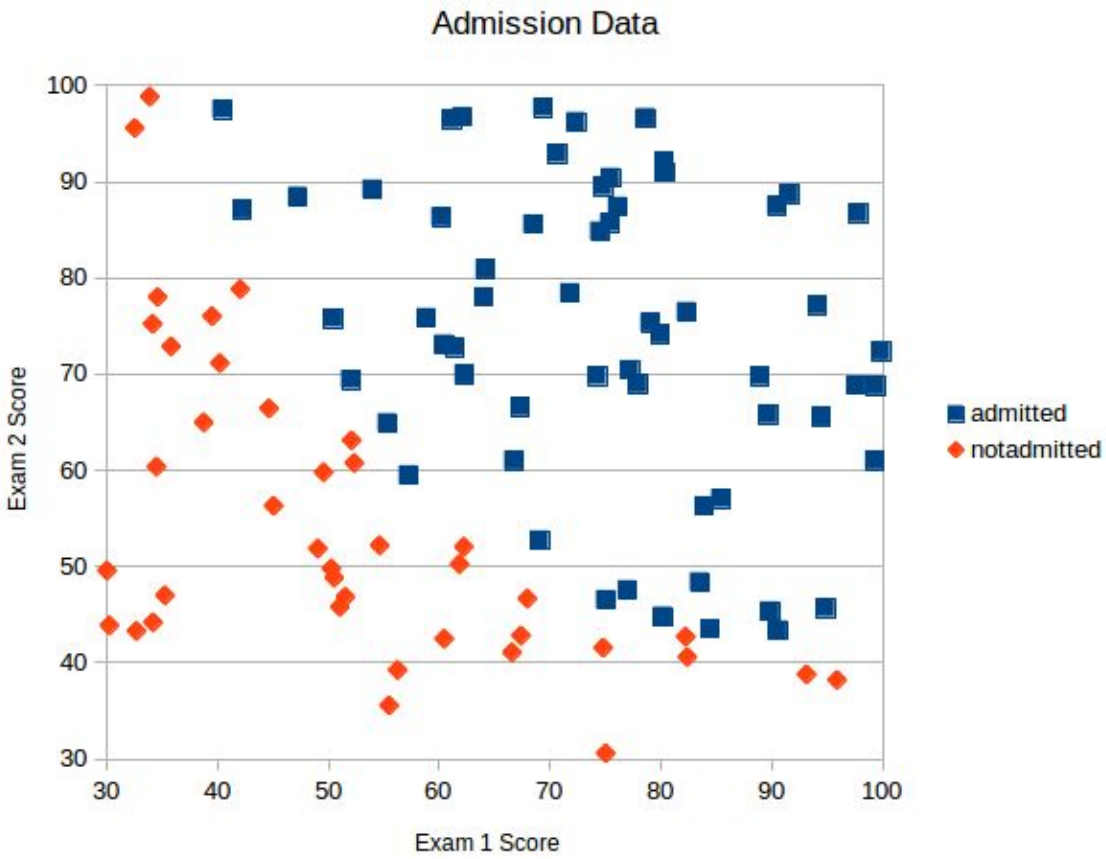
```ruby
# Predict the price of a 2000 sq feet property with a 1500 sq feet house
prediction_data = [2000, 1500]
predicted_price = linear_regression.predict(prediction_data)
puts "Predicted selling price for a 1500 sq feet house on a 2000 sq feet property: #{predicted_price.round}$"
```

```
$ ruby example.rb

Predicted selling price for a 1500 sq feet house on a 2000 sq

feet property: 395853$
```

Simple classification using a Neural Network



Setting up a Neural Network in Ruby

```
gem install ruby-fann
```

```
1. require 'csv'
2. require 'ruby-fann'
```

```
1. x_data = []
2. y_data = []
3. # Load data from CSV file into two arrays - one for
   independent variables X and one for the dependent variable
   Y
4. CSV.foreach("./data/admission.csv", :headers => false) do
   |row|
5.  x_data.push( [row[0].to_f, row[1].to_f] )
6.  y_data.push( [row[2].to_i] )
7. end
```

```
1.  # Divide data into a training set and test set
2.  test_size_percentange = 20.0 # 20.0%
3.  test_set_size = x_data.size * (test_size_percentange/100.to_f)
4.
5.  test_x_data = x_data[0 .. (test_set_size-1)]
6.  test_y_data = y_data[0 .. (test_set_size-1)]
7.
8.  training_x_data = x_data[test_set_size .. x_data.size]
9.  training_y_data = y_data[test_set_size .. y_data.size]
```

```
# Setup training data model
train = RubyFann::TrainData.new( :inputs=> training_x_data,
:desired_outputs=>training_y_data );

# Setup model and train using training data
model = RubyFann::Standard.new(
            num_inputs: 2,
            hidden_neurons: [6],
            num_outputs: 1 );

# 5000 max_epochs, 500 errors between reports and 0.01 desired
mean-squared-error
model.train_on_data(train, 5000, 500, 0.01)
```

```ruby
# Predict single class
prediction = model.run( [45, 85] )
# Round the output to get the prediction
puts "Algorithm predicted class: #{prediction.map{ |e| e.round }}"
```

```ruby
predicted = []
test_x_data.each do |params|
 predicted.push( model.run(params).map{ |e| e.round } )
end

correct = predicted.collect.with_index { |e,i| (e == test_y_data[i]) ? 1 : 0 }.inject{ |sum,e| sum+e }

puts "Accuracy: #{((correct.to_f / test_set_size) * 100).round(2)}% - test set of size #{test_size_percentange}%"
```

```
$ ruby nn.rb
Max epochs 5000. Desired error: 0.0099999998.
Epochs 1. Current error: 0.2485879362. Bit fail 80.
Epochs 500. Current error: 0.0141996695. Bit fail 4.
Epochs 1000. Current error: 0.0120923920. Bit fail 1.

Epochs 1500. Current error: 0.0116548287. Bit fail 1.
Epochs 2000. Current error: 0.0118962619. Bit fail 2.
Epochs 2500. Current error: 0.0116990097. Bit fail 2.
Epochs 3000. Current error: 0.0111343693. Bit fail 2.
Epochs 3366. Current error: 0.0099999355. Bit fail 1.
Algorithm predicted class: [1]
Accuracy: 100.0% - test set of size 20.0%
```


#TheRundownBET
Questions?

You can find me at

- @guriadeprograma
- nraythz@gmail.com