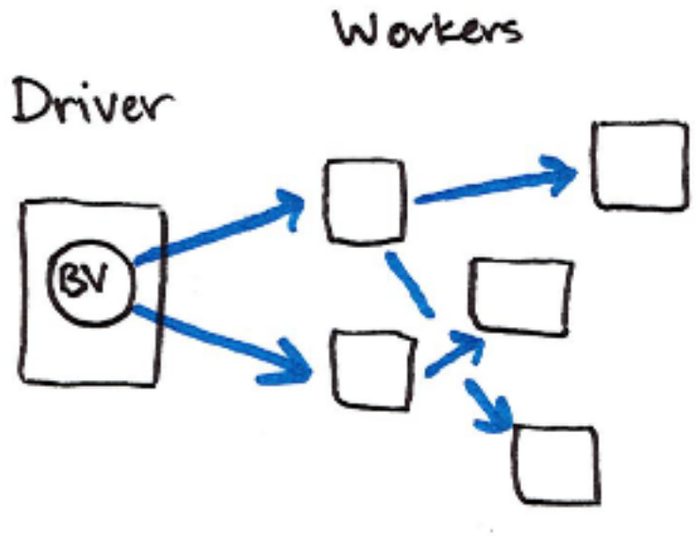
CS 696 Intro to Big Data: Tools and Methods
Fall Semester, 2017
Doc 11 Variables, IO, ML
Oct 4, 2017

Copyright ©, All rights reserved. 2017 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.

Distributed Variables

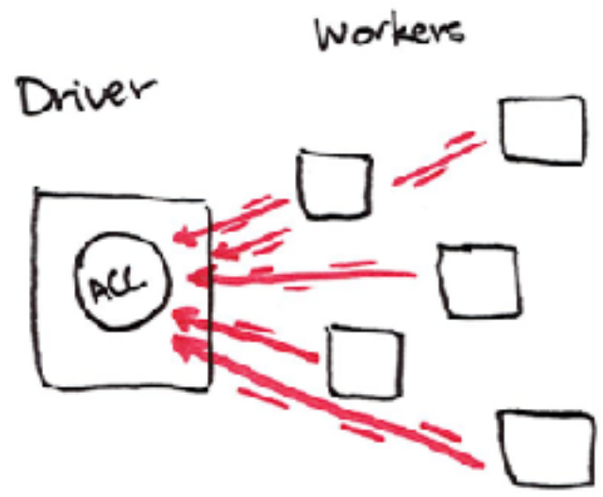
Broadcast

- Read-only data shared among workers



Accumulator

- Write only by workers
- Read only on master



Broadcast Example

```
val courseSize = 58
val courseSizeBroadcast = spark.sparkContext.broadcast(courseSize)

courseSizeBroadcast                 //Broadcast(1)
courseSizeBroadcast.value           // 58


val data = spark.sparkContext.parallelize(List(1,2,3,4,5,6,7,8), 2)
data.map(x => x + courseSizeBroadcast.value).collect

        Array(59, 60, 61, 62, 63, 64, 65, 66)
```

Using ComplexType

```
val sampleMap = Map("a"-> 10, "bat" -> 1)
val sampleBroadCast = spark.sparkContext.broadcast(sampleMap)
sampleBroadCast.value
```

```
import org.apache.spark.sql.SparkSession
val blockSize =  "4096"
val spark = SparkSession.builder().
      appName("Broadcast Test").
      config("spark.broadcast.blockSize", blockSize).
      getOrCreate()

val sc = spark.sparkContext
val slices =  2
val num =  10000000

val arr1 = (0 until num).toArray

for (i <- 0 until 3) {
    println("Iteration " + i)
    println("===========")
    val startTime = System.nanoTime
    val barr1 = sc.broadcast(arr1)
    val observedSizes = sc.parallelize(1 to 10, slices).map(_ => barr1.value.length)
   observedSizes.collect().foreach(i => println(i))
    println("Iteration %d took %.0f milliseconds".format(i, (System.nanoTime - startTime) / 1E6))
}
```

Accumulator Example

```
import org.apache.spark.util.LongAccumulator

val countMap = new LongAccumulator
spark.sparkContext.register(countMap, "mapCount")

val data = spark.sparkContext.parallelize(List(1,2,3,4,5,6,7,8), 2)
data.map(x => {countMap.add(1); x + 1})
```

countMap.value

ShortCut

```
val shortCut = spark.sparkContext.longAccumulator("Foo")

shortCut.value

        0
```

Accumulators

CollectionAccumulator
DoubleAccumulator
LongAccumulator

AccumulatorV2
    Subclass to create own Accumulator

IO

Reading into a DataFrame:

- Format
- Schema
- Read Mode
- Options
- Path

```
spark.read.format("csv")
  .schema(someSchema)
  .option("mode", "FAILFAST")
  .option("inferSchema", "true")
  .load("path/to/file(s)")
```

Read Modes

readMode	Description
`permissive`	Sets all fields to null when it encounters a corrupted record.
`dropMalformed`	Drops the row that contains malformed records.
`failFast`	Fails immediately upon encountering malformed records.

Example

ages.csv

```
name,age
A,10
B,20
C,3cat
D,30
E,
F,5.0
```

```
import org.apache.spark.sql.types.{StructField, StructType,
                                   StringType, IntegerType}

val manualSchema = new StructType(Array(
  new StructField("name", StringType, true),
  new StructField("age", IntegerType, true)
))

val age = spark.read.format("csv").
        schema(manualSchema).
        option("mode","DROPMALFORMED").
        option("header",true).
load("ages.csv")
```

PERMISSIVE

```
+----+----+
|name| age|
+----+----+
|   A|  10|
|   B|  20|
|null|null|
|   D|  30|
|   E|null|
|null|null|
+----+----+
```

DROPMALFORMED

```
+----+----+
|name| age|
+----+----+
|   A|  10|
|   B|  20|
|   D|  30|
|   E|null|
+----+----+
```

FAILFAST

```
Name: org.apache.spark.SparkException
```

Save Mode

saveMode	Description
`append`	Appends the output files to the list of files that already exist at that location.
`overwrite`	Will completely overwrite any data that already exists there.
`errorIfExists`	Throws an error and fails the write if data or files already exist at the specified location.
`ignore`	If data or files exist at the location, do nothing with the current DataFrame.

```
dataframe.write.format("csv")
  .option("mode", "OVERWRITE")
  .option("dateFormat", "yyyy-MM-dd")
  .save(“FooBar")
```

What are the Options?

org.apache.spark.sql.DataFrameWriter

sep
quote
header
nullValue
compression

dateFormat
timestampFormat

```
def csv(path: String): Unit
```

Saves the content of the `DataFrame` in CSV format at the specified path. This is equivalent to:

```
format("csv").save(path)
```

You can set the following CSV-specific option(s) for writing CSV files:

- `sep` (default `,`): sets the single character as a separator for each field and value.
- `quote` (default `"`): sets the single character used for escaping quoted values where the separator can be part of the value.
- `escape` (default `\`): sets the single character used for escaping quotes inside an already quoted value.
- `escapeQuotes` (default `true`): a flag indicating whether values containing quotes should always be enclosed in quotes. Default is escape all values containing a quote character.
- `quoteAll` (default `false`): a flag indicating whether all values should always be enclosed in quotes. Default is to only escape val containing a quote character.
- `header` (default `false`): writes the names of columns as the first line.
- `nullValue` (default empty string): sets the string representation of a null value.
- `compression` (default `null`): compression codec to use when saving to file. This can be one of the known case-insensitive short names (`none`, `bzip2`, `gzip`, `lz4`, `snappy` and `deflate`).
- `dateFormat` (default `yyyy-MM-dd`): sets the string that indicates a date format. Custom date formats follow the formats at `java.text.SimpleDateFormat`. This applies to date type.
- `timestampFormat` (default `yyyy-MM-dd'T'HH:mm:ss.SSSXXX`): sets the string that indicates a timestamp format. Custom dat formats follow the formats at `java.text.SimpleDateFormat`. This applies to timestamp type.
- `ignoreLeadingWhiteSpace` (default `true`): a flag indicating whether or not leading whitespaces from values being written shou be skipped.
- `ignoreTrailingWhiteSpace` (default `true`): a flag indicating defines whether or not trailing whitespaces from values being written should be skipped.

Parquet Files

Column-oriented data store for Hadoop ecosystem

Column

- Stored in contiguous memory locations
- Each column has its own encoding and compression
- Can read specific columns

```
age.write.format("parquet").save("parquetExample.parquet")
```

Reading from SQL Databases

Need to load JDBC driver for given database

```
val props = new java.util.Properties
props.setProperty("driver", "org.sqlite.JDBC") /
props.setProperty("username", "some-username")
props.setProperty("password", "some-password")
val hostname = "192.168.1.5"
val port = "2345"

val dbDatabase = "DATABASE"
val dbTable = "test"

val dbDataFrame = spark.read.jdbc(url, tablename, props)
```

Query Pushdown

Spark will try to filter the data using the database

To avoid reading data

Reading Databases in Parallel

When reading from SQL database each read puts data in one partition.

Specify multiple partitions using predicates

```
val predicates = Array(
  "DEST_COUNTRY_NAME = 'India'",
  "DEST_COUNTRY_NAME = 'United States'")

val dbDataFrame = spark.read.jdbc(url, tablename, predicates, props)
```

Partitioning on a Column

```
val colName = "count"
val lowerBound = 0L
val upperBound = 348113L // this is the max count in our database
val numPartitions = 10


spark.read.jdbc(url,
                tablename,
                colName,
                lowerBound,
                upperBound,
                numPartitions,
                props)
.count()
```

Dealing with NULL, NaN and Bad Values

```
+----+----+
|name| age|
+----+----+
|   A|  10|
|   B|  20|
|null|null|
|   D|  30|
|   E|null|
|null|null|
+----+----+
```

org.apache.spark.sql.DataFrameNaFunctions

drop

- drop rows with null or NaN
- Multiple versions

fill

- Replace null and NaN
- Multiple versions

replace

- Replace values
- Multiple versions

```
import org.apache.spark.sql.types.{StructField, StructType, StringType, IntegerType}

val manualSchema = new StructType(Array(
  new StructField("name", StringType, true),
  new StructField("age", IntegerType, true)
))

val age = spark.read.format("csv").
        schema(manualSchema).
        option("header",true).
        load("ages.csv")
age.show
```

```
+----+----+
|name| age|
+----+----+
|   A|  10|
|   B|  20|
|null|null|
|   D|  30|
|   E|null|
|null|null|
+----+----+
```

```
val nameCleaned = age.na.drop("any",Seq("name"))
nameCleaned.show
```

```
+----+----+
|name| age|
+----+----+
|   A|  10|
|   B|  20|
|   D|  30|
|   E|null|
+----+----+
```

```
val nameCleaned = age.na.drop("any",Seq("name"))
nameCleaned.show
```

```
+----+----+
|name| age|
+----+----+
|   A|  10|
|   B|  20|
|   D|  30|
|   E|null|
+----+----+
```

```
val cleaned = nameCleaned.na.fill(-1)
cleaned.show
```

```
+----+---+
|name|age|
+----+---+
|   A| 10|
|   B| 20|
|   D| 30|
|   E| -1|
+----+---+
```

```
val cleaned = nameCleaned.na.fill(0,Seq("age"))
cleaned.show
```

```
+----+---+
|name|age|
+----+---+
|   A| 10|
|   B| 20|
|   D| 30|
|   E|  0|
+----+---+
```

```
val replaced = cleaned.na.replace("age",Map(10->5))
replaced.show
```

```
+----+---+
|name|age|
+----+---+
|   A|  5|
|   B| 20|
|   D| 30|
|   E|  0|
+----+---+
```

Machine Learning in Spark

MLlib

RDD-based

org.apache.spark.mllib

Maintenance mode

DataFrame based (Spark ML)

org.apache.spark.ml

Pipelines

Inspired by Python scikit-learn

Classification

Regression

Clustering

Collaborative Filtering

Dimension reduction

Linear Algebra

Statistics

Machine Learning

Supervised

Unsupervised

Reinforcement learning

Classification

Regression

Clustering

Density Estimation

Dimensionality Reduction

Supervised learning

Artificial neural network
Bayesian statistics
Bayesian network
Gaussian process regression
Inductive logic programming
Learning Vector Quantization
Logistic Model Tree
Nearest Neighbor Algorithm
Random Forests
Ordinal classification
ANOVA
Linear classifiers
Fisher's linear discriminant
Linear regression
Logistic regression
Multinomial logistic regression
Naive Bayes classifier
Quadratic classifiers
k-nearest neighbor
Boosting
Decision trees
Random forests
Bayesian networks
Naive Bayes
Hidden Markov models

Unsupervised learning

Expectation-maximization algorithm
Vector Quantization
Generative topographic map
Information bottleneck method
Artificial neural networks

Hierarchical clustering
  Single-linkage clustering
  Conceptual clustering
  Cluster analysis[edit]
  K-means algorithm
  Fuzzy clustering
  DBSCAN
  OPTICS algorithm

Outlier Detection
  Local Outlier Factor

Other

- Reinforcement learning
  - Temporal difference learning
  - Q-learning
  - Learning Automata
  - SARSA
- Deep learning
  - Deep belief networks
  - Deep Boltzmann machines
  - Deep Convolutional neural networks
  - Deep Recurrent neural networks
  - Hierarchical temporal memory

Machine Learning & Patterns

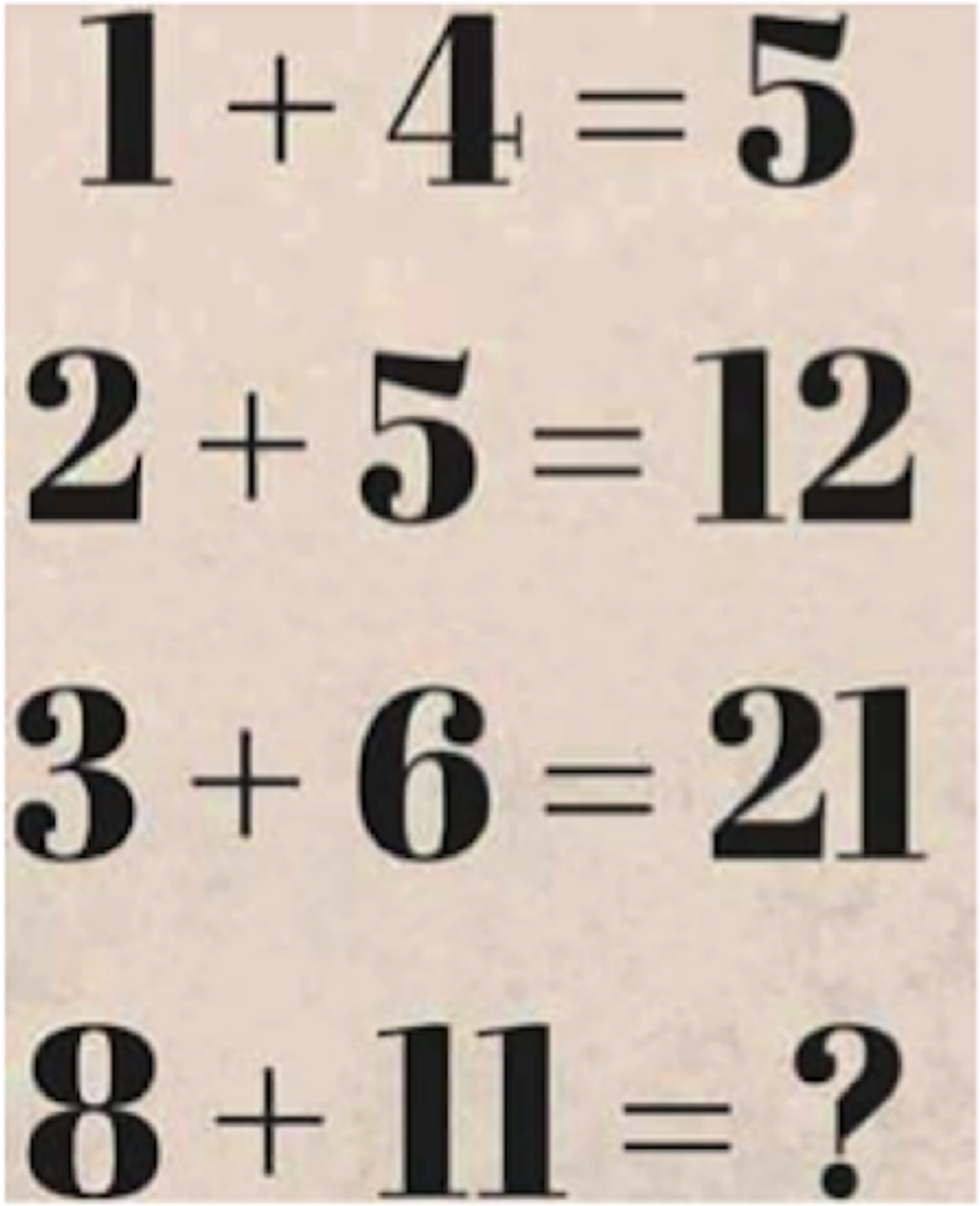
Machine learning algorithms
- Detect patterns
- Generate models based on those patterns

Feed a neural network pictures of cats
- Neural net can identify cats
- Can automate finding cat photo on internet

Drive a car with neural network “watching”
- You actions
- Videos of surroundings

- Neural net can identify patterns & start to drive

Limits of Pattern Matching



1 * (4 + 1) = 5

2 * (5 + 1) = 12

3 * (6 + 1) = 21

8 * (11 + 1) = 96

0 + 1 + 4 = 5

5 + 2 + 5 = 12

12 + 3 + 6 = 21

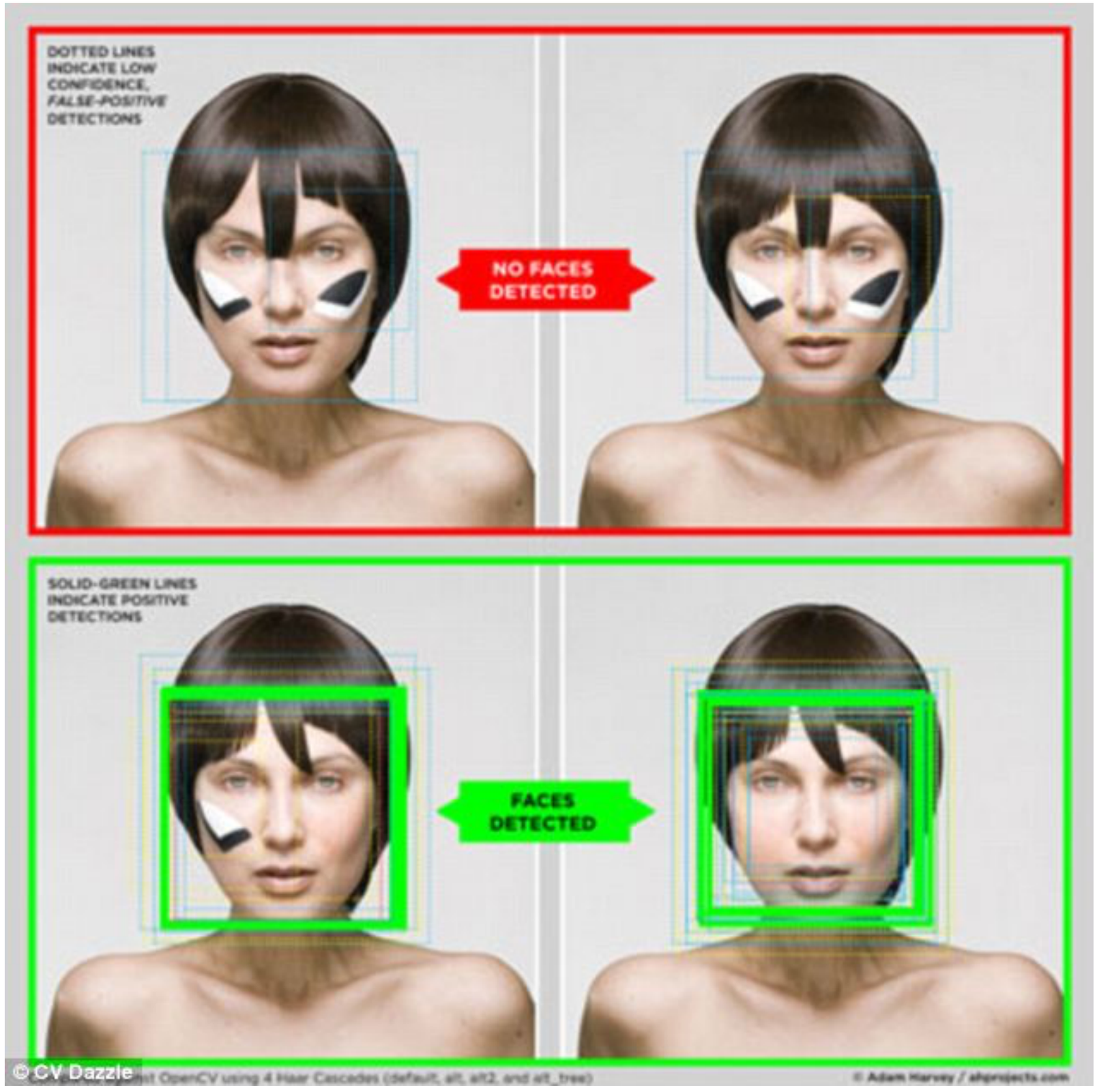
21 + 8 + 11 = 40

No Free Lunch Theorems

David Wolpert

> For every pattern a machine learning algorithm is good at learning, there's another pattern that same learner would be terrible at picking up

No Free Lunch



7 Deadly Sins of AI Predictions

Rodney Brooks, October 6, 2017

https://goo.gl/oK6z5Z

1. Amara's law

   We tend to overestimate the effect of a technology in the short run and underestimate the effect in the long run.

Example U.S. Global Positioning System (GPS)

- Started 1978
- Precise delivery of bombs
- First real use 1991, but not fully embraced by US military for several more years
- Now
  - On mobile phones
  - Tracks planes, trucks
  - Sync US electrical grid
  - Determines which seed to plant in a field

7 Deadly Sins of AI Predictions

Rodney Brooks, October 6, 2017

https://goo.gl/oK6z5Z

3. Performance versus competence

7 Deadly Sins of AI Predictions

Rodney Brooks, October 6, 2017

https://goo.gl/oK6z5Z

3. Exponentials

Exponential growth not sustainable

iPod memory

year	gigabytes
2002	10
2003	20
2004	40
2006	80
2007	160

Deep Learning Breakthrough

Breakthrough paper on deep learning - back propagation

1986

Idea was abandoned for ~20 years because it was not producing results

Models

Machine Learning algorithms produce models

Models allow predictions or offer insights

Examples

Decreasing latency by X increases Amazon's daily revenue by Y

White males without college degrees favor Trump by X%

Females favor Clinton by Y%

...

Models Approximate Reality

World is flat

World is a sphere

World is an oblate ellipsoid

Does the model provide useful predictions/insights

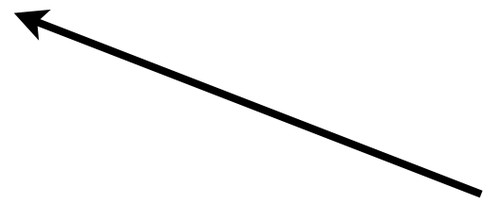
Under what condidtions is the model useful

What are the estimates of the model’s error

Multiple Factors in Model

Amazon's daily revenue depends on

- Latency
- Price
- Steps needed to order
- Page layout
- Relevant suggestions
- Search results
- Font sizes
- Color
- Shipping costs



Some factors will be more important

Stochastic in nature

Independent variables

Features

Regression

Regression

Measure of relation between mean of one variable (dependent) on

one or more other variables (independent)

Overview

Linear regression

Multiple linear regression

Generalized linear regression (model)

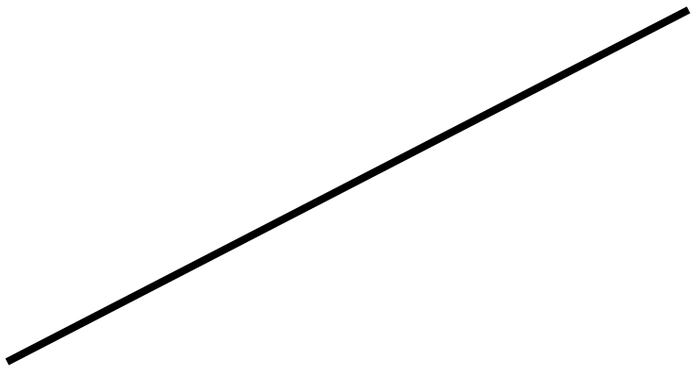
Is the dependent variable related to the independent variable

Generating the model

Error in the model

Effect of independent variables

Linear Regression



Model

f(x) = 2x + 3

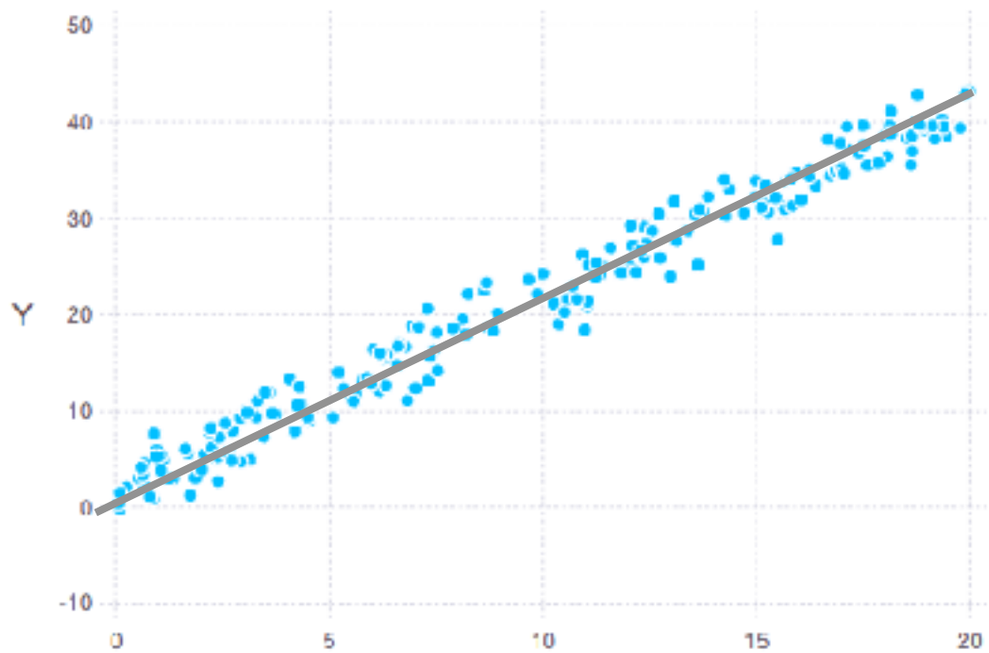
y = 2x + 3

y = 2x + 3

Dependent Variable

Independent Variable

Linear Regression



Actual relation (assumed)

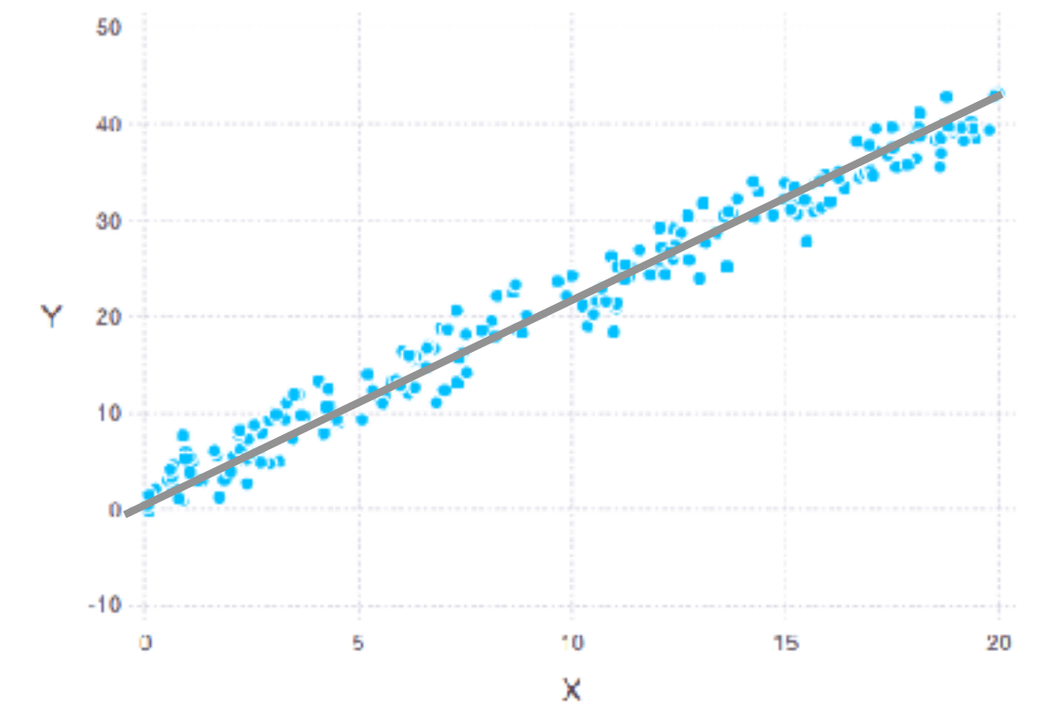
$$y = a + bx$$

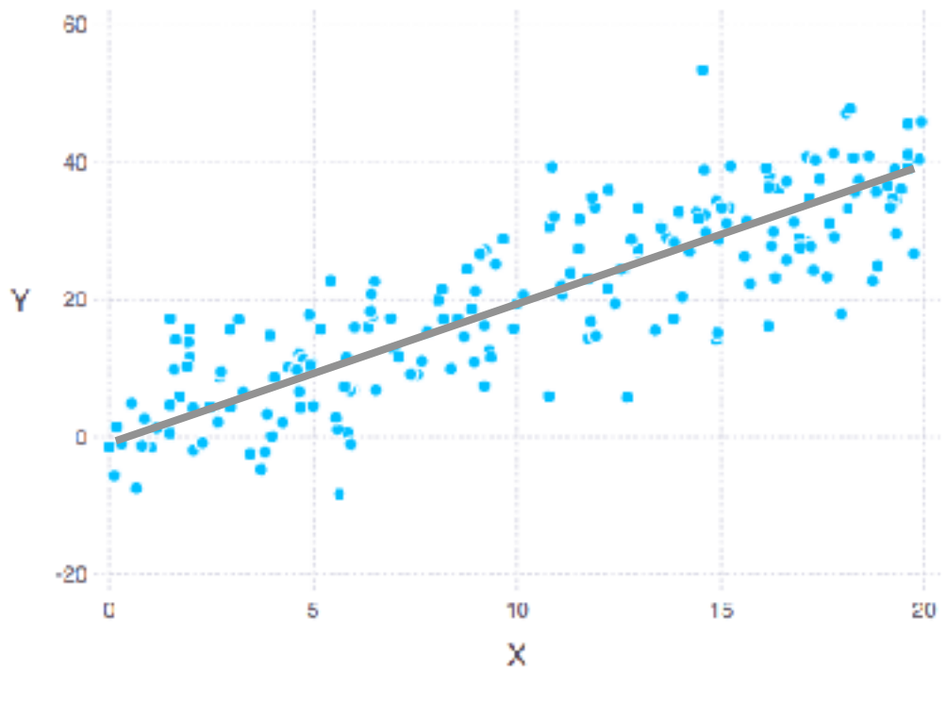
Compute linear line that fits the data best

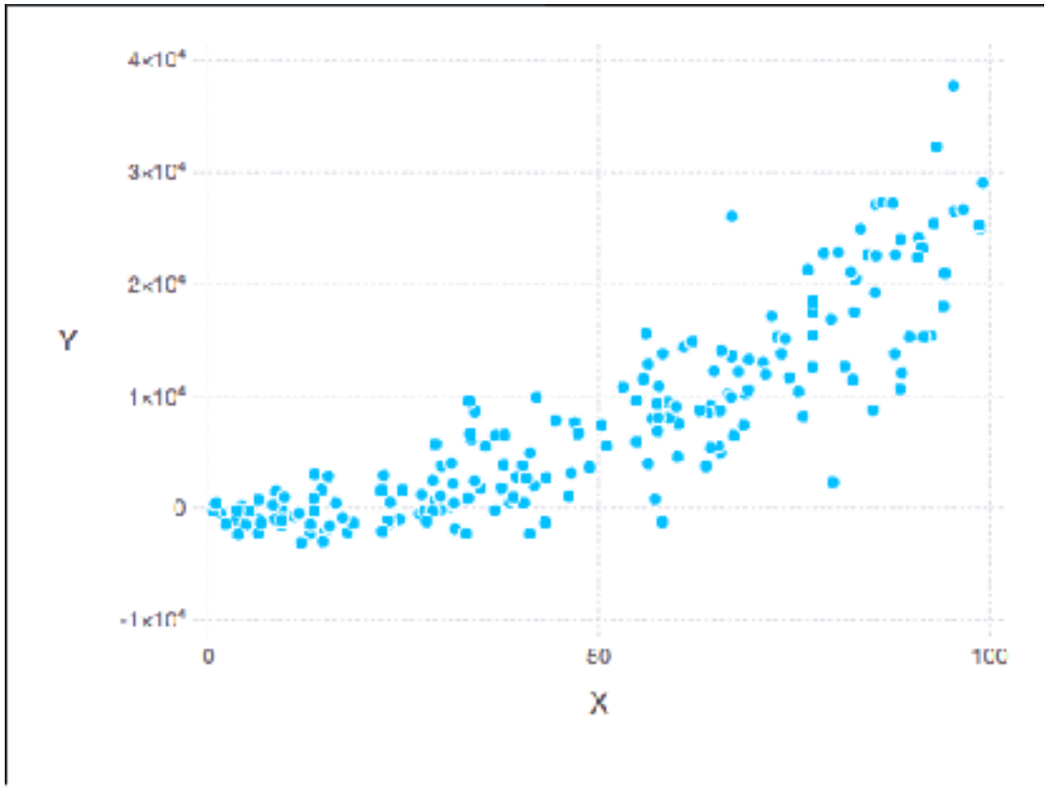
$$\hat{y} = a + bx + e$$

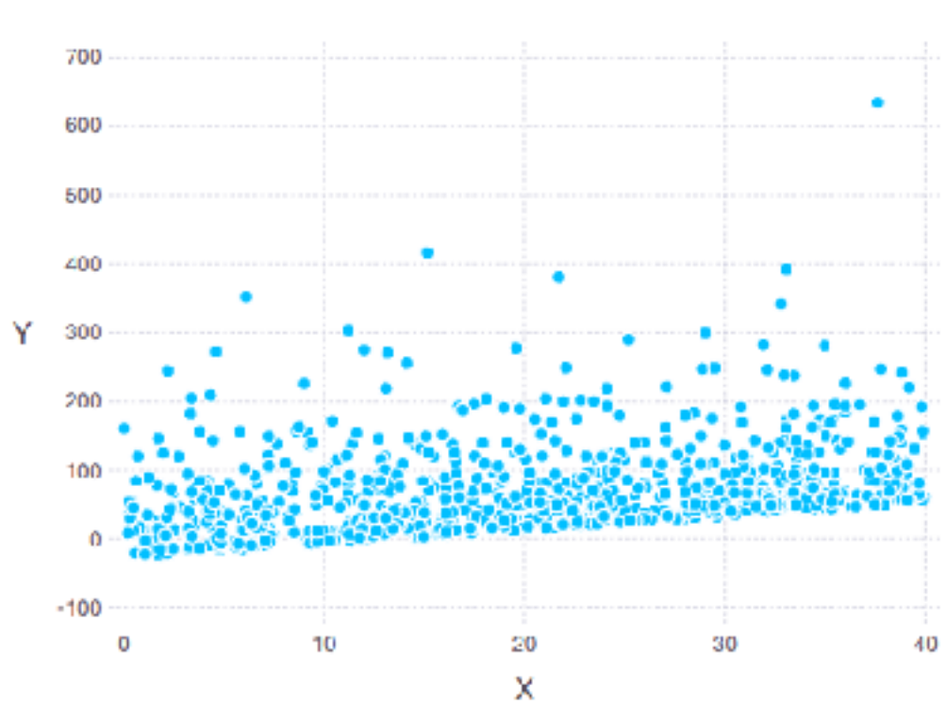
e - error or residual

Goal is to minimize residual overall

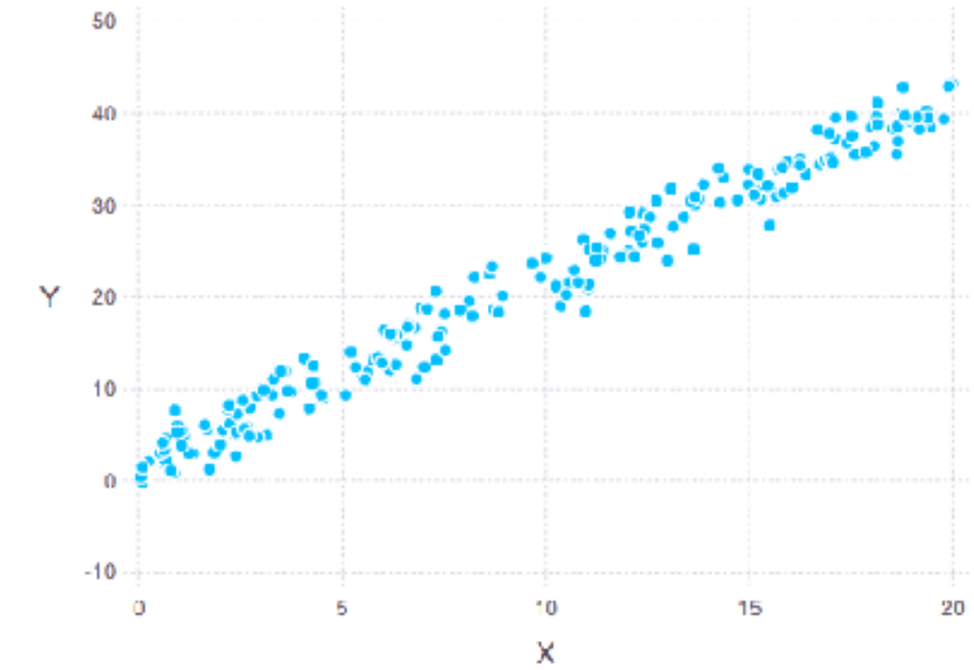

50
40
30
20
10
0
-10
Y
0
5
10
15
20
X

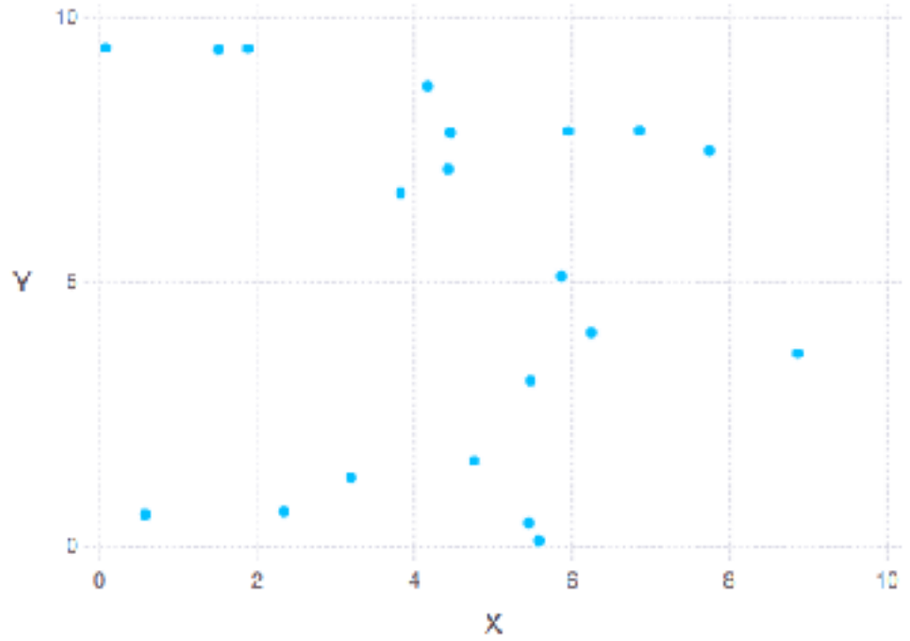

60
40
20
0
-20
Y
0
5
10
15
20
X


4×10⁴
3×10⁴
2×10⁴
1×10⁴
0
-1×10⁴
Y
0
50
100
X

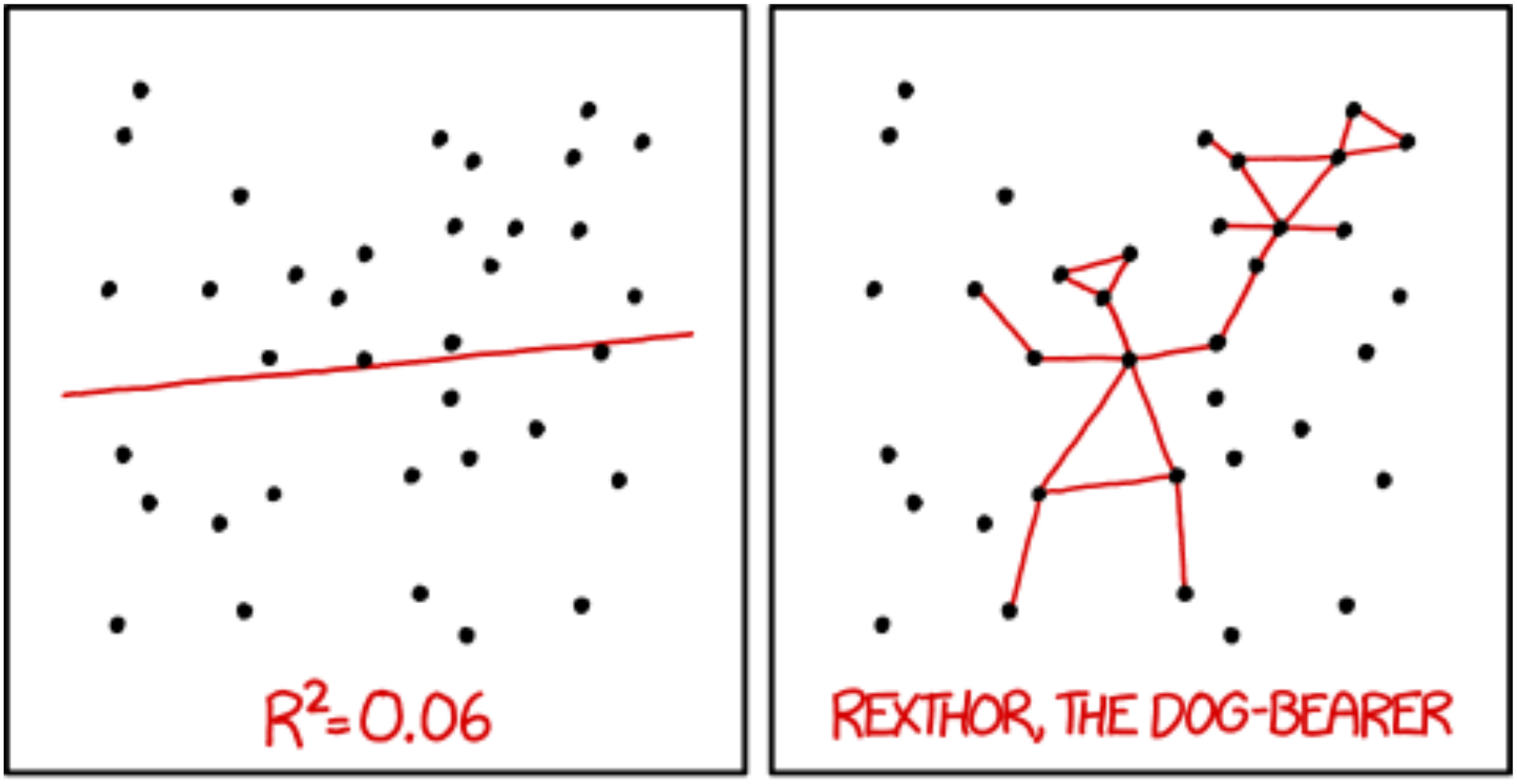

700
600
500
400
300
200
100
0
-100
Y
0
10
20
30
40
X

Are They Related?





XKCD



I DON'T TRUST LINEAR REGRESSIONS WHEN IT'S HARDER TO GUESS THE DIRECTION OF THE CORRELATION FROM THE SCATTER PLOT THAN TO FIND NEW CONSTELLATIONS ON IT.

Covariance

$$dx_i = x_i - \overline{x}$$

$$dy_i = y_i - \overline{y}$$

$$\mathrm{cov}(X,Y) = \frac{1}{n}\sum_{i=1}^{n} dx_i dy_i$$

If x & y are related then they should vary from their means in a similar way

Values near zero indicate no relation

positive values - positive relation

negative values - negative relation

Effects of Scale

Cost USD	Pounds	Grams
9	3	1357.8
24	7	3168.2
38	10	4526

1 Pound = 452.6 grams

Changing the scale of units

Does not change the relationship

Does change magnitude of Covariance

Makes covariance hard to evaluate

cov(pounds,Cost USD) == 50.8

cov(grams, Cost USD) == 23007

cov(grams, Cost INR) == 1,528,308.996

Units

$$dx_i = x_i - \overline{x}$$ Lbs

$$dy_i = y_i - \overline{y}$$ USD

$$\mathrm{cov}(X,Y) = \frac{1}{n}\sum_{i=1}^{n} dx_i dy_i$$ cov(pounds,Cost USD) == 50.8 lbs*USD

cov(grams, Cost USD) == 23007 grams*USD

Cost USD	Pounds	Grams
9	3	1357.8
24	7	3168.2
38	10	4526

Normalizing Data

Convert data to a common scale

Example - divide by maximum value

Cost USD	Pounds	Grams
9	3	1357.8
24	7	3168.2
38	10	4526

Cost	Amount
0.237	0.3
0.632	0.7
1	1

cov(Cost,Amount) == 0.134 (unit less)

Pearson's Correlation - r

$$r = \frac{\mathrm{cov}(X,Y)}{\sigma_x \sigma_y}$$

Normalized Covariance

Unit less

Range -1 to 1

1 = maximumly related

-1 - maximumly inversely related

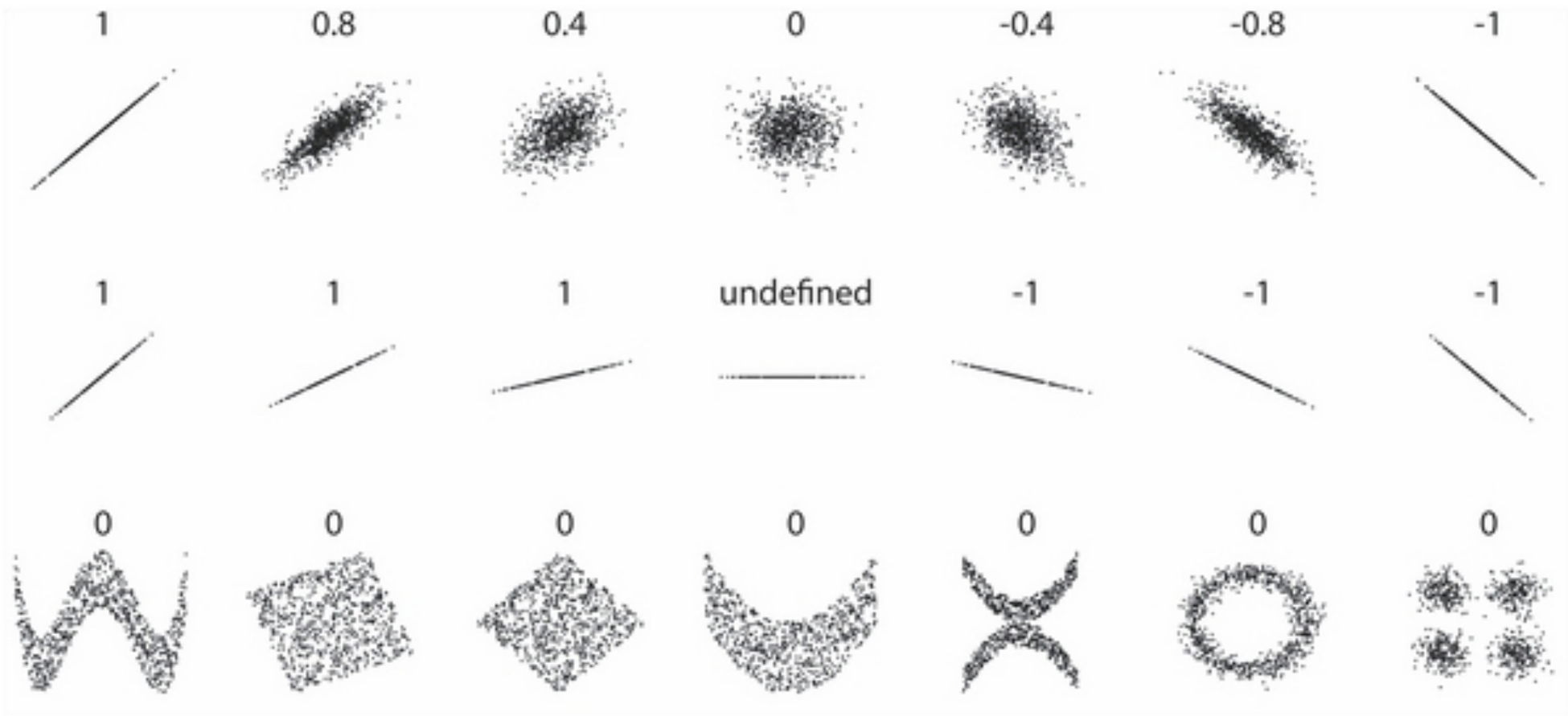
0 - not related

Pearson's Correlation - r

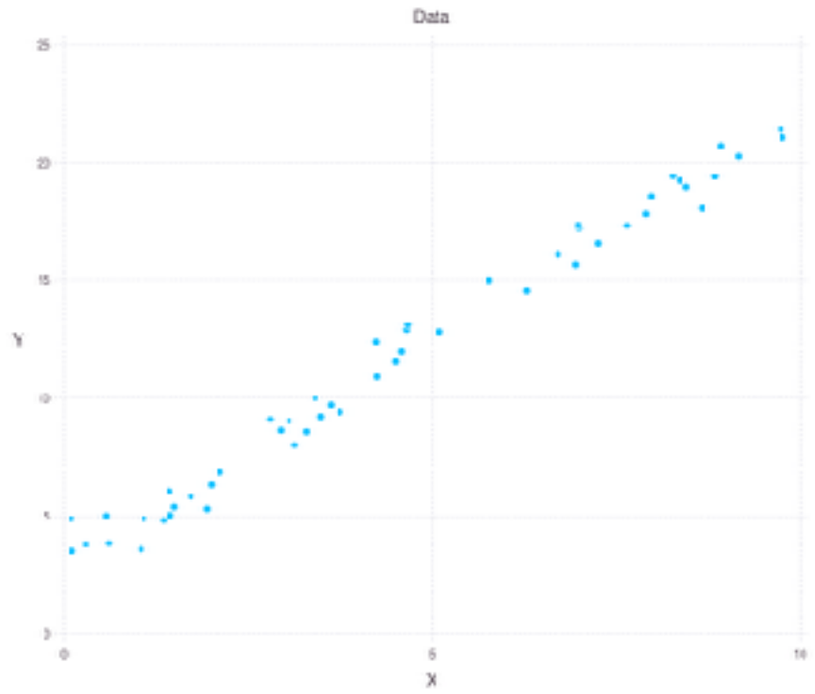
Cost USD	Pounds	Grams
9	3	1357.8
24	7	3168.2
38	10	4526

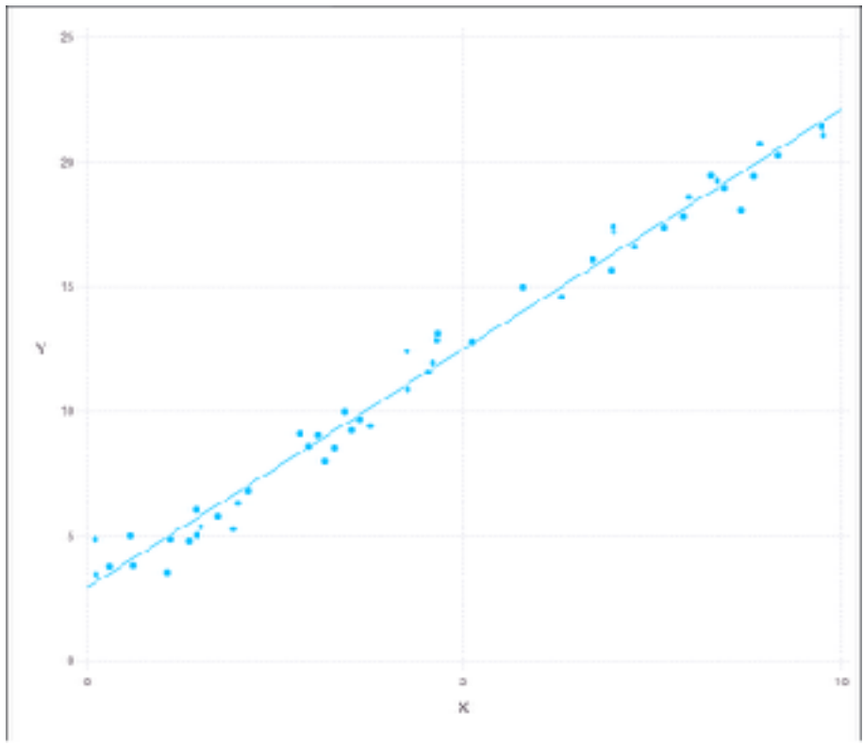
cor(Cost USD,pounds) == 0.998
cor(Cost USD,grams) == 0.998

Pearson's Correlation r Value Examples



Regression Line





Pearson’s Co

cor(x,y) == 0.992

What the line that minimizes the amount of residuals

Ordinary least squares

$$b = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Standard way to fit line to data

$$\mathrm{b} = \frac{\mathrm{cov}(\mathrm{X},\mathrm{Y})}{\mathrm{var}(\mathrm{X})}$$

$$a = \bar{y} - b\bar{x}$$

Spark MLlib Data Types

Vector	Dense
Matrix	Sparse

```
import org.apache.spark.mllib.linalg.{Vector, Vectors}

val dense: Vector = Vectors.dense(1.0, 0, 3.0)                    [1.0,0.0,3.0]

val sparse: Vector = Vectors.sparse(10, Array(0, 8), Array(5.1, 3.0))    (10,[0,8],[5.1,3.0])
```

index values

```
sparse.toArray
sparce.toDense

val sparseToo: Vector = Vectors.sparse(3, Seq((0, 1.0), (2, 3.0)))
```

But for ML need different imports

```
import org.apache.spark.ml.linalg.{Matrix, Vectors, Vector}
val dense: Vector = Vectors.dense(1.0, 0, 3.0)
```

Scala has aVector class so may better to use:

```
import org.apache.spark.ml.linalg.{Matrix, Vectors}
val dense = Vectors.dense(1.0, 0, 3.0)
```

```
import org.apache.spark.ml.linalg.{Matrix, Matrices}

val dm: Matrix = Matrices.dense(3, 2, Array(1.0, 2.0, 3.0, 4.0, 5.0, 6.0))
```

```
1.0  4.0
2.0  5.0
3.0  6.0
```

Regression Example

f(x) = x

f(x)	x
1	1
2	2
3	3
5	5
10	10
20	20
30	30
100	100
500	500

linearExact.svm

1 1:1
2 1:2
3 1:3
5 1:5
10 1:10
20 1:20
30 1:30
100 1:100
500 1:500

SVM, Libsvm, File format

SVM - Support Vector Machines

- Supervised learning models with learning algorithms
- Classification & regression

LIBSVM

- Popular machine learning library
- National Taiwan University
- Open source
- Code reused in other open source machine learning toolkits
  - scikit

File Format

<label> <index1>:<value1> <index2>:<value2> ...

Target Dependent variable

Starts at one

Reading the Data

val data = spark.read.format("libsvm").load("linearExact.csv")

data.show

```
+-----+---------------+
|label|       features|
+-----+---------------+
|  1.0|  (1,[0],[1.0])|
|  2.0|  (1,[0],[2.0])|
|  3.0|  (1,[0],[3.0])|
|  5.0|  (1,[0],[5.0])|
| 10.0| (1,[0],[10.0])|
| 20.0| (1,[0],[20.0])|
| 30.0| (1,[0],[30.0])|
|100.0|(1,[0],[100.0])|
|500.0|(1,[0],[500.0])|
+-----+---------------+
```

linearExact.svm

1 1:1

2 1:2

3 1:3

5 1:5

10 1:10

20 1:20

30 1:30

100 1:100

500 1:500

Computing Pearson Correlation

```
import org.apache.spark.ml.stat.Correlation
import org.apache.spark.sql.Row

val Row(coeff1: Matrix) = Correlation.corr(data, "features").head
println("Pearson correlation matrix:\n" + coeff1.toString)
```

```
Pearson correlation matrix:
1.0
```

Fitting the Data

```
val lrModel = lr.fit(data)
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
```

```
Coefficients: [0.998046433448248] Intercept: 0.145649239580626 1
```

```
val trainingSummary = lrModel.summary
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"r2: ${trainingSummary.r2}")
```

```
+--------------------+
|           residuals|
+--------------------+
| -0.1436956730288741|
|-0.1417421064771220 4|
|   -0.139788539925 37|
|-0.1358814068218663 4|
|-0.1261135740631065 3|
|-0.1065779085455851 3|
|-0.0870422430280619 6|
| 0.0497074155945824 7|
|  0.831134036295395 6|
+--------------------+

RMSE: 0.29941393003447314
r2: 0.9999961835777279
```

Data Formats, Multiple ways of Computing

linearExact.svm

1 1:1
2 1:2
3 1:3
5 1:5
10 1:10
20 1:20
30 1:30
100 1:100
500 1:500

linearExactSimple.csv

1,1
2,2
3,3
5,5
10,10
20,20
30,30
100,100
500,500

```
val linear = spark.read.format("csv").
        option("header",true).
        option("inferschema",true).
        load("linearExactSimple.csv")
linear.show
```

```
+---+---+
|  y|  x|
+---+---+
|  1|  1|
|  2|  2|
|  3|  3|
|  5|  5|
| 10| 10|
| 20| 20|
| 30| 30|
|100|100|
|500|500|
+---+---+
```

```
linear.stat.corr("y","x")
```

```
1.0
```

```
linear.agg( corr("y","x")).show
```

```
+----------+
|corr(y, x)|
+----------+
|       1.0|
+----------+
```

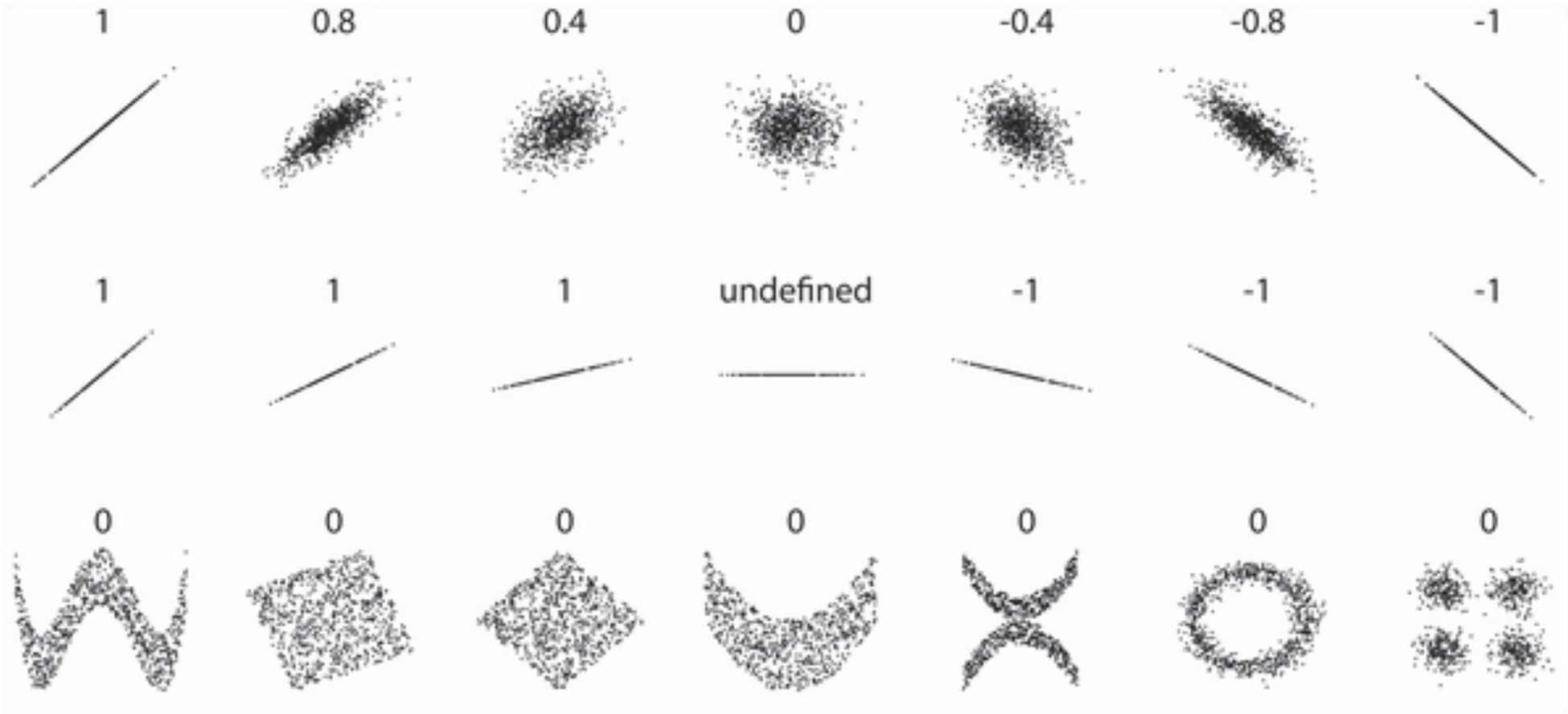
```
import org.apache.spark.sql.functions._
val withOnes = linear.withColumn("1",lit(1))
withOnes.show
```

```
+---+---+---+
|  y|  x|  1|
+---+---+---+
|  1|  1|  1|
|  2|  2|  1|
|  3|  3|  1|
|  5|  5|  1|
| 10| 10|  1|
| 20| 20|  1|
| 30| 30|  1|
|100|100|  1|
|500|500|  1|
+---+---+---+
```

```
withOnes.stat.corr("y","1")
```

```
NaN
```

Pearson's Correlation r Value Examples



```
var n = 10
val id = spark.range(1,n)
val randomX = id.withColumn("x", org.apache.spark.sql.functions.rand)
randomX.show
```

```
+---+-------------------+
| id|                  x|
+---+-------------------+
|  1|0.1384114992200986 5|
|  2| 0.022731920387712 1|
|  3| 0.954892566123584 1|
|  4| 0.421108673176546 2|
|  5| 0.538726573091720 3|
|  6| 0.865048671658372 8|
|  7|0.3958412208126712 6|
|  8|0.2141985344278276 6|
|  9|   0.393682609968 54|
+---+-------------------+
```

```
randomX.stat.corr("id","x")
```

```
0.1350213560785435 4
```

n	corr
10	0.135
10	0.122
20	-0.084
50	0.162
100	0.092
500	0.024

GPA & GRE Scores

```
val gre = spark.read.format("csv").
        option("header",true).
        option("inferschema",true).
        load("gpa-gre.csv")
gre.show(5)
```

600 students

```
+----+----+------+-----+
|Year| GPA|Verbal|Quant|
+----+----+------+-----+
|   1| 4.0|   420|  800|
|   1|3.88|   480|  770|
|   1|3.88|   480|  780|
|   1|3.87|   440|  690|
|   1|3.85|   320|  800|
+----+----+------+-----+
```

```
gre.stat.corr("GPA","Quant")

    0.2423456460373627 8
```

```
gre.stat.corr("Quant","GPA")

    0.2423456460373627 8
```

Take Away

Number of ways to compute Pearson's Correlation Coefficient

Must be important

Need to be able to transform data

Transformers, Estimators, Pipelines

Transformer

- Converts data
- Clean, add features, remove feature, format

Estimators

- Models or variations of same model

Evaluator

- See how an estimator performs

Pipeline

- Specifying transformers and estimators together