Session #1.1

Introduction to Machine Learning

Imagine we have a car classifieds website

Reduced Price

$115,000

Used 2018 Porsche 911 Carrera T

Brickell Luxury Motors

KBB.com Rating ★ 3.0

(22.75 mi. away)

(786) 899-4752

Get Directions | Confirm Availability

BLM

3 Similar vehicles in stock

View vehicles

Your Search Results

Sponsored

Used 2018 Porsche 911 Carrera

$149,998

Est. Finance Payment $2,050/mo.

See payment details

14,993 miles

HGreg Nissan Delray (24.5 mi. away)

KBB.com Rating ★ 4.5

(561) 926-5769 | Confirm Availability

Video Walkaround • Test Drive • Delivery • Online Paperwork

Accelerate My Deal

Sponsored

Certified 2021 Porsche 911 Carrera 4S

$174,975

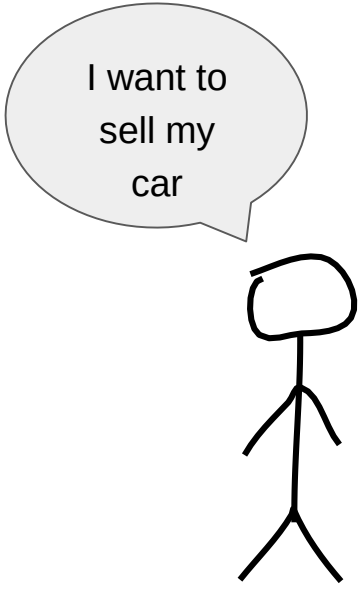
See estimated payment

6,298 miles

Porsche West Broward (10.7 mi. away)

(754) 755-5665 | Confirm Availability

Video Walkaround • Test Drive • Delivery



John Torcasio/unsplash (source)

Describe in detail

Enter a name *

toyota hilux

Add words to make your ad easier to find in the title.

Toyota
Transport / Cars

Photo
The first photo will be on the cover of the announcement. Drag to reorder.

Description*

Toyota Hilux, almost new

Price | Exchange

Price *

UAH

Required field

Price | Exchange

Price *

UAH

Required field

Price | Exchange

Price *

UAH

Required field

How can we help our user select the best price?

What do we know about cars?

					$1.1k
					$0.6k
					$23k

What do we know about cars?

	Year				Price
	1995				$1.1k
	1980				$0.6k
	2016				$23k

What do we know about cars?

	Year	Make			Price
	1995	GAZ			$1.1k
	1980	VAZ			$0.6k
	2016	BWM			$23k

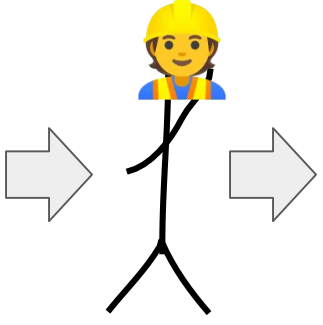
What do we know about cars?

	Year	Make	Mileage		Price
	1995	GAZ	200.000		$1.1k
	1980	VAZ	100.000		$0.6k
	2016	BWM	5.000		$23k

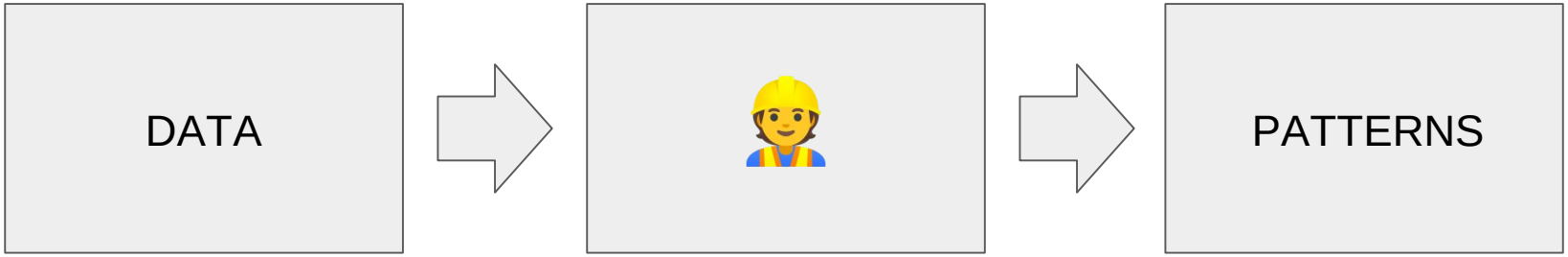
What do we know about cars?

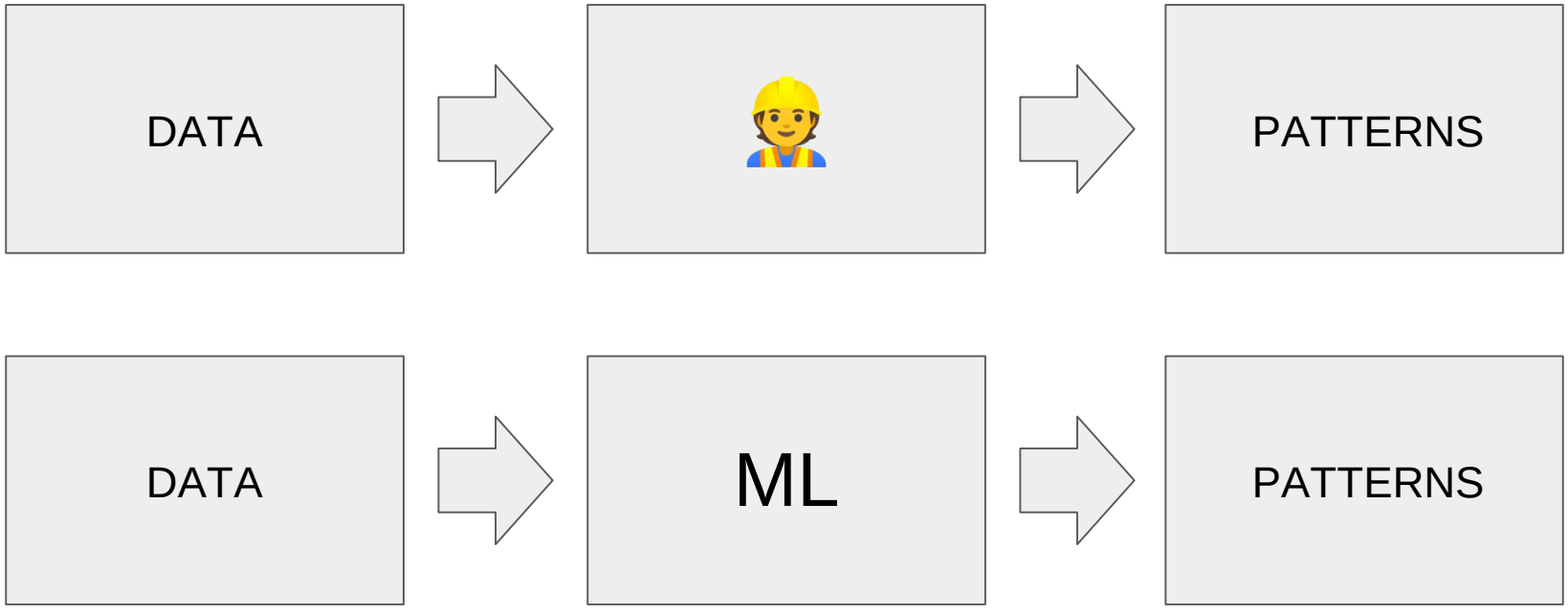
	Year	Make	Mileage	...	Price
	1995	GAZ	200.000	...	$1.1k
	1980	VAZ	100.000	...	$0.6k
	2016	BWM	5.000	...	$23k

	Year	Make	Mileage	...		Price
	1995	GAZ	200.000	...		$1.1k
	1980	VAZ	100.000	...		$0.6k
	2016	BWM	5.000	...		$23k



Using this information, an expert can determine the price


DATA
PATTERNS



If an expert can, so can a model!

	Year	Make	Mileage	...	Price
	1995	GAZ	200.000	...	$1.1k
	1980	VAZ	100.000	...	$0.6k
	2016	BWM	5.000	...	$23k

“Features”
what we know about cars

“Target”
what we want to predict

	Year	Make	Mileage	...	Price
	1995	GAZ	200.000	...	$1.1k
	1980	VAZ	100.000	...	$0.6k
	2016	BWM	5.000	...	$23k
🚗	...	...	...	...	...

Machine Learning

	Year	Make	Mileage	...
	1995	GAZ	200.000	...
	1980	VAZ	100.000	...
	2016	BWM	5.000	...
	...	...	...	...

“Features”

Price
$1.1k
$0.6k
$23k
...

“Target”

train

model

Using a model

	Year	Make	Mileage	...	Price
	1995	GAZ	200.000	...	$1.1k
	1980	VAZ	100.000	...	
	2016	BWM	5.000	...	
	...	...	...	...	...

model

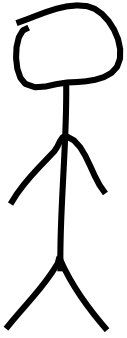
predict

Price
$1.5k
$0.4k
$20k
...

“Features”

“Target”

“Predictions”



Describe in detail

Enter a name *

toyota hilux

Add words to make your ad easier to find in the title.

Toyota
Transport / Cars

Photo — The first photo will be on the cover of the announcement. Drag to reorder.

Description*

Toyota Hilux, almost new

Price | Exchange

Price *

UAH

Required field

Describe in detail

Enter a name *

toyota hilux

Year	1995
Make	GAZ
Mileage	200.000
...	...

model

Describe in detail

Enter a name *

toyota hilux

Year	1995
Make	GAZ
Mileage	200.000
...	...

model

Price

Price *

50000

Create an ad

Describe in detail

Enter a name *

toyota hilux

Toyota
Transport / Cars

Photo
The first photo will be on the cover of the announcement. Drag to reorder.

Description*

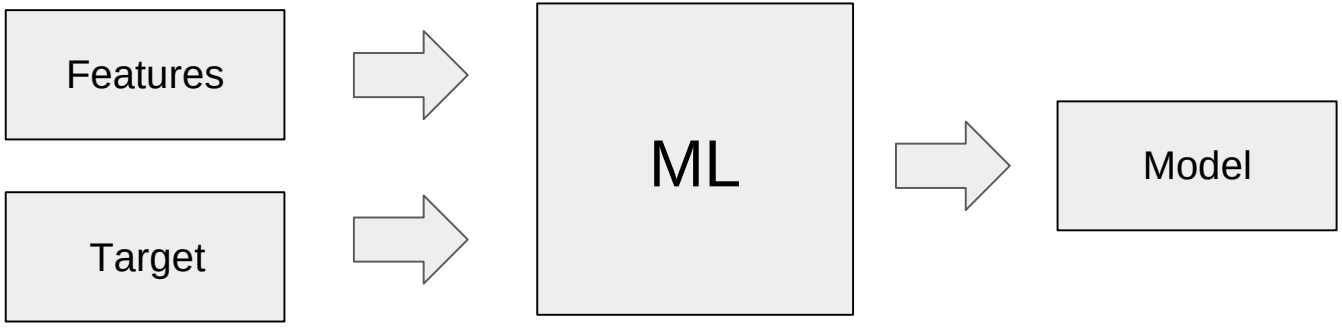
Toyota Hilux, almost new

Price | Exchange

Price *

50000 $

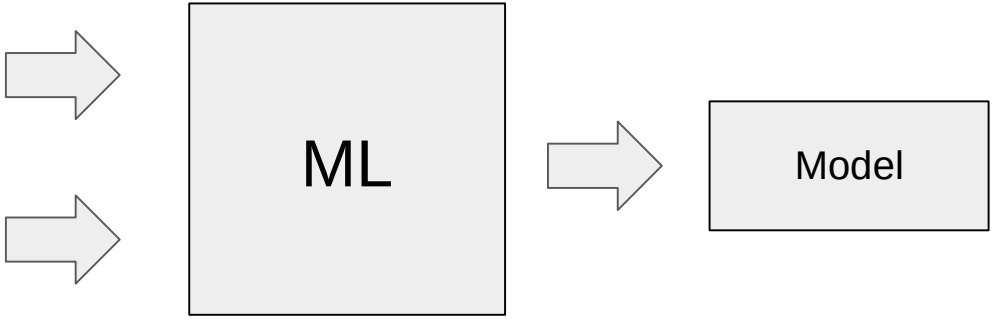
Model training



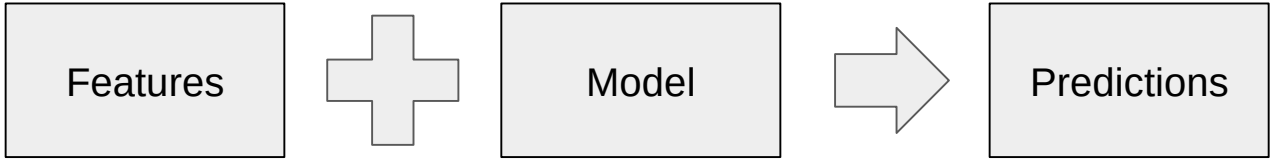
Model training

Year	Make	Mileage	...
1995	GAZ	200.000	...
1980	VAZ	100.000	...
2016	BWM	5.000	...
...	...	...	...

Price
$1.1k
$0.6k
$23k
...

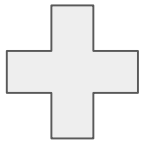


Predictions

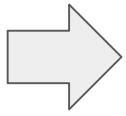


Predictions

Year	Make	Mileage	...
1996	Volvo	100.000	...
1991	GAZ	50.000	...
2018	Audi	2.000	...
...	...	...	...



Model



Price
$1.1k
$0.6k
$23k
...

Next

Machine Learning vs Rule-Based System

- Spam detection example

Exercise

Go to:

https://bit.ly/carData21

and from there, generate a dataset from a car of your choice. You will have a dataset that includes year, price, mileage.

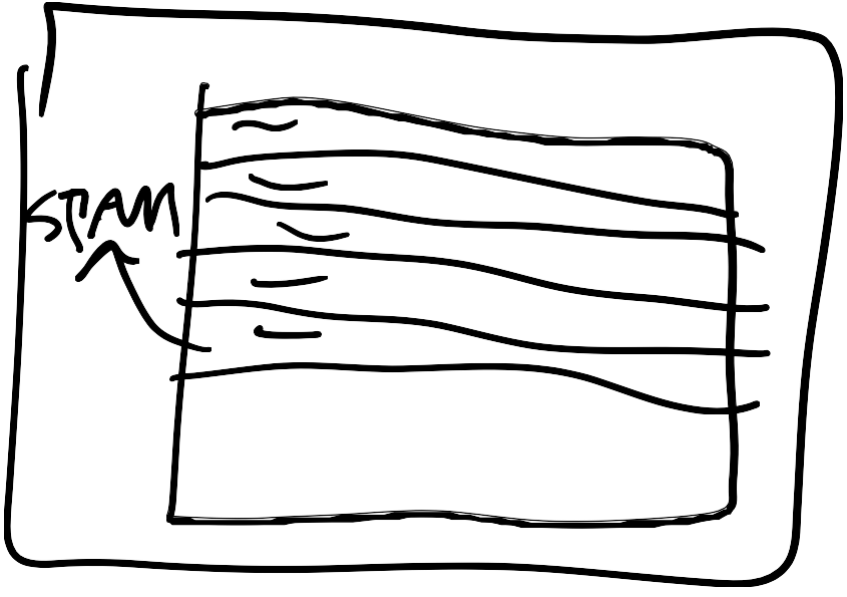
Session #1.2

Machine Learning vs Rule-Based Systems

Session #1.2: Plan

- A rule-based system for spam detection
- Using ML for spam detection
- Extracting features for ML

Email system



Spam

Subject: Get 50% off now
From: [redacted]

Wh[e]
spa
you[C]
clic
Pa[i]
wit[s]

[r]

[y]
[p]

Rules

- If sender = promotions@online.com then “spam”
- If title contains “tax review” and sender domain is “online.com” then “spam”
- Otherwise, “good email”

Code

```
def detect_spam(email):
    if email.sender == 'promotions@online.com':
        return SPAM
    if contains(email.title, ['tax', 'rewiew']) and
            domain(email.sender, 'online.com'):
        return SPAM
    return GOOD
```

More

Subject: Waiting for your reply
From: prince1@test.com

We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please [redacted] $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer.

Congratulations again!

Rules

- If sender = promotions@online.com then “spam”
- If title contains “tax review” and sender domain is “online.com” then “spam”
- **If body contains a word “deposit” then “spam”**
- Otherwise, “good email”

Code

```
def detect_spam(email):
    if email.sender == 'promotions@online.com':
        return SPAM
    if contains(email.title, ['tax', 'rewiew']) and
            domain(email.sender, 'online.com'):
        return SPAM
    if contains(email.body, ['deposit']):
        return SPAM
    return GOOD
```

More

Subject: Totally legit email
From: pedro@gmail.com

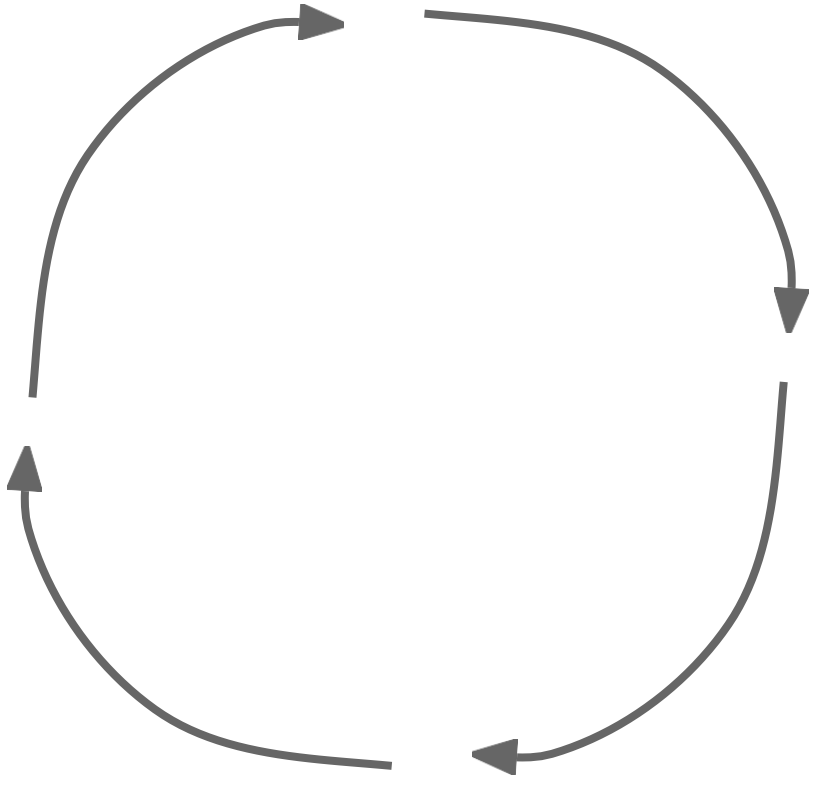
I transferred $50 to you one year ago, and now I'm moving out.
Please refund my [redacted]

Pedro.

Rules

- If sender = promotions@online.com then “spam”
- If title contains “tax review” and sender domain is “online.com” then “spam”
- If body contains a word “deposit”
  - **If sender domain is “test.com” then “spam”**
  - **If body >= 100 words then spam**
- Otherwise, “good email”

Repeat



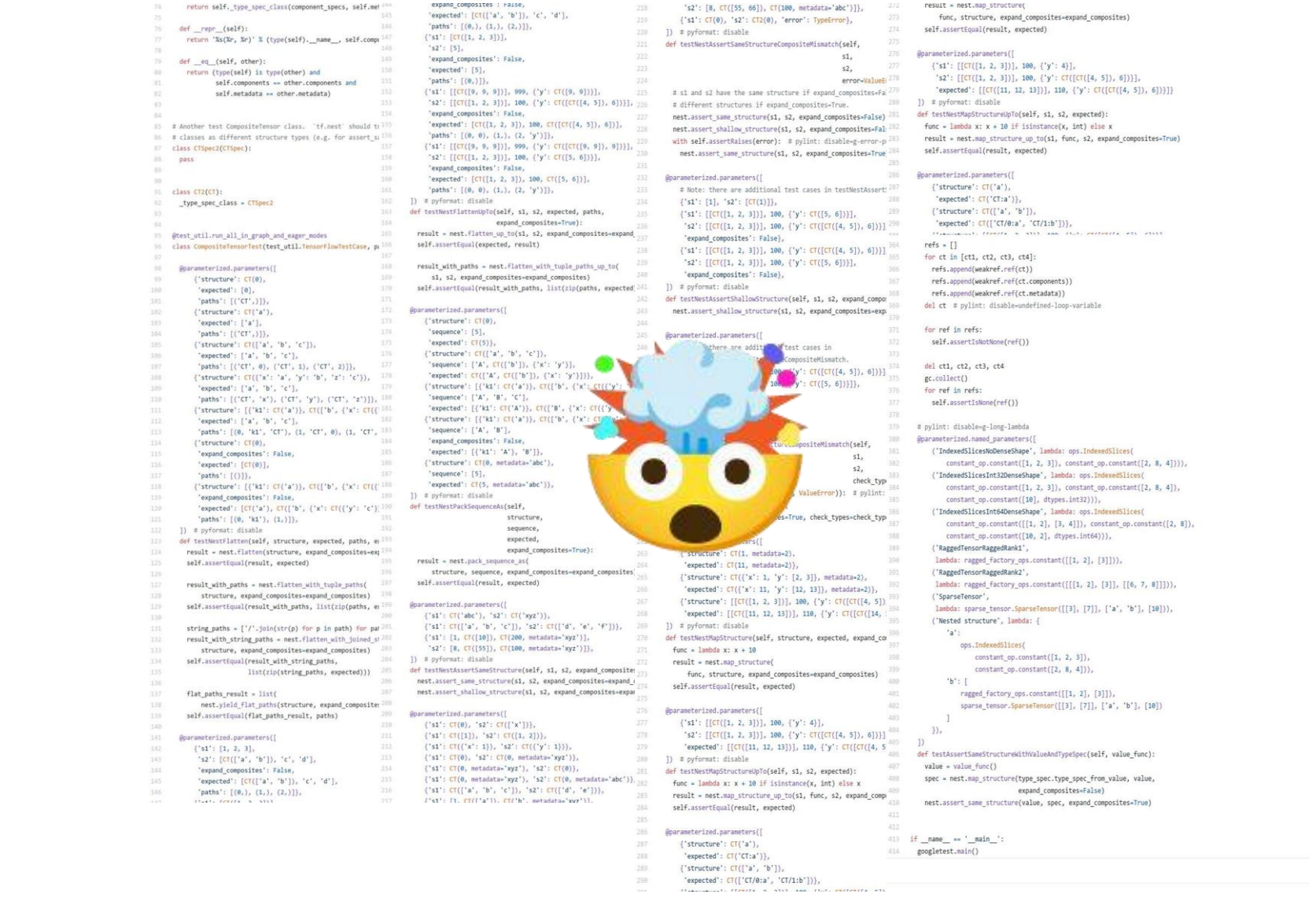
```
    return self._type_spec_class(component_specs, self.metadata)

  def __repr__(self):
    return '%s(%r, %r)' % (type(self).__name__, self.components, self.metadata)

  def __eq__(self, other):
    return (type(self) is type(other) and
            self.components == other.components and
            self.metadata == other.metadata)


# Another test CompositeTensor class.  `tf.nest` should treat different CT
# classes as different structure types (e.g. for assert_same_structure).
class CTSpec2(CTSpec):
  pass


class CT2(CT):
  _type_spec_class = CTSpec2


@test_util.run_all_in_graph_and_eager_modes
class CompositeTensorTest(test_util.TensorFlowTestCase, parameterized.TestCase):

  @parameterized.parameters([
      {'structure': CT(0),
       'expected': [0],
       'paths': [('CT',)]},
      {'structure': CT('a'),
       'expected': ['a'],
       'paths': [('CT',)]},
      {'structure': CT(['a', 'b', 'c']),
       'expected': ['a', 'b', 'c'],
       'paths': [('CT', 0), ('CT', 1), ('CT', 2)]},
      {'structure': CT({'x': 'a', 'y': 'b', 'z': 'c'}),
       'expected': ['a', 'b', 'c'],
       'paths': [('CT', 'x'), ('CT', 'y'), ('CT', 'z')]},
      {'structure': [{'k1': CT('a')}, CT(['b', {'x': CT({'y': 'c'})})]],
       'expected': ['a', 'b', 'c'],
       'paths': [(0, 'k1', 'CT'), (1, 'CT', 0), (1, 'CT', 1, 'x', 'CT', 'y')]},
      {'structure': CT(0),
       'expand_composites': False,
       'expected': [CT(0)],
       'paths': [()]},
      {'structure': [{'k1': CT('a')}, CT(['b', {'x': CT({'y': 'c'})})]],
       'expand_composites': False,
       'expected': [CT('a'), CT(['b', {'x': CT({'y': 'c'})})])],
       'paths': [(0, 'k1'), (1,)]},
  ])  # pyformat: disable
  def testNestFlatten(self, structure, expected, paths, expand_composites=True):
    result = nest.flatten(structure, expand_composites=expand_composites)
    self.assertEqual(result, expected)

    result_with_paths = nest.flatten_with_tuple_paths(
        structure, expand_composites=expand_composites)
    self.assertEqual(result_with_paths, list(zip(paths, expected)))

    string_paths = ['/'.join(str(p) for p in path) for path in paths]  # pylint: disable=g-complex-comprehension
    result_with_string_paths = nest.flatten_with_joined_string_paths(
        structure, expand_composites=expand_composites)
    self.assertEqual(result_with_string_paths,
                     list(zip(string_paths, expected)))

    flat_paths_result = list(
        nest.yield_flat_paths(structure, expand_composites=expand_composites))
    self.assertEqual(flat_paths_result, paths)

  @parameterized.parameters([
      {'s1': [1, 2, 3],
       's2': [CT(['a', 'b']), 'c', 'd'],
       'expand_composites': False,
       'expected': [CT(['a', 'b']), 'c', 'd'],
       'paths': [(0,), (1,), (2,)]},
```

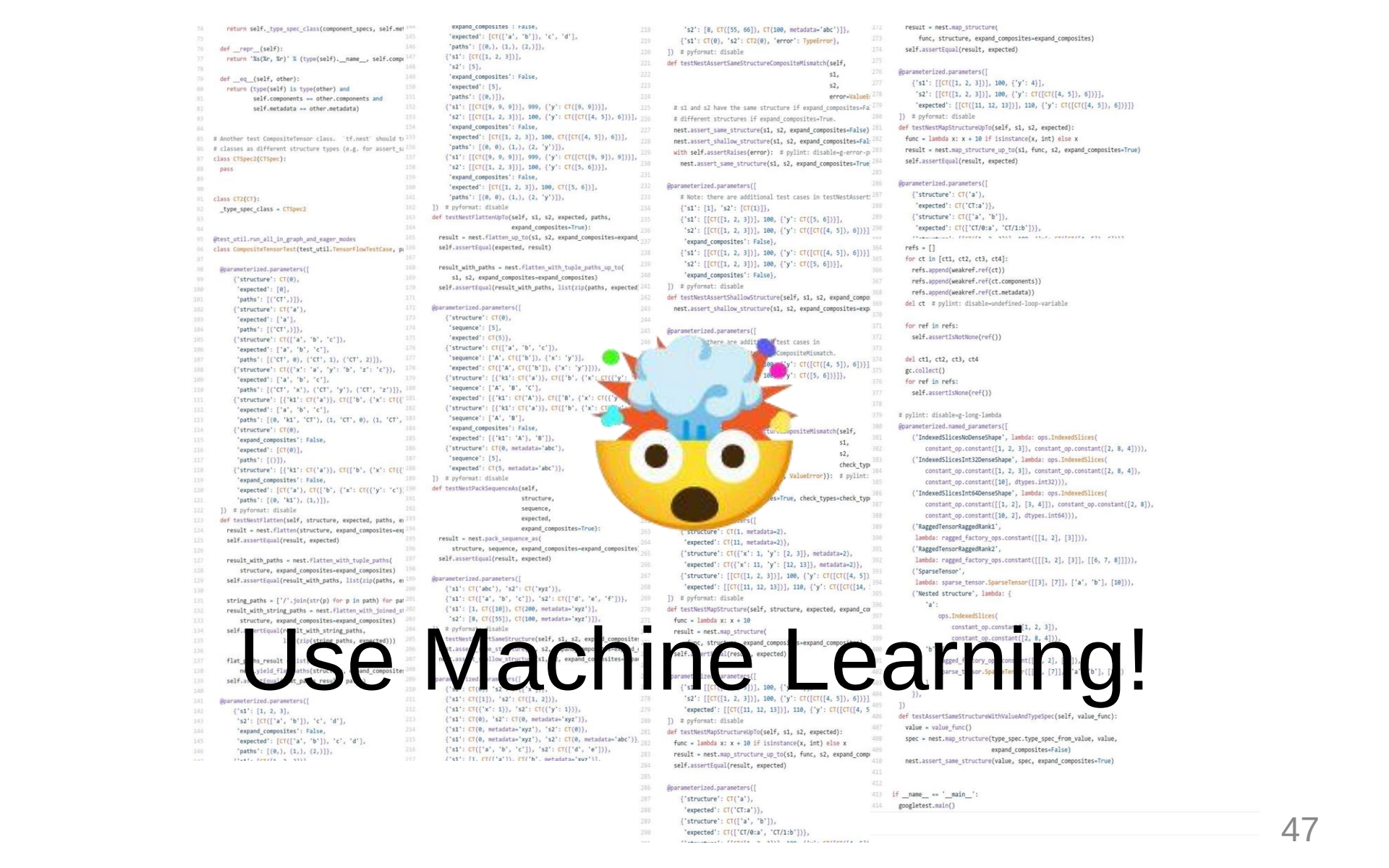
```
    return self._type_spec_class(component_specs, self.me

  def __repr__(self):
    return '%s(%r, %r)' % (type(self).__name__, self.compo

  def __eq__(self, other):
    return (type(self) is type(other) and
            self.components == other.components and
            self.metadata == other.metadata)


# Another test CompositeTensor class.  `tf.nest` should t
# classes as different structure types (e.g. for assert_s
class CTSpec2(CTSpec):
  pass


class CT2(CT):
  _type_spec_class = CTSpec2


@test_util.run_all_in_graph_and_eager_modes
class CompositeTensorTest(test_util.TensorFlowTestCase, pa

  @parameterized.parameters([
      {'structure': CT(0),
       'expected': [0],
       'paths': [('CT',)]},
      {'structure': CT('a'),
       'expected': ['a'],
       'paths': [('CT',)]},
      {'structure': CT(['a', 'b', 'c']),
       'expected': ['a', 'b', 'c'],
       'paths': [('CT', 0), ('CT', 1), ('CT', 2)]},
      {'structure': CT({'x': 'a', 'y': 'b', 'z': 'c'}),
       'expected': ['a', 'b', 'c'],
       'paths': [('CT', 'x'), ('CT', 'y'), ('CT', 'z')]},
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({
       'expected': ['a', 'b', 'c'],
       'paths': [(0, 'k1', 'CT'), (1, 'CT', 0), (1, 'CT',
      {'structure': CT(0),
       'expand_composites': False,
       'expected': [CT(0)],
       'paths': [()]},
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({
       'expand_composites': False,
       'expected': [CT('a'), CT({'b': {'x': CT({'y': 'c'}
       'paths': [(0, 'k1'), (1,)]},
  ])  # pyformat: disable
  def testNestFlatten(self, structure, expected, paths, e
    result = nest.flatten(structure, expand_composites=exp
    self.assertEqual(result, expected)

    result_with_paths = nest.flatten_with_tuple_paths(
        structure, expand_composites=expand_composites)
    self.assertEqual(result_with_paths, list(zip(paths, e

    string_paths = ['/'.join(str(p) for p in path) for pat
    result_with_string_paths = nest.flatten_with_joined_st
        structure, expand_composites=expand_composites)
    self.assertEqual(result_with_string_paths,
                     list(zip(string_paths, expected)))

    flat_paths_result = list(
        nest.yield_flat_paths(structure, expand_composite
    self.assertEqual(flat_paths_result, paths)

  @parameterized.parameters([
      {'s1': [1, 2, 3],
       's2': [CT(['a', 'b']), 'c', 'd'],
       'expand_composites': False,
       'expected': [CT(['a', 'b']), 'c', 'd'],
       'paths': [(0,), (1,), (2,)]},
```

```
       expand_composites': False,
       'expected': [CT(['a', 'b']), 'c', 'd'],
       'paths': [(0,), (1,), (2,)]},
      {'s1': [CT([1, 2, 3])],
       's2': [5],
       'expand_composites': False,
       'expected': [5],
       'paths': [(0,)]},
      {'s1': [[CT([9, 9, 9])], 999, {'y': CT([9, 9])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}],
       'expand_composites': False,
       'expected': [CT([1, 2, 3]), 100, CT([CT([4, 5]), 6])],
       'paths': [(0, 0), (1,), (2, 'y')]},
      {'s1': [[CT([9, 9, 9])], 999, {'y': CT([CT([9, 9]), 9])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([5, 6])}],
       'expand_composites': False,
       'expected': [CT([1, 2, 3]), 100, CT([5, 6])],
       'paths': [(0, 0), (1,), (2, 'y')]},
  ])  # pyformat: disable
  def testNestFlattenUpTo(self, s1, s2, expected, paths,
                          expand_composites=True):
    result = nest.flatten_up_to(s1, s2, expand_composites=expand_composites)
    self.assertEqual(expected, result)

    result_with_paths = nest.flatten_with_tuple_paths_up_to(
        s1, s2, expand_composites=expand_composites)
    self.assertEqual(result_with_paths, list(zip(paths, expected)))

  @parameterized.parameters([
      {'structure': CT(0),
       'sequence': [5],
       'expected': CT(5)},
      {'structure': CT(['a', 'b', 'c']),
       'sequence': ['A', CT(['b']), {'x': 'y'}],
       'expected': CT(['A', CT(['b']), {'x': 'y'}])},
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({'y': 'c'})}})],
       'sequence': ['A', 'B', 'C'],
       'expected': [{'k1': CT('A')}, CT({'B': {'x': CT({'y': 'C'})}})]},
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({'y': 'c'})}})],
       'sequence': ['A', 'B'],
       'expand_composites': False,
       'expected': [{'k1': 'A'}, 'B']},
      {'structure': CT(0, metadata='abc'),
       'sequence': [5],
       'expected': CT(5, metadata='abc')},
  ])  # pyformat: disable
  def testNestPackSequenceAs(self,
                             structure,
                             sequence,
                             expected,
                             expand_composites=True):
    result = nest.pack_sequence_as(
        structure, sequence, expand_composites=expand_composites)
    self.assertEqual(result, expected)

  @parameterized.parameters([
      {'s1': CT('abc'), 's2': CT('xyz')},
      {'s1': CT(['a', 'b', 'c']), 's2': CT(['d', 'e', 'f'])},
      {'s1': [1, CT([10]), CT(200, metadata='xyz')],
       's2': [8, CT([55]), CT(100, metadata='xyz')]},
  ])  # pyformat: disable
  def testNestAssertSameStructure(self, s1, s2, expand_composites=True):
    nest.assert_same_structure(s1, s2, expand_composites=expand_composites)
    nest.assert_shallow_structure(s1, s2, expand_composites=expand_composites)

  @parameterized.parameters([
      {'s1': CT(0), 's2': CT(['x'])},
      {'s1': CT([1]), 's2': CT([1, 2])},
      {'s1': CT({'x': 1}), 's2': CT({'y': 1})},
      {'s1': CT(0), 's2': CT(0, metadata='xyz')},
      {'s1': CT(0, metadata='xyz'), 's2': CT(0)},
      {'s1': CT(0, metadata='xyz'), 's2': CT(0, metadata='abc')},
      {'s1': CT(['a', 'b', 'c']), 's2': CT(['d', 'e'])},
      {'s1': [1, CT('a')], CT('b', metadata='xyz')},
```

```
    return self._type_spec_class(component_specs, self.metadata)

  def __repr__(self):
    return '%s(%r, %r)' % (type(self).__name__, self.components,

  def __eq__(self, other):
    return (type(self) is type(other) and
            self.components == other.components and
            self.metadata == other.metadata)


# Another test CompositeTensor class.  `tf.nest` should tr
# classes as different structure types (e.g. for assert_sa
class CTSpec2(CTSpec):
  pass


class CT2(CT):
  _type_spec_class = CTSpec2


@test_util.run_all_in_graph_and_eager_modes
class CompositeTensorTest(test_util.TensorFlowTestCase, pa

  @parameterized.parameters([
      {'structure': CT(0),
       'expected': [0],
       'paths': [('CT',)]},
      {'structure': CT('a'),
       'expected': ['a'],
       'paths': [('CT',)]},
      {'structure': CT(['a', 'b', 'c']),
       'expected': ['a', 'b', 'c'],
       'paths': [('CT', 0), ('CT', 1), ('CT', 2)]},
      {'structure': CT({'x': 'a', 'y': 'b', 'z': 'c'}),
       'expected': ['a', 'b', 'c'],
       'paths': [('CT', 'x'), ('CT', 'y'), ('CT', 'z')]},
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({'
       'expected': ['a', 'b', 'c'],
       'paths': [(0, 'k1', 'CT'), (1, 'CT', 0), (1, 'CT',
      {'structure': CT(0),
       'expand_composites': False,
       'expected': [CT(0)],
       'paths': [()]},
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({'
       'expand_composites': False,
       'expected': [CT('a'), CT({'b': {'x': CT({'y': 'c'}
       'paths': [(0, 'k1'), (1,)]},
  ])  # pyformat: disable
  def testNestFlatten(self, structure, expected, paths, ex
    result = nest.flatten(structure, expand_composites=exp
    self.assertEqual(result, expected)

    result_with_paths = nest.flatten_with_tuple_paths(
        structure, expand_composites=expand_composites)
    self.assertEqual(result_with_paths, list(zip(paths, ex

    string_paths = ['/'.join(str(p) for p in path) for pat
    result_with_string_paths = nest.flatten_with_joined_st
        structure, expand_composites=expand_composites)
    self.assertEqual(result_with_string_paths,
                     list(zip(string_paths, expected)))

    flat_paths_result = list(
        nest.yield_flat_paths(structure, expand_composite
    self.assertEqual(flat_paths_result, paths)

  @parameterized.parameters([
      {'s1': [1, 2, 3],
       's2': [CT(['a', 'b']), 'c', 'd'],
       'expand_composites': False,
       'expected': [CT(['a', 'b']), 'c', 'd'],
       'paths': [(0,), (1,), (2,)]},
      {'s1': [CT([1, 2, 3])],
       's2': [5],
       'expand_composites': False,
       'expected': [5],
       'paths': [(0,)]},
      {'s1': [[CT([9, 9, 9])], 999, {'y': CT([9, 9])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}],
       'expand_composites': False,
       'expected': [CT([1, 2, 3]), 100, CT([CT([4, 5]), 6])],
       'paths': [(0, 0), (1,), (2, 'y')]},
      {'s1': [[CT([9, 9, 9])], 999, {'y': CT([CT([9, 9]), 9])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([5, 6])}],
       'expand_composites': False,
       'expected': [CT([1, 2, 3]), 100, CT([5, 6])],
       'paths': [(0, 0), (1,), (2, 'y')]},
  ])  # pyformat: disable
  def testNestFlattenUpTo(self, s1, s2, expected, paths,
                          expand_composites=True):
    result = nest.flatten_up_to(s1, s2, expand_composites=expand_
    self.assertEqual(expected, result)

    result_with_paths = nest.flatten_with_tuple_paths_up_to(
        s1, s2, expand_composites=expand_composites)
    self.assertEqual(result_with_paths, list(zip(paths, expected

  @parameterized.parameters([
      {'structure': CT(0),
       'sequence': [5],
       'expected': CT(5)},
      {'structure': CT(['a', 'b', 'c']),
       'sequence': ['A', CT(['b']), {'x': 'y'}],
       'expected': CT(['A', CT(['b']), {'x': 'y'}])},
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({'y': 'c
       'sequence': ['A', 'B', 'C'],
       'expected': [{'k1': CT('A')}, CT({'B': {'x': CT({'y': 'C'
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({'y': 'c
       'sequence': ['A', 'B'],
       'expand_composites': False,
       'expected': [{'k1': 'A'}, 'B']},
      {'structure': CT(0, metadata='abc'),
       'sequence': [5],
       'expected': CT(5, metadata='abc')},
  ])  # pyformat: disable
  def testNestPackSequenceAs(self,
                             structure,
                             sequence,
                             expected,
                             expand_composites=True):
    result = nest.pack_sequence_as(
        structure, sequence, expand_composites=expand_composites
    self.assertEqual(result, expected)

  @parameterized.parameters([
      {'s1': CT('abc'), 's2': CT('xyz')},
      {'s1': CT(['a', 'b', 'c']), 's2': CT(['d', 'e', 'f'])},
      {'s1': [1, CT([10]), CT(200, metadata='xyz')],
       's2': [8, CT([55]), CT(100, metadata='xyz')]},
  ])  # pyformat: disable
  def testNestAssertSameStructure(self, s1, s2, expand_composite
    nest.assert_same_structure(s1, s2, expand_composites=expand_
    nest.assert_shallow_structure(s1, s2, expand_composites=expa

  @parameterized.parameters([
      {'s1': CT(0), 's2': CT('x')},
      {'s1': CT([1]), 's2': CT([1, 2])},
      {'s1': CT({'x': 1}), 's2': CT({'y': 1})},
      {'s1': CT(0), 's2': CT(0, metadata='xyz')},
      {'s1': CT(0, metadata='xyz'), 's2': CT(0)},
      {'s1': CT(0, metadata='xyz'), 's2': CT(0, metadata='abc')},
      {'s1': CT(['a', 'b', 'c']), 's2': CT(['d', 'e'])},
      {'s1': [1, CT(['a']), CT('b', metadata='xyz')],
       's2': [8, CT([55, 66]), CT(100, metadata='abc')]},
      {'s1': CT(0), 's2': CT2(0), 'error': TypeError},
  ])  # pyformat: disable
  def testNestAssertSameStructureCompositeMismatch(self,
                                                   s1,
                                                   s2,
                                                   error=ValueError):
    # s1 and s2 have the same structure if expand_composites=False; but
    # different structures if expand_composites=True.
    nest.assert_same_structure(s1, s2, expand_composites=False)
    nest.assert_shallow_structure(s1, s2, expand_composites=False)
    with self.assertRaises(error):  # pylint: disable=g-error-prone-assert-raises
      nest.assert_same_structure(s1, s2, expand_composites=True)

  @parameterized.parameters([
      # Note: there are additional test cases in testNestAssertSameStructure.
      {'s1': [1], 's2': [CT(1)]},
      {'s1': [[CT([1, 2, 3])], 100, {'y': CT([5, 6])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}],
       'expand_composites': False},
      {'s1': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([5, 6])}],
       'expand_composites': False},
  ])  # pyformat: disable
  def testNestAssertShallowStructure(self, s1, s2, expand_composites=True):
    nest.assert_shallow_structure(s1, s2, expand_composites=expand_composites)

  @parameterized.parameters([
      # Note: there are additional test cases in
      # testNestAssertSameStructureCompositeMismatch.
      {'s1': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([5, 6])}]},
      {'s1': CT([1, 2, 3]),
       's2': [1, 2, 3],
       'check_types': False},
  ])  # pyformat: disable
  def testNestAssertShallowStructureCompositeMismatch(self,
                                                      s1,
                                                      s2,
                                                      check_types=True):
    with self.assertRaises((TypeError, ValueError)):  # pylint: disable=g-error-prone-assert-raises
      nest.assert_shallow_structure(
          s1, s2, expand_composites=True, check_types=check_types)

  @parameterized.parameters([
      {'structure': CT(1, metadata=2),
       'expected': CT(11, metadata=2)},
      {'structure': CT({'x': 1, 'y': [2, 3]}, metadata=2),
       'expected': CT({'x': 11, 'y': [12, 13]}, metadata=2)},
      {'structure': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}],
       'expected': [[CT([11, 12, 13])], 110, {'y': CT([CT([14, 15]), 16])}]},
  ])  # pyformat: disable
  def testNestMapStructure(self, structure, expected, expand_composites=True):
    func = lambda x: x + 10
    result = nest.map_structure(
        func, structure, expand_composites=expand_composites)
    self.assertEqual(result, expected)

  @parameterized.parameters([
      {'s1': [[CT([1, 2, 3])], 100, {'y': 4}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}],
       'expected': [[CT([11, 12, 13])], 110, {'y': CT([CT([4, 5]), 6])}]}
  ])  # pyformat: disable
  def testNestMapStructureUpTo(self, s1, s2, expected):
    func = lambda x: x + 10 if isinstance(x, int) else x
    result = nest.map_structure_up_to(s1, func, s2, expand_composites=True)
    self.assertEqual(result, expected)

  @parameterized.parameters([
      {'structure': CT('a'),
       'expected': CT('CT:a')},
      {'structure': CT(['a', 'b']),
       'expected': CT(['CT/0:a', 'CT/1:b'])},
```

```
    return self._type_spec_class(component_specs, self.me

  def __repr__(self):
    return '%s(%r, %r)' % (type(self).__name__, self.compo

  def __eq__(self, other):
    return (type(self) is type(other) and
            self.components == other.components and
            self.metadata == other.metadata)


# Another test CompositeTensor class.  `tf.nest` should t
# classes as different structure types (e.g. for assert_s
class CTSpec2(CTSpec):
  pass


class CT2(CT):
  _type_spec_class = CTSpec2

@test_util.run_all_in_graph_and_eager_modes
class CompositeTensorTest(test_util.TensorFlowTestCase, p

  @parameterized.parameters([
      {'structure': CT(0),
       'expected': [0],
       'paths': [('CT',)]},
      {'structure': CT('a'),
       'expected': ['a'],
       'paths': [('CT',)]},
      {'structure': CT(['a', 'b', 'c']),
       'expected': ['a', 'b', 'c'],
       'paths': [('CT', 0), ('CT', 1), ('CT', 2)]},
      {'structure': CT({'x': 'a', 'y': 'b', 'z': 'c'}),
       'expected': ['a', 'b', 'c'],
       'paths': [('CT', 'x'), ('CT', 'y'), ('CT', 'z')]},
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({
       'expected': ['a', 'b', 'c'],
       'paths': [(0, 'k1', 'CT'), (1, 'CT', 0), (1, 'CT',
      {'structure': CT(0),
       'expand_composites': False,
       'expected': [CT(0)],
       'paths': [()]},
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({
       'expand_composites': False,
       'expected': [CT('a'), CT({'b': {'x': CT({'y': 'c'}
       'paths': [(0, 'k1'), (1,)]},
  ])  # pyformat: disable
  def testNestFlatten(self, structure, expected, paths, e
    result = nest.flatten(structure, expand_composites=exp
    self.assertEqual(result, expected)

    result_with_paths = nest.flatten_with_tuple_paths(
        structure, expand_composites=expand_composites)
    self.assertEqual(result_with_paths, list(zip(paths, e

    string_paths = ['/'.join(str(p) for p in path) for pat
    result_with_string_paths = nest.flatten_with_joined_st
        structure, expand_composites=expand_composites)
    self.assertEqual(result_with_string_paths,
                     list(zip(string_paths, expected)))

    flat_paths_result = list(
        nest.yield_flat_paths(structure, expand_composite
    self.assertEqual(flat_paths_result, paths)

  @parameterized.parameters([
      {'s1': [1, 2, 3],
       's2': [CT(['a', 'b']), 'c', 'd'],
       'expand_composites': False,
       'expected': [CT(['a', 'b']), 'c', 'd'],
       'paths': [(0,), (1,), (2,)]},
      {'s1': [CT([1, 2, 3])],
       's2': [5],
       'expand_composites': False,
       'expected': [5],
       'paths': [(0,)]},
      {'s1': [[CT([9, 9, 9])], 999, {'y': CT([9, 9])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}],
       'expand_composites': False,
       'expected': [CT([1, 2, 3]), 100, CT([CT([4, 5]), 6])],
       'paths': [(0, 0), (1,), (2, 'y')]},
      {'s1': [[CT([9, 9, 9])], 999, {'y': CT([CT([9, 9])], 9])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([5, 6])}],
       'expand_composites': False,
       'expected': [CT([1, 2, 3]), 100, CT([5, 6])],
       'paths': [(0, 0), (1,), (2, 'y')]},
  ])  # pyformat: disable
  def testNestFlattenUpTo(self, s1, s2, expected, paths,
                          expand_composites=True):
    result = nest.flatten_up_to(s1, s2, expand_composites=expand_
    self.assertEqual(expected, result)

    result_with_paths = nest.flatten_with_tuple_paths_up_to(
        s1, s2, expand_composites=expand_composites)
    self.assertEqual(result_with_paths, list(zip(paths, expected

  @parameterized.parameters([
      {'structure': CT(0),
       'sequence': [5],
       'expected': CT(5)},
      {'structure': CT(['a', 'b', 'c']),
       'sequence': ['A', CT(['b']), {'x': 'y'}],
       'expected': CT(['A', CT(['b']), {'x': 'y'}])},
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({'y': 'c
       'sequence': ['A', 'B', 'C'],
       'expected': [{'k1': CT('A')}, CT({'B': {'x': CT({'y': 'C'
      {'structure': [{'k1': CT('a')}, CT({'b': {'x': CT({'y': 'c
       'sequence': ['A', 'B'],
       'expand_composites': False,
       'expected': [{'k1': 'A'}, 'B']},
      {'structure': CT(0, metadata='abc'),
       'sequence': [5],
       'expected': CT(5, metadata='abc')},
  ])  # pyformat: disable
  def testNestPackSequenceAs(self,
                             structure,
                             sequence,
                             expected,
                             expand_composites=True):
    result = nest.pack_sequence_as(
        structure, sequence, expand_composites=expand_composites
    self.assertEqual(result, expected)

  @parameterized.parameters([
      {'s1': CT('abc'), 's2': CT('xyz')},
      {'s1': CT(['a', 'b', 'c']), 's2': CT(['d', 'e', 'f'])},
      {'s1': [1, CT([10]), CT(200, metadata='xyz')],
       's2': [8, CT([55]), CT(100, metadata='xyz')]},
  ])  # pyformat: disable
  def testNestAssertSameStructure(self, s1, s2, expand_composite
    nest.assert_same_structure(s1, s2, expand_composites=expand_
    nest.assert_shallow_structure(s1, s2, expand_composites=expa

  @parameterized.parameters([
      {'s1': CT(0), 's2': CT(['x'])},
      {'s1': CT([1]), 's2': CT([1, 2])},
      {'s1': CT({'x': 1}), 's2': CT({'y': 1})},
      {'s1': CT(0), 's2': CT(0, metadata='xyz')},
      {'s1': CT(0, metadata='xyz'), 's2': CT(0)},
      {'s1': CT(0, metadata='xyz'), 's2': CT(0, metadata='abc')},
      {'s1': CT(['a', 'b', 'c']), 's2': CT(['d', 'e'])},
      {'s1': [1, CT(['a']), CT('b', metadata='xyz')],
       's2': [8, CT([55, 66]), CT(100, metadata='abc')]},
      {'s1': CT(0), 's2': CT2(0), 'error': TypeError},
  ])  # pyformat: disable
  def testNestAssertSameStructureCompositeMismatch(self,
                                                   s1,
                                                   s2,
                                                   error=ValueE
    # s1 and s2 have the same structure if expand_composites=Fa
    # different structures if expand_composites=True.
    nest.assert_same_structure(s1, s2, expand_composites=False)
    nest.assert_shallow_structure(s1, s2, expand_composites=Fal
    with self.assertRaises(error):  # pylint: disable=g-error-p
      nest.assert_same_structure(s1, s2, expand_composites=True

  @parameterized.parameters([
      # Note: there are additional test cases in testNestAssert
      {'s1': [1], 's2': [CT(1)]},
      {'s1': [[CT([1, 2, 3])], 100, {'y': CT([5, 6])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}]
       'expand_composites': False},
      {'s1': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}]
       's2': [[CT([1, 2, 3])], 100, {'y': CT([5, 6])}],
       'expand_composites': False},
  ])  # pyformat: disable
  def testNestAssertShallowStructure(self, s1, s2, expand_compo
    nest.assert_shallow_structure(s1, s2, expand_composites=exp

  @parameterized.parameters([
      # Note: there are additional test cases in
      # testNestAssertSameStructureCompositeMismatch.
      {'s1': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([5, 6])}]},
      {'s1': CT([1, 2, 3]),
       's2': [1, 2, 3],
       'check_types': False},
  ])  # pyformat: disable
  def testNestAssertShallowStructureCompositeMismatch(self,
                                                      s1,
                                                      s2,
                                                      check_typ
    with self.assertRaises((TypeError, ValueError)):  # pylint:
      nest.assert_shallow_structure(
          s1, s2, expand_composites=True, check_types=check_typ

  @parameterized.parameters([
      {'structure': CT(1, metadata=2),
       'expected': CT(11, metadata=2)},
      {'structure': CT({'x': 1, 'y': [2, 3]}, metadata=2),
       'expected': CT({'x': 11, 'y': [12, 13]}, metadata=2)},
      {'structure': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5])
       'expected': [[CT([11, 12, 13])], 110, {'y': CT([CT([14,
  ])  # pyformat: disable
  def testNestMapStructure(self, structure, expected, expand_co
    func = lambda x: x + 10
    result = nest.map_structure(
        func, structure, expand_composites=expand_composites)
    self.assertEqual(result, expected)

  @parameterized.parameters([
      {'s1': [[CT([1, 2, 3])], 100, {'y': 4}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}]
       'expected': [[CT([11, 12, 13])], 110, {'y': CT([CT([4, 5
  ])  # pyformat: disable
  def testNestMapStructureUpTo(self, s1, s2, expected):
    func = lambda x: x + 10 if isinstance(x, int) else x
    result = nest.map_structure_up_to(s1, func, s2, expand_comp
    self.assertEqual(result, expected)

  @parameterized.parameters([
      {'structure': CT('a'),
       'expected': CT('CT:a')},
      {'structure': CT(['a', 'b']),
       'expected': CT(['CT/0:a', 'CT/1:b'])},
    result = nest.map_structure(
        func, structure, expand_composites=expand_composites)
    self.assertEqual(result, expected)

  @parameterized.parameters([
      {'s1': [[CT([1, 2, 3])], 100, {'y': 4}],
       's2': [[CT([1, 2, 3])], 100, {'y': CT([CT([4, 5]), 6])}],
       'expected': [[CT([11, 12, 13])], 110, {'y': CT([CT([4, 5]), 6])}]}
  ])  # pyformat: disable
  def testNestMapStructureUpTo(self, s1, s2, expected):
    func = lambda x: x + 10 if isinstance(x, int) else x
    result = nest.map_structure_up_to(s1, func, s2, expand_composites=True)
    self.assertEqual(result, expected)

  @parameterized.parameters([
      {'structure': CT('a'),
       'expected': CT('CT:a')},
      {'structure': CT(['a', 'b']),
       'expected': CT(['CT/0:a', 'CT/1:b'])},
    refs = []
    for ct in [ct1, ct2, ct3, ct4]:
      refs.append(weakref.ref(ct))
      refs.append(weakref.ref(ct.components))
      refs.append(weakref.ref(ct.metadata))
    del ct  # pylint: disable=undefined-loop-variable

    for ref in refs:
      self.assertIsNotNone(ref())

    del ct1, ct2, ct3, ct4
    gc.collect()
    for ref in refs:
      self.assertIsNone(ref())

  # pylint: disable=g-long-lambda
  @parameterized.named_parameters([
      ('IndexedSlicesNoDenseShape', lambda: ops.IndexedSlices(
          constant_op.constant([1, 2, 3]), constant_op.constant([2, 8, 4]))),
      ('IndexedSlicesInt32DenseShape', lambda: ops.IndexedSlices(
          constant_op.constant([1, 2, 3]), constant_op.constant([2, 8, 4]),
          constant_op.constant([10], dtypes.int32))),
      ('IndexedSlicesInt64DenseShape', lambda: ops.IndexedSlices(
          constant_op.constant([[1, 2], [3, 4]]), constant_op.constant([2, 8]),
          constant_op.constant([10, 2], dtypes.int64))),
      ('RaggedTensorRaggedRank1',
       lambda: ragged_factory_ops.constant([[1, 2], [3]])),
      ('RaggedTensorRaggedRank2',
       lambda: ragged_factory_ops.constant([[[1, 2], [3], [6, 7, 8]]])),
      ('SparseTensor',
       lambda: sparse_tensor.SparseTensor([[3], [7]], ['a', 'b'], [10])),
      ('Nested structure', lambda: {
          'a':
              ops.IndexedSlices(
                  constant_op.constant([1, 2, 3]),
                  constant_op.constant([2, 8, 4])),
          'b': [
              ragged_factory_ops.constant([[1, 2], [3]]),
              sparse_tensor.SparseTensor([[3], [7]], ['a', 'b'], [10])
          ]
      }),
  ])
  def testAssertSameStructureWithValueAndTypeSpec(self, value_func):
    value = value_func()
    spec = nest.map_structure(type_spec.type_spec_from_value, value,
                              expand_composites=False)
    nest.assert_same_structure(value, spec, expand_composites=True)


if __name__ == '__main__':
  googletest.main()
```




Use Machine Learning!

Machine Learning

- Get data
- Define & calculate features
- Train and use the model

Getting data

SPAM

Subject: Waiting for your reply
From: prince1@test.com

We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please deposit $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer.

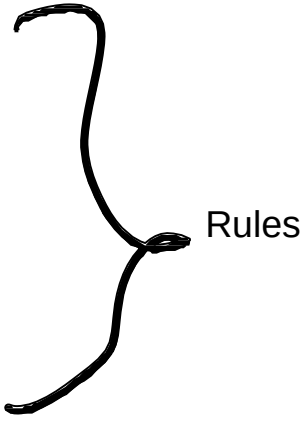
Congratulations again!

Machine Learning

- Get data
- **Define & calculate features**
- Train and use the model

Features

- Length of title > 10? true/false
- Length of body > 10? true/false
- Sender [redacted] true/false
- Sender “hpYOSKmL@test.com”? true/false
- Sender domain [redacted] true/false
- Description contains “[redacted] true/false



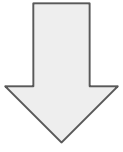
Start with rules and then use these rules as features

Subject: Waiting for your reply
From: prince1@test.com

We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please deposit $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer.

Congratulations again!

SPAM



$$[1, \quad 1, \quad 0, \quad 0, \quad 1, \quad 1]$$

Subject: Waiting for your reply
From: prince1@test.com

We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please deposit $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer.

Congratulations again!

SPAM

Length of title > 10? **True**

T

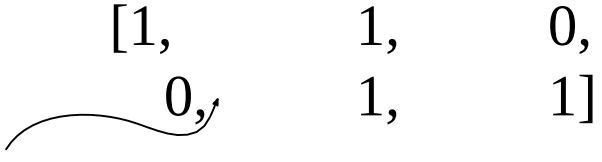
[1, 1, 0,
0, 1,
1]

Subject: Waiting for your reply
From: prince1@test.com

We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please deposit $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer.

Congratulations again!

SPAM



Length of body > 10? **True**

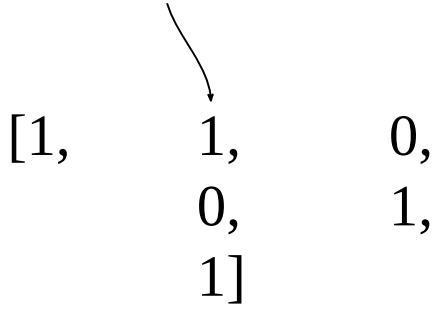
Subject: Waiting for your reply
From: prince1@test.com

We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please deposit $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer.

Congratulations again!

SPAM

Sender “promotions@online.com”? **False**



Subject: Waiting for your reply
From: prince1@test.com

We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please deposit $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer.

Congratulations again!

SPAM

$$[1,\ 1,\ 0,\ 0,\ 1,\ 1]$$

Sender “hpYOSKmL@test.com”? **False**

Subject: Waiting for your reply
From: prince1@

We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please deposit $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer.

Congratulations again!

SPAM

Sender domain “test.com”? **True**

[1, 1, , 0, 1, 1]

Subject: Waiting for your reply
From: prince1@test.com

We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer.

Congratulations again!

SPAM

[1, 1, 0, 1, 0, 1, 0, 1]

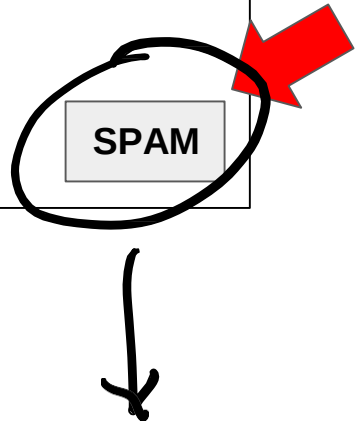
TRUE

Description contains “deposit”? ~~False~~

Subject: Waiting for your reply
From: prince1@test.com

We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please deposit $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer.

Congratulations again!



$$[1, \quad 1, \quad 0, \quad 1, \quad 0, \quad 1, \quad 1]$$

	Features (data)		Target (desired output)
[1,	1, 0, 0,	1, 1]	1

Features (data)						Target (desired output)
[1,	1,	0,	0,	1,	1]	1
[0,	0,	0,	1,	0,	1]	0

Features (data)						Target (desired output)
[1,	1,	0,	0,	1,	1]	1
[0,	0,	0,	1,	0,	1]	0
[1,	1,	1,	0,	1,	0]	1

Features (data)						Target (desired output)
[1,	1,	0,	0,	1,	1]	1
[0,	0,	0,	1,	0,	1]	0
[1,	1,	1,	0,	1,	0]	1
[1,	0,	0,	0,	0,	1]	1

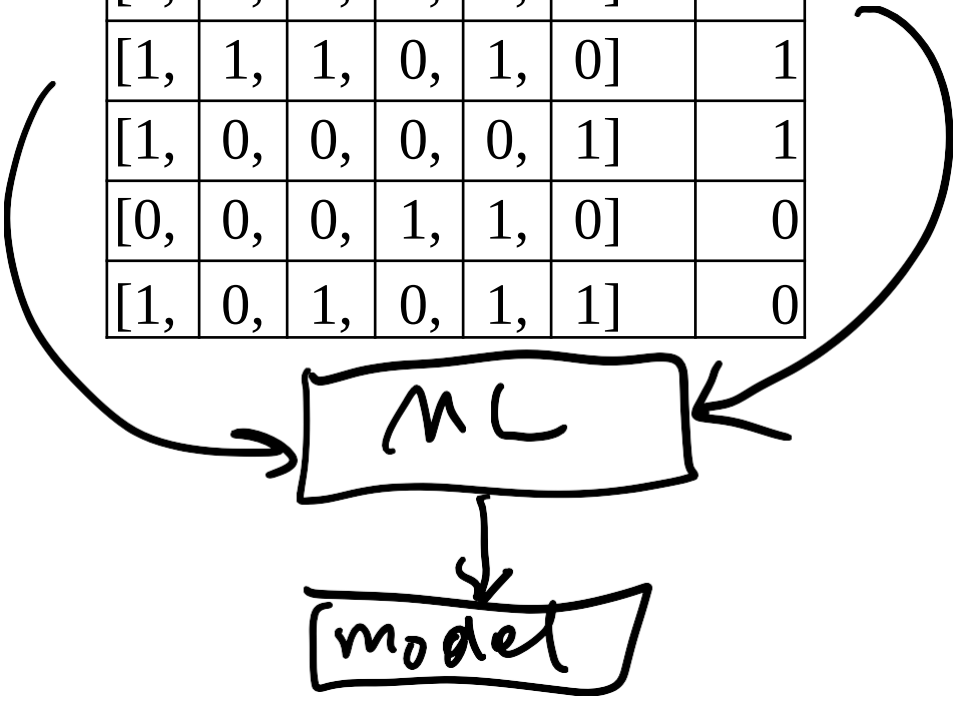
Features (data)						Target (desired output)
[1,	1,	0,	0,	1,	1]	1
[0,	0,	0,	1,	0,	1]	0
[1,	1,	1,	0,	1,	0]	1
[1,	0,	0,	0,	0,	1]	1
[0,	0,	0,	1,	1,	0]	0

Features (data)						Target (desired output)
[1,	1,	0,	0,	1,	1]	1
[0,	0,	0,	1,	0,	1]	0
[1,	1,	1,	0,	1,	0]	1
[1,	0,	0,	0,	0,	1]	1
[0,	0,	0,	1,	1,	0]	0
[1,	0,	1,	0,	1,	1]	0

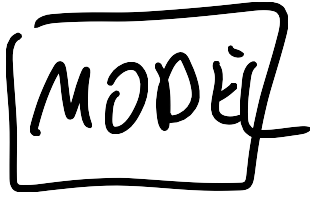
Machine Learning

- Get data
- Define & calculate features
- **Train and use the model**

Features (data)						Target (desired output)
[1,	1,	0,	0,	1,	1]	1
[0,	0,	0,	1,	0,	1]	0
[1,	1,	1,	0,	1,	0]	1
[1,	0,	0,	0,	0,	1]	1
[0,	0,	0,	1,	1,	0]	0
[1,	0,	1,	0,	1,	1]	0

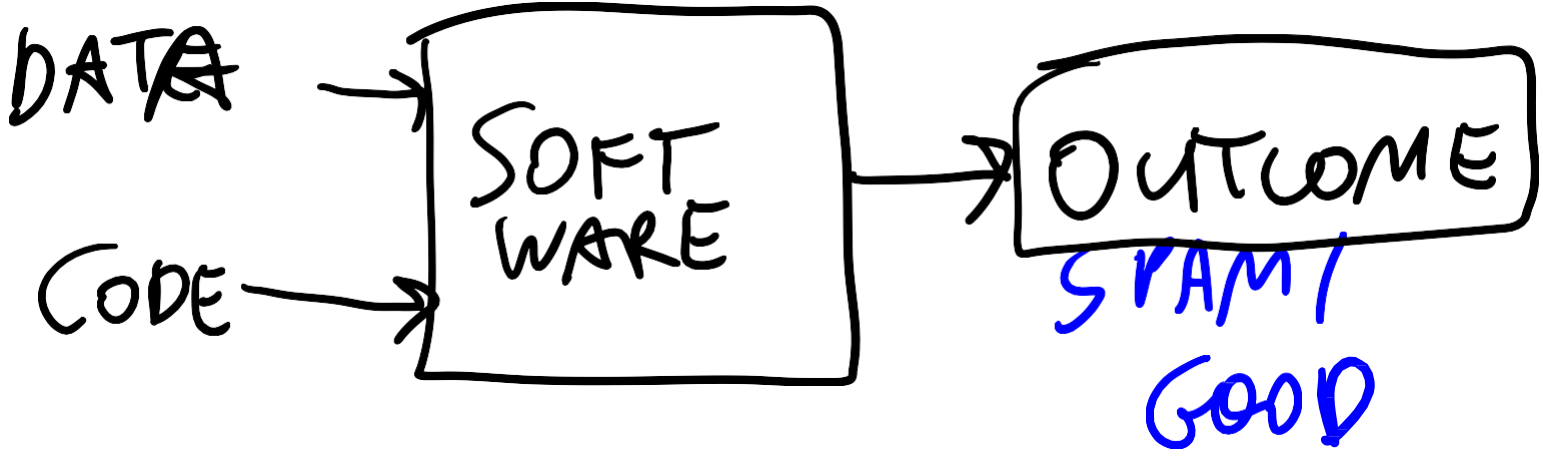


Apply

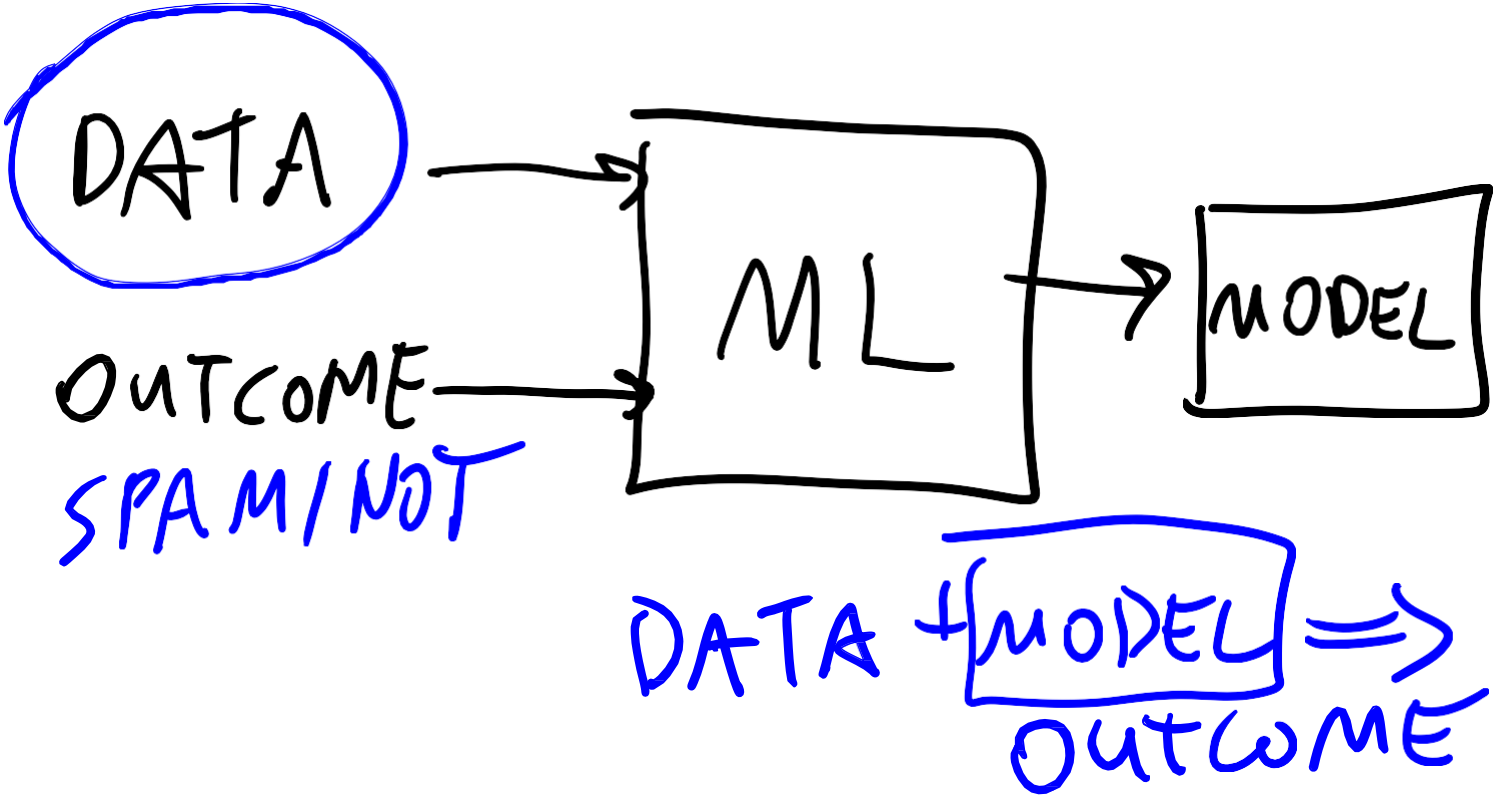


Features (data)						Predictions (output)	Final outcome (decision)
[0,	0,	0,	1,	0,	1]	0.8	SPAM
[0,	0,	0,	1,	1,	0]	0.6	S
[1,	0,	1,	0,	1,	1]	0.1	GOOD
[1,	1,	1,	0,	1,	0]	0.31	G
[1,	0,	0,	0,	0,	1]	0.7	S
[1,	1,	0,	0,	1,	1]	0.4	G

≥ 0.5



data + code => software => outcome



data + outcome => ML => model

Next

Supervised machine learning

- A bit more formal definition
- Examples: regression, classification, ranking

Machine Learning

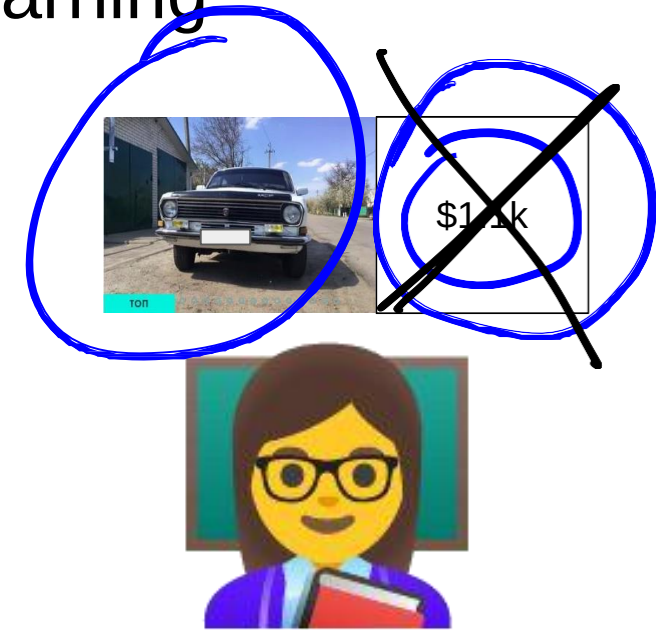
Session #1.3

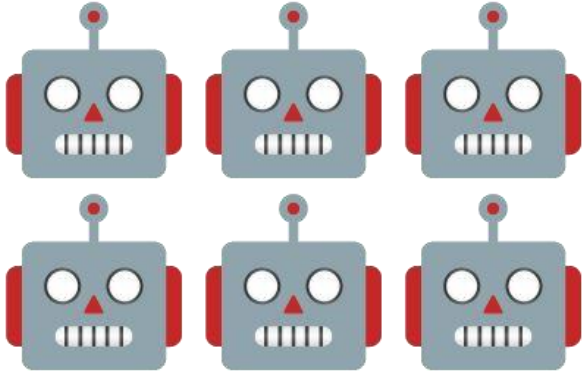
Supervised Machine Learning

Session #1.2: Plan

- Formal definition of supervised machine learning
- Examples: regression, classification, ranking

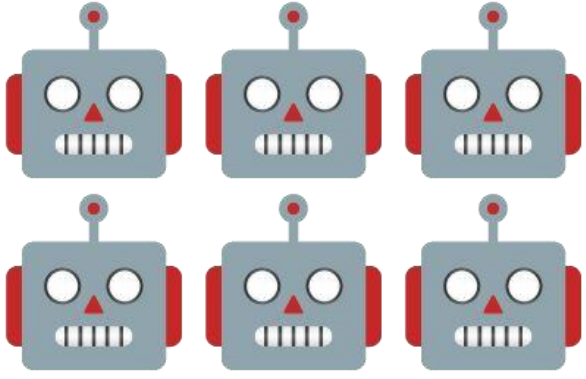
Supervised Machine Learning

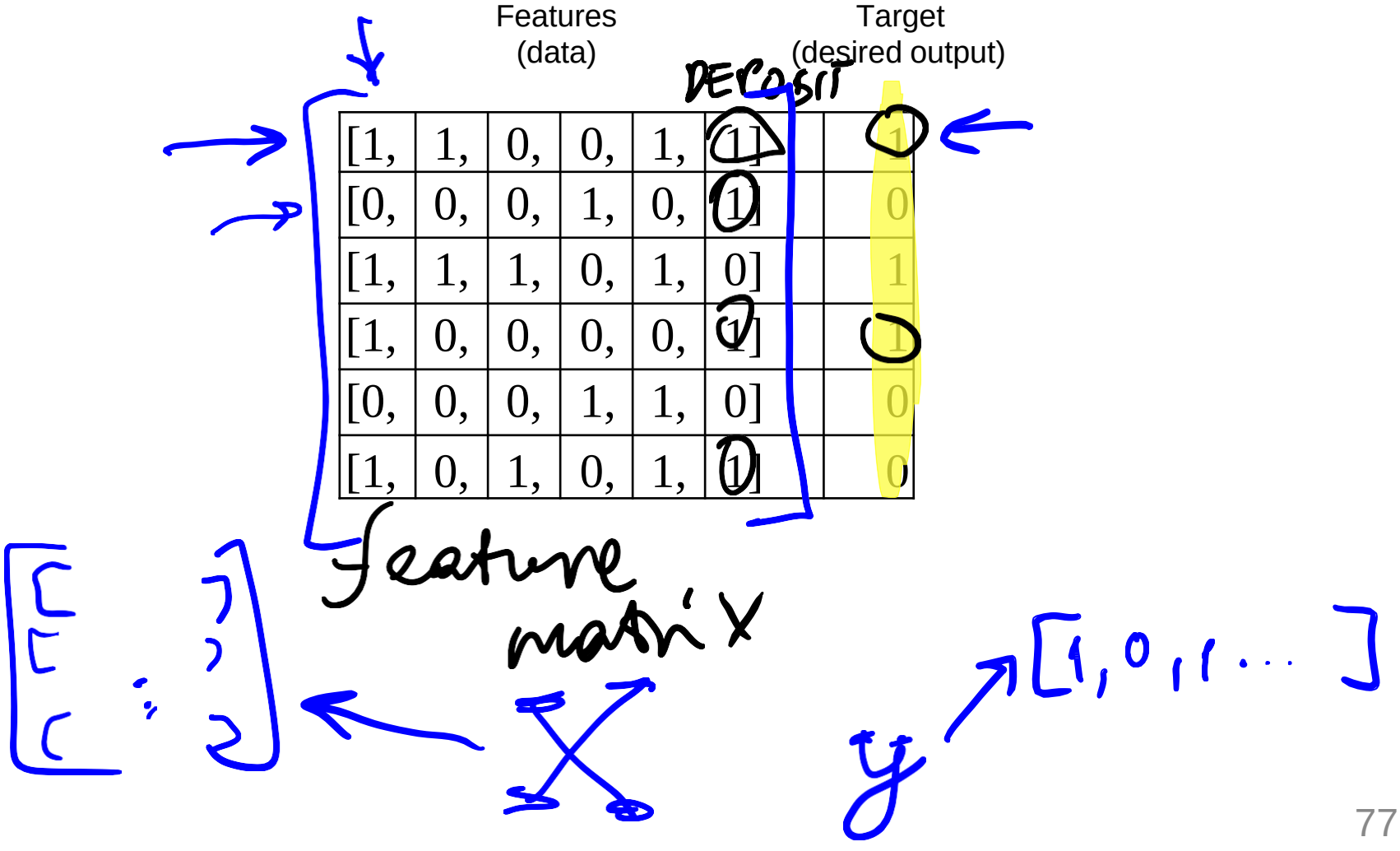


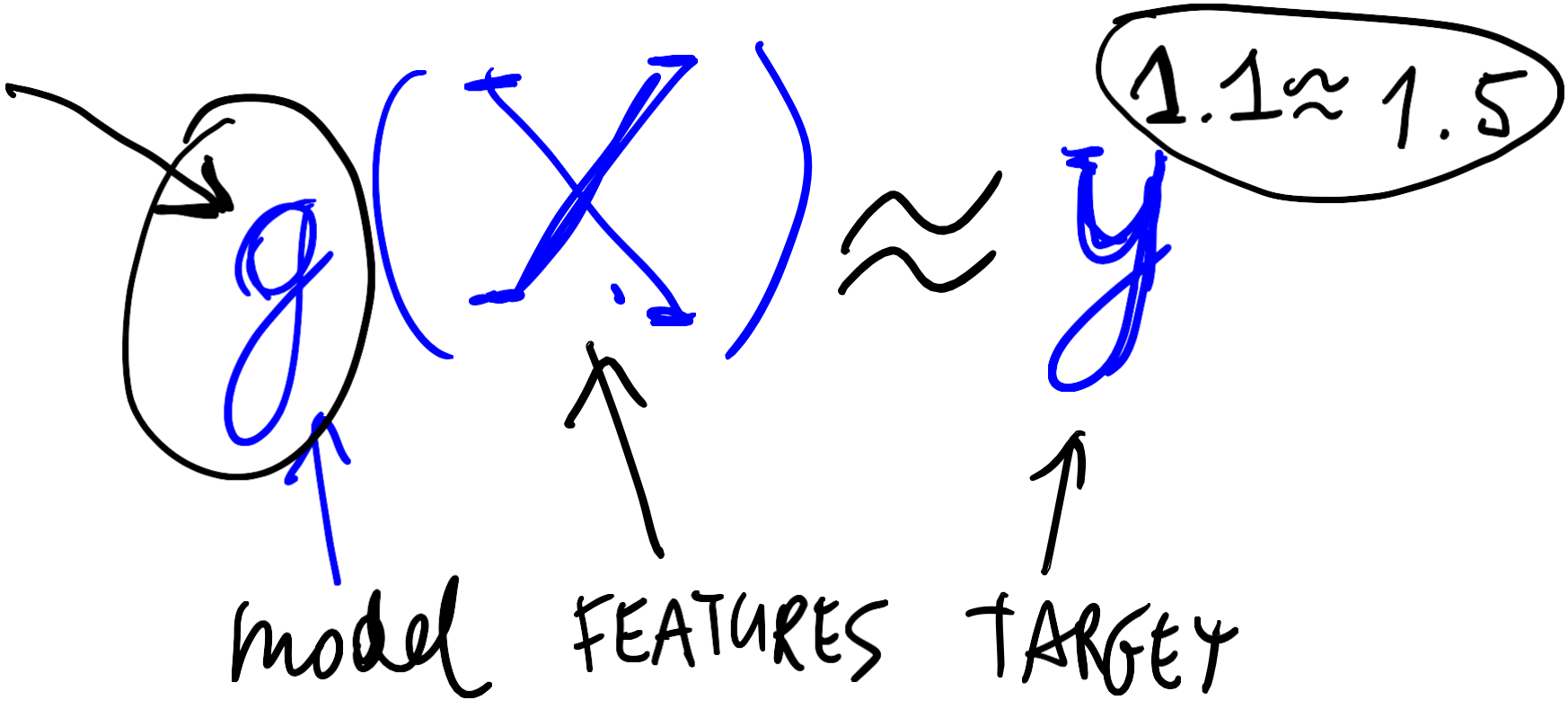


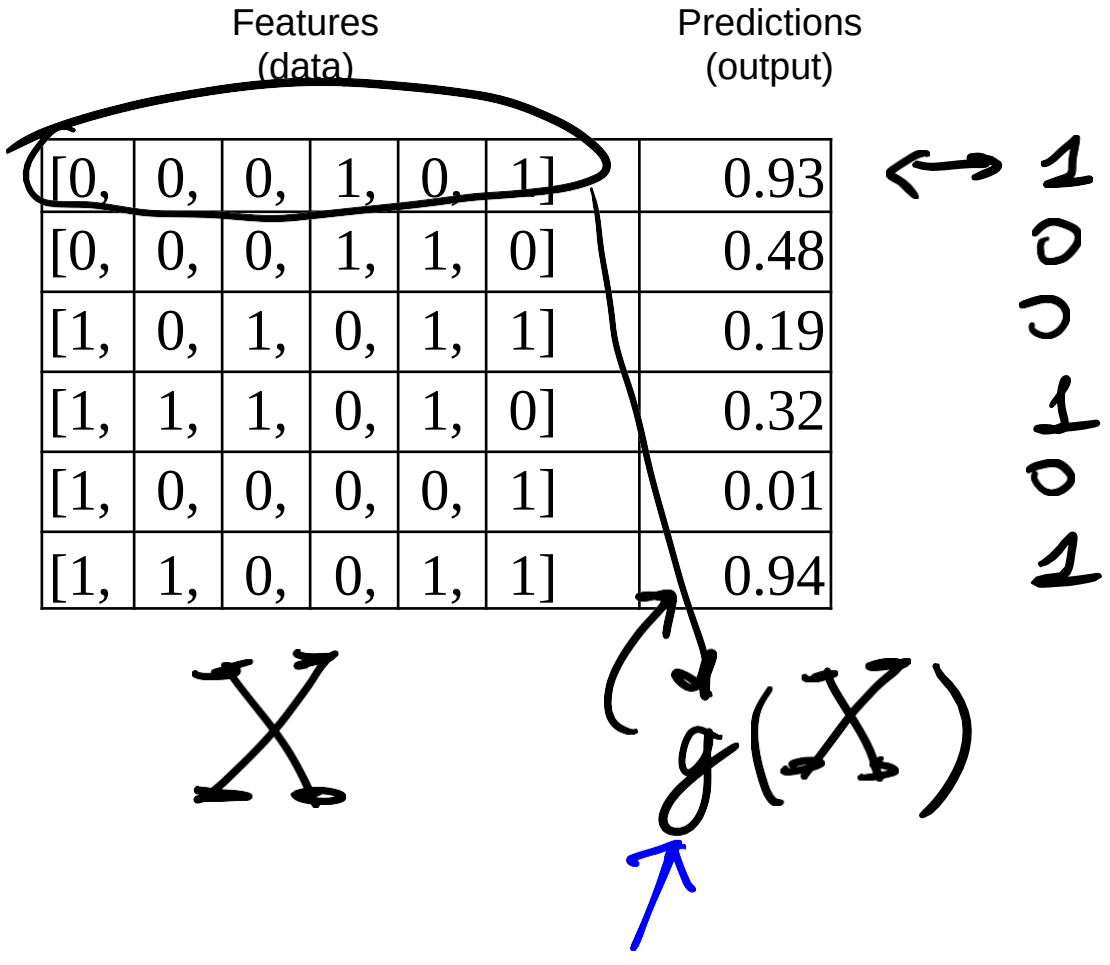
Supervised Machine Learning

Subject: Waiting for your reply **From:** prince1@test.com We are delighted to inform you that you won 1.000.000 (one million) US Dollars. To claim the prize, you need to pay a small processing fee. Please deposit $10 to our PayPal account at prince@test.com. Once we receive the money, we will start the transfer. Congratulations again!	spam




Features
(data)
Target
(desired output)
DEPOSIT
[1, 1, 0, 0, 1, 1]
[0, 0, 0, 1, 0, 1]
[1, 1, 1, 0, 1, 0]
[1, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 1, 0]
[1, 0, 1, 0, 1, 1]
1
0
1
1
0
0
feature matrix
y
[1, 0, 1 ...]


g(X) ≈ y
1.1 ≈ 1.5
model
FEATURES
TARGET


Features
(data)
Predictions
(output)
[0, 0, 0, 1, 0, 1] 0.93
[0, 0, 0, 1, 1, 0] 0.48
[1, 0, 1, 0, 1, 1] 0.19
[1, 1, 1, 0, 1, 0] 0.32
[1, 0, 0, 0, 0, 1] 0.01
[1, 1, 0, 0, 1, 1] 0.94
1
0
1
0
1
X
g(X)

Supervised Machine Learning

Regression:

$50k

$0 - +\infty$

Supervised Machine Learning

Regression:

$50k

Regression:

$1m

Supervised Machine Learning

Classification:

car

Supervised Machine Learning

Classification:

car

Classification:

spam

Classification

Multiclass:

cat
dog 10
car 1000

Classification

Multiclass:

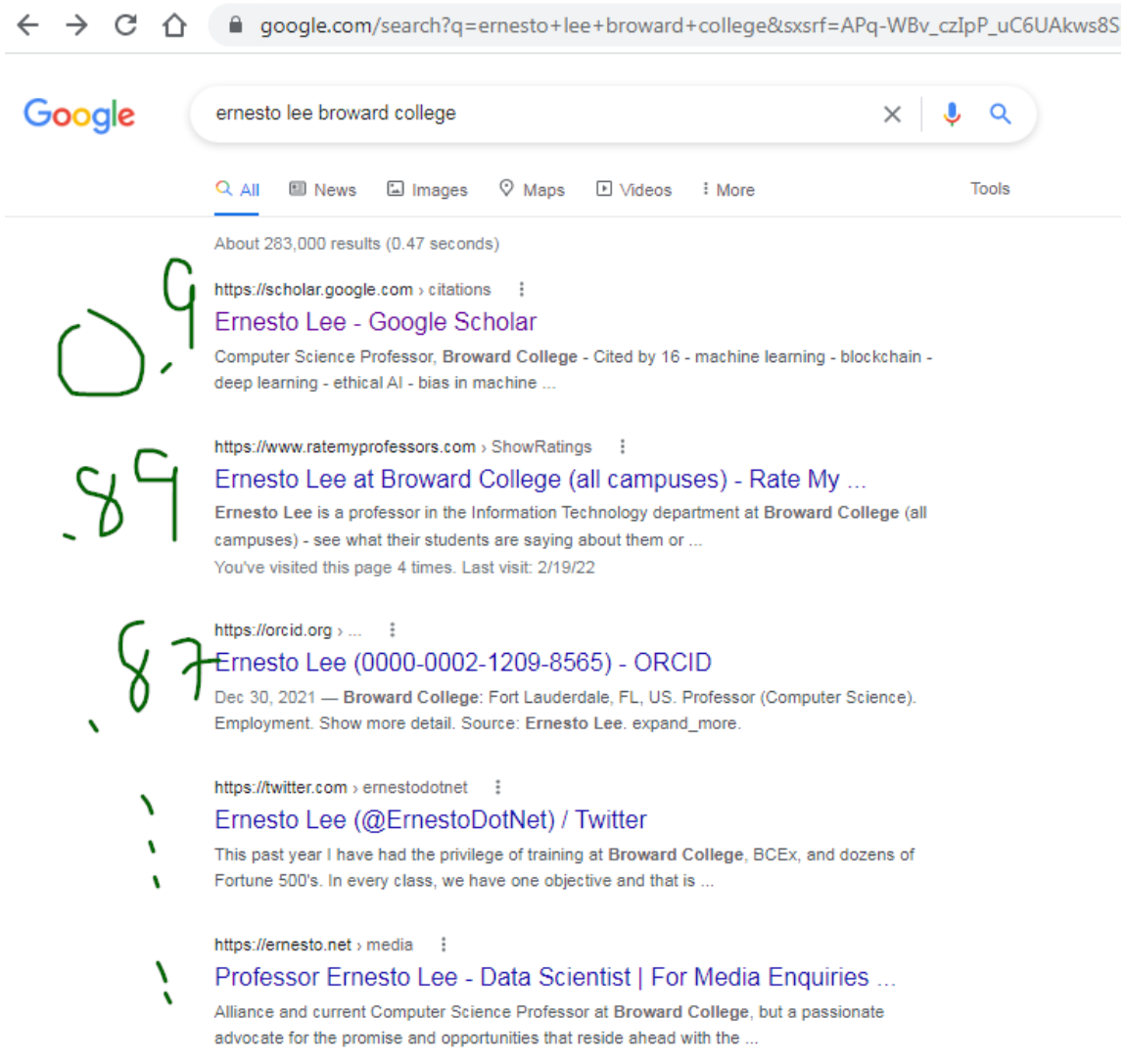
cat
dog
car

Binary:

spam 1
not spam 0

g 0-1

Ranking


google.com/search?q=ernesto+lee+broward+college&sxsrf=APq-WBv_czIpP_uC6UAkws8S
Google
ernesto lee broward college
All
News
Images
Maps
Videos
More
Tools
About 283,000 results (0.47 seconds)
https://scholar.google.com › citations
Ernesto Lee - Google Scholar
Computer Science Professor, Broward College - Cited by 16 - machine learning - blockchain - deep learning - ethical AI - bias in machine ...
https://www.ratemyprofessors.com › ShowRatings
Ernesto Lee at Broward College (all campuses) - Rate My ...
Ernesto Lee is a professor in the Information Technology department at Broward College (all campuses) - see what their students are saying about them or ...
You've visited this page 4 times. Last visit: 2/19/22
https://orcid.org › ...
Ernesto Lee (0000-0002-1209-8565) - ORCID
Dec 30, 2021 — Broward College: Fort Lauderdale, FL, US. Professor (Computer Science). Employment. Show more detail. Source: Ernesto Lee. expand_more.
https://twitter.com › ernestodotnet
Ernesto Lee (@ErnestoDotNet) / Twitter
This past year I have had the privilege of training at Broward College, BCEx, and dozens of Fortune 500's. In every class, we have one objective and that is ...
https://ernesto.net › media
Professor Ernesto Lee - Data Scientist | For Media Enquiries ...
Alliance and current Computer Science Professor at Broward College, but a passionate advocate for the promise and opportunities that reside ahead with the ...

iphone

Cała Polska

+ 0 km

Szukaj

Znaleziono 35 404 ogłoszenia

2 sztuki x Kabel Lightning AUKEY 2 m iPhone

Pszczew - dzisiaj 12:14

45 zł

Iphone 6s 16gb polecam

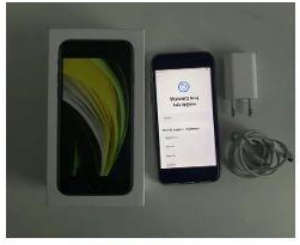
Gdańsk, Młyniska - dzisiaj 12:14

399 zł

iPhone 6 srebrny 64 GB w bardzo dobrym stanie

Wólka Radzymińska - dzisiaj 12:14

470 zł



iPhone SE 2020 64GB ładny, zadbany

Poznań, Junikowo - dzisiaj 12:14

1 000 zł

Okazja iPhone 8 64 GB

Olsztyn - dzisiaj 12:14

Etui , Pokrowiec Case iPhone 6 + Plus / 6S Plus Henri Bendel New York

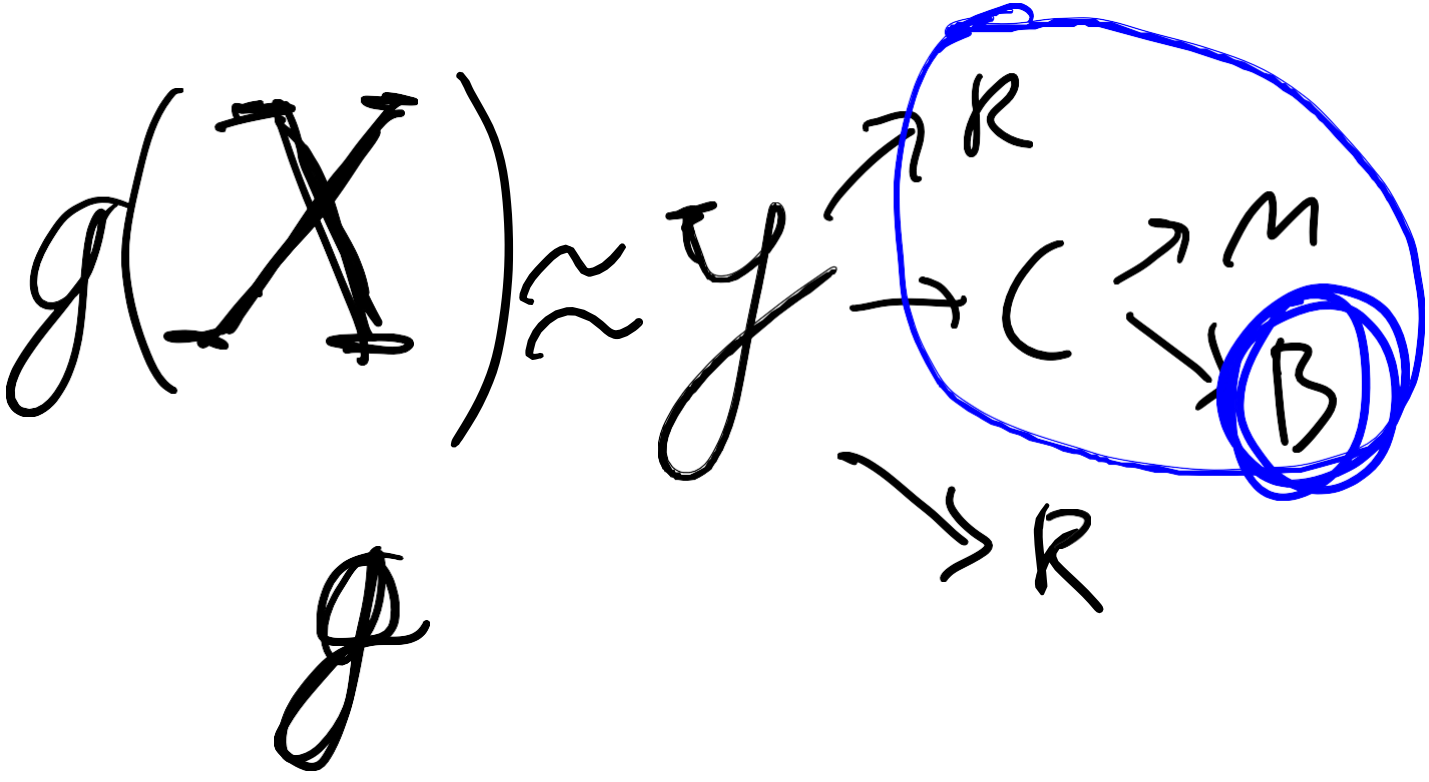
Przeworsk - dzisiaj 12:14

iPhone 8 64gb silver

Wejherowo - dzisiaj 12:13

SKUP TELEFONÓW Telewizorów Laptopów Konsoli SAMSUNG Iphon...

Rzeszów - dzisiaj 12:13


g(X) ≃ Y
R
C
M
B
R
g

Next

Process: CRISP-DM

Machine Learning

Session #1.4

CRISP-DM ML Process

Session #1.4: Plan

- CRISP-DM — methodology for organizing ML projects

Session #1.4: Plan

- CRISP-DM — methodology for organizing ML projects
- From problem understanding to deployment

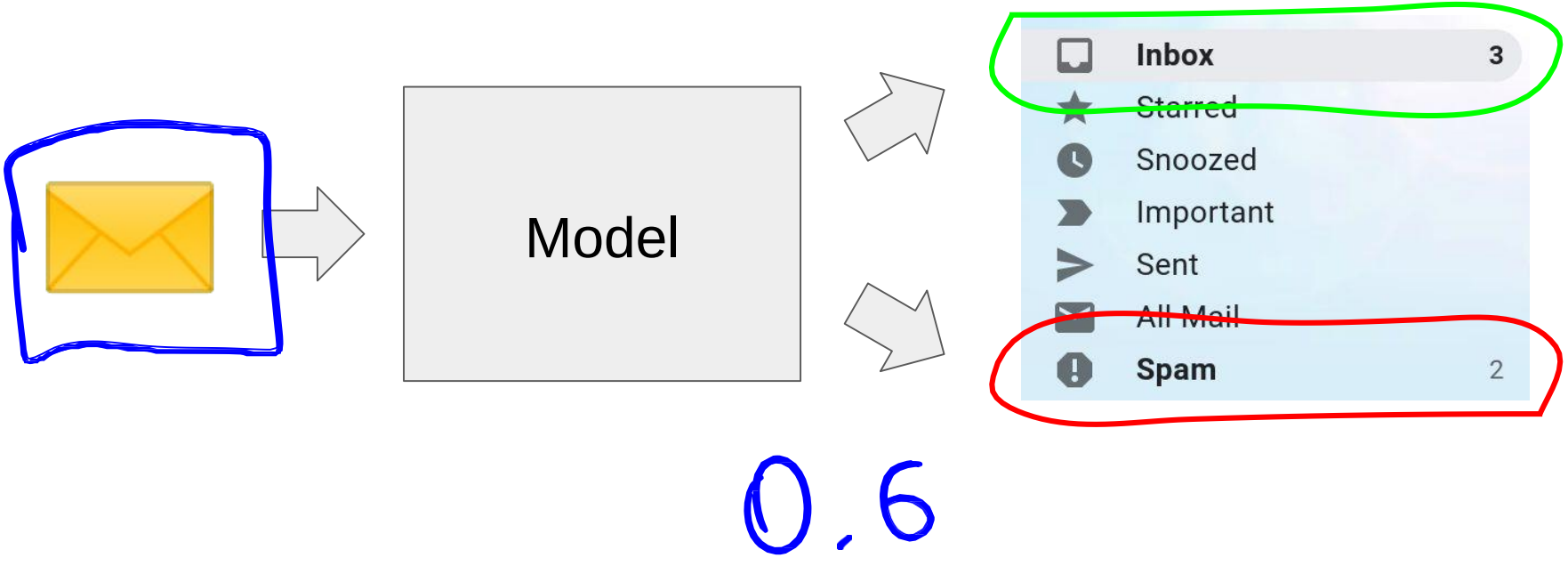
Session #1.4: Plan

- CRISP-DM
- From problem understanding to deployment
- Spam detection example

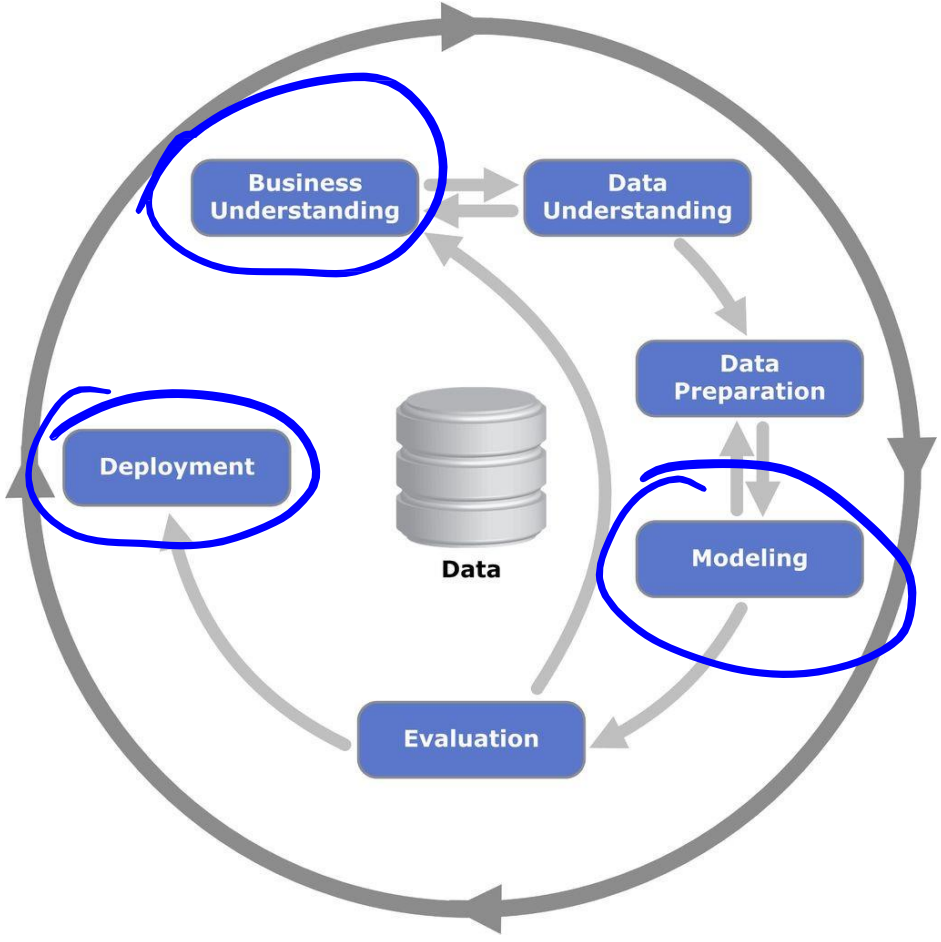
ML Projects

- Understand the problem
- Collect the data
- Train the model
- Use it

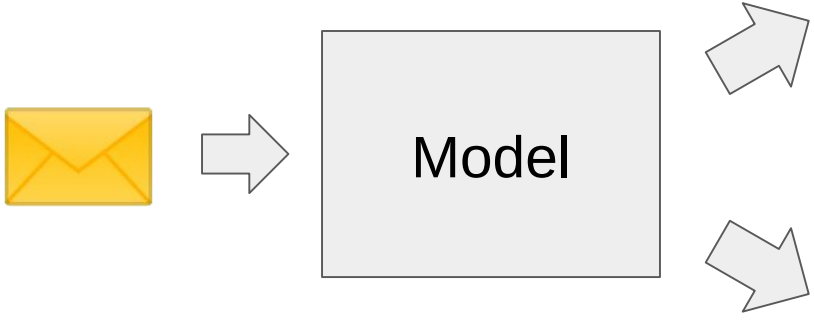
Spam detection

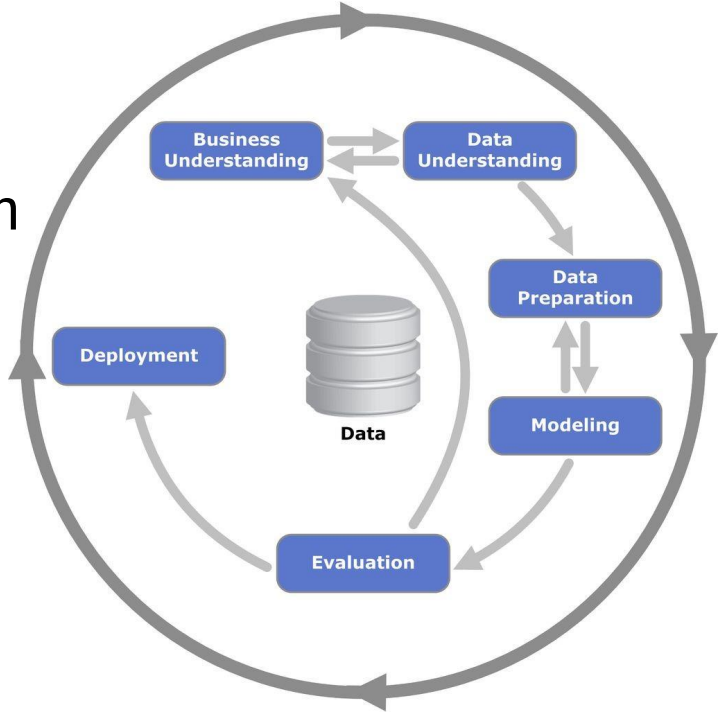


CRISP-DM

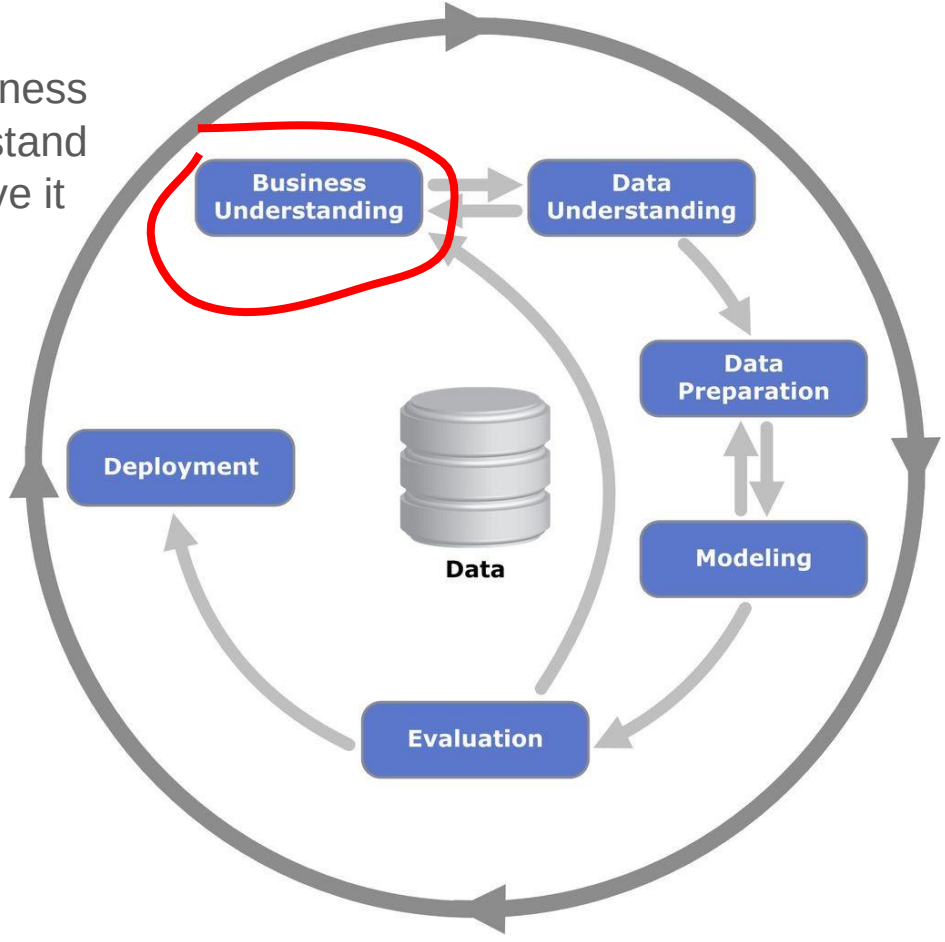


Picture: CRISP-DM


Model
spam
not spam

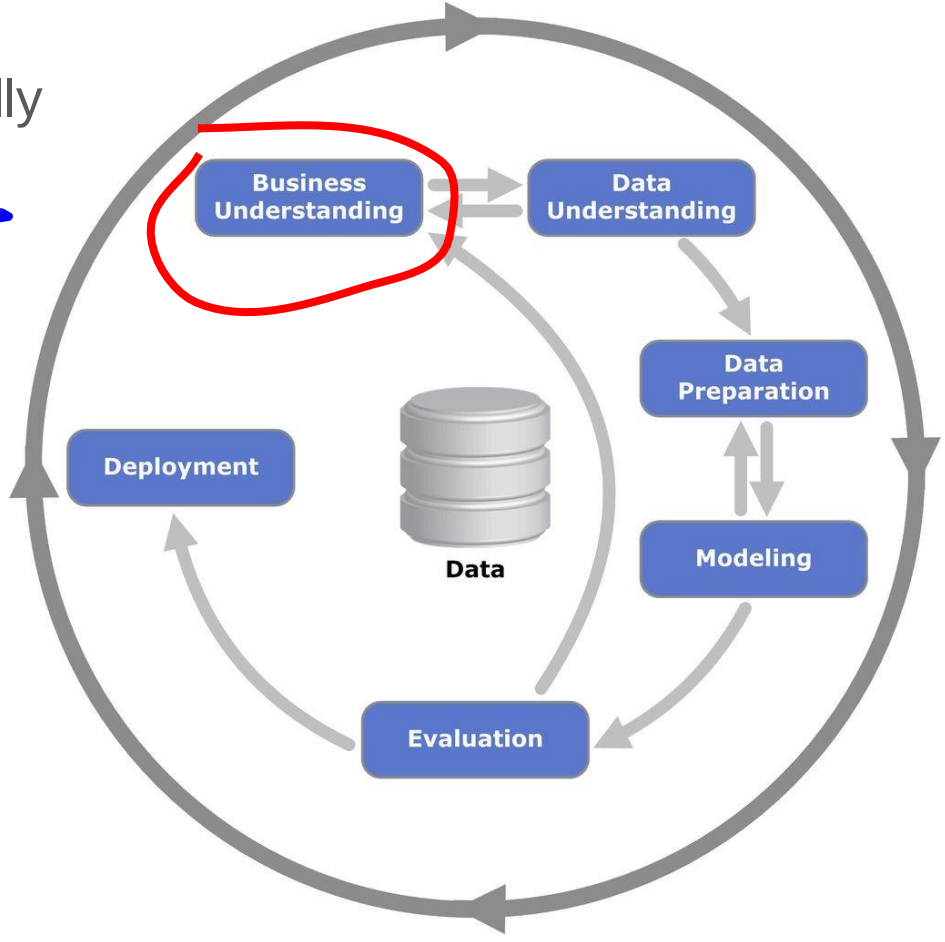

Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment
Data

Identify the business problem, understand how we can solve it



Picture: CRISP-DM

Do we actually need ML here?



Picture: CRISP-DM

Business understanding

- Our users complain about spam
- Analyze to what extent it’s a problem
- Will Machine Learning help?
- If not: propose an alternative solution

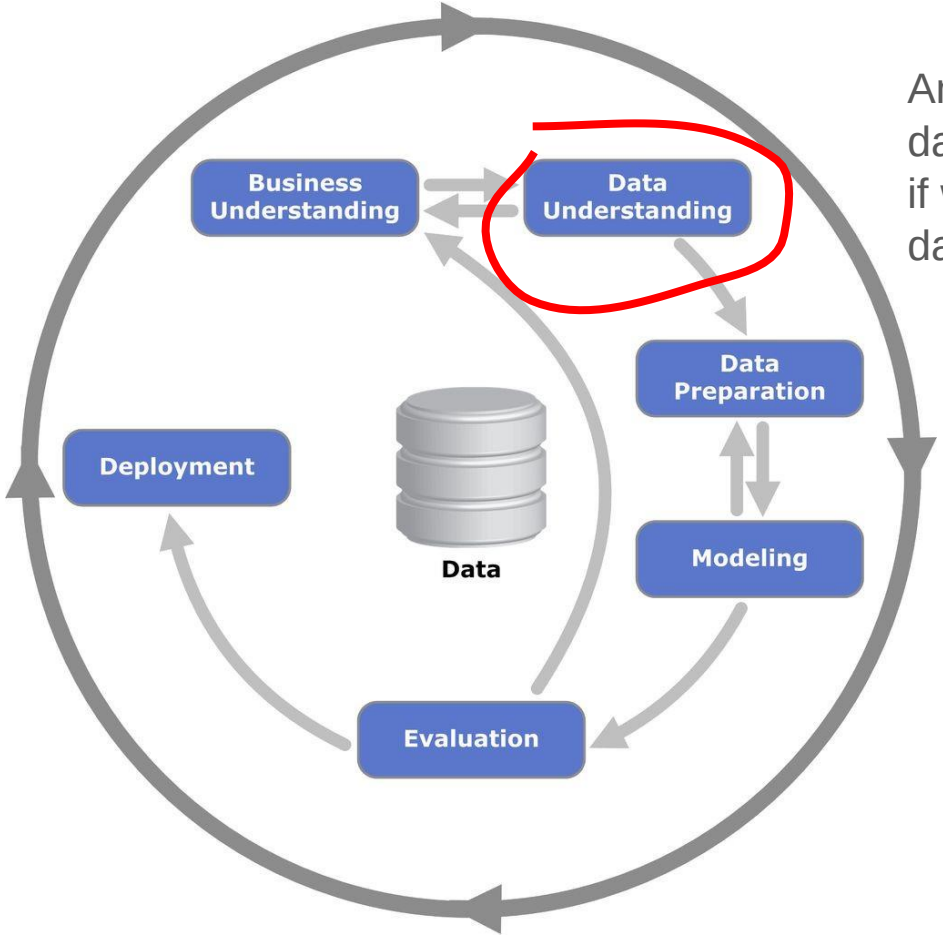
Business understanding

Define the goal:

- Reduce the amount of spam messages, or
- Reduce the amount of complaints about spam

The goal has to be measurable

- Reduce the amount of spam by 50%



Analyze available data sources, decide if we need to get more data

Picture: CRISP-DM

Data understanding

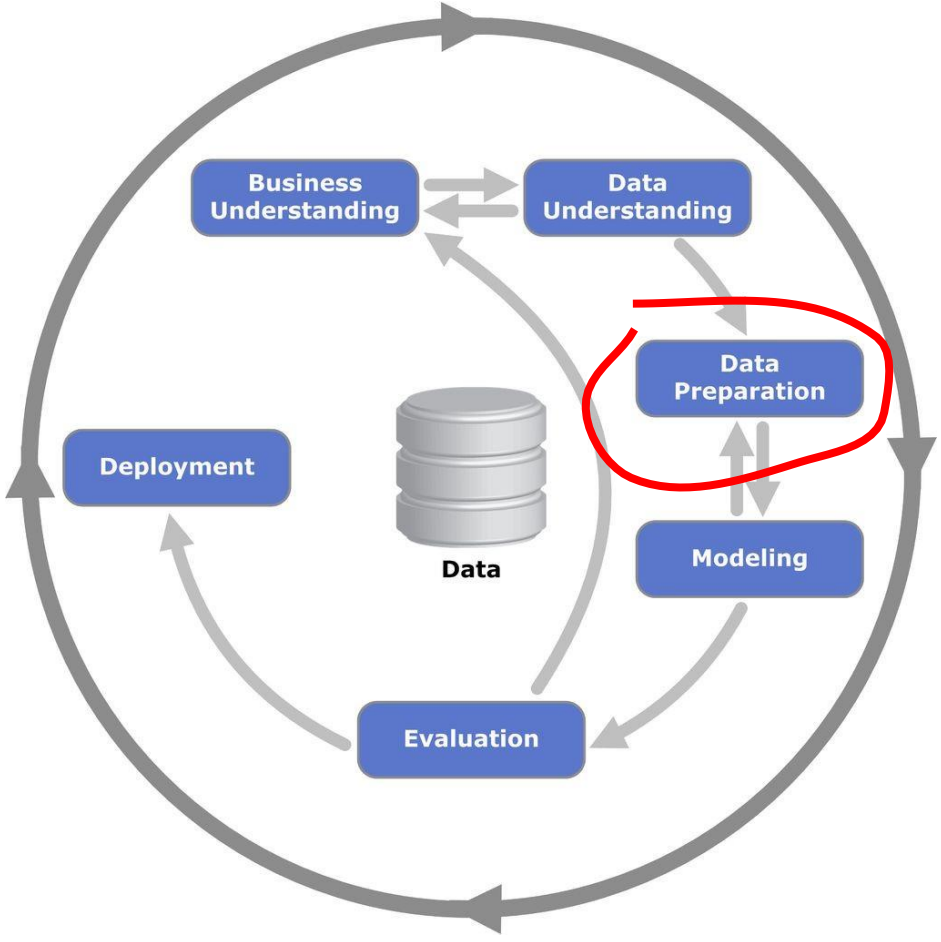
Identify the data sources

- We have a report spam button
- Is the data behind this button good enough?
- Is it reliable?
- Do we track it correctly?
- Is the dataset large enough?
- Do we need to get more data?

Data understanding

Identify the data sources

- It may influence the goal
- We may go back to the previous step and adjust it

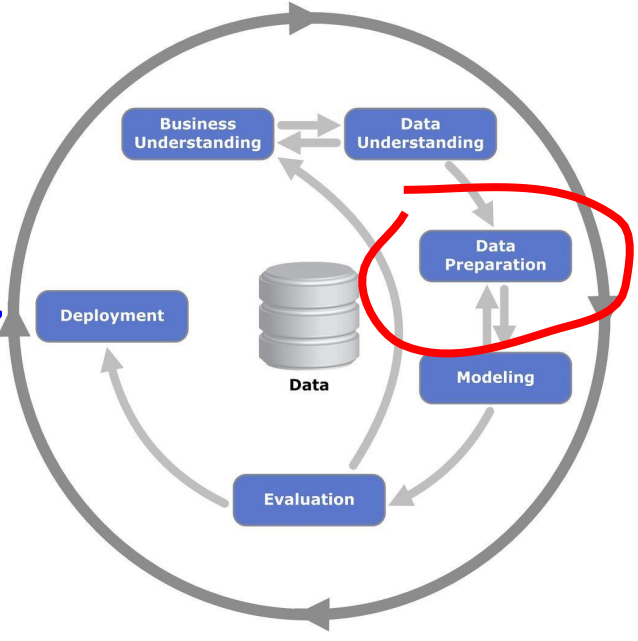


Transform the data so it can be put into a ML algorithm

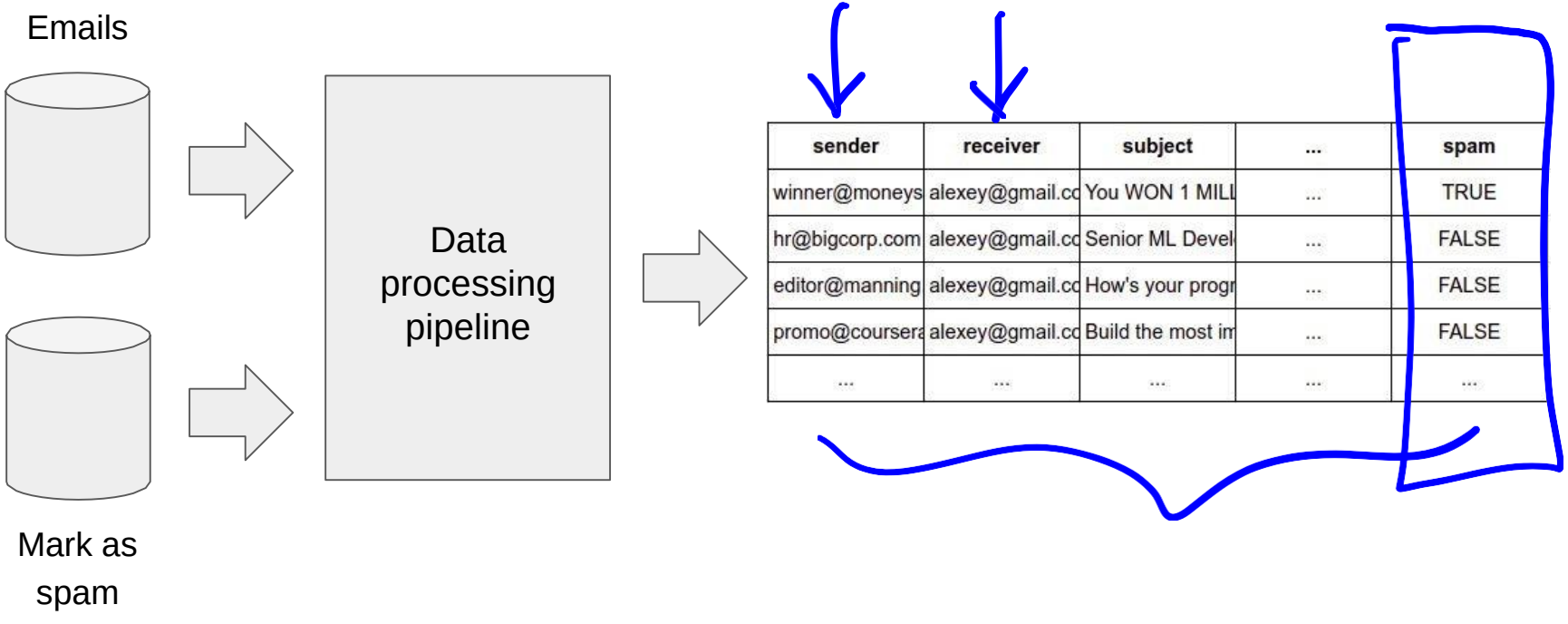
Picture: CRISP-DM

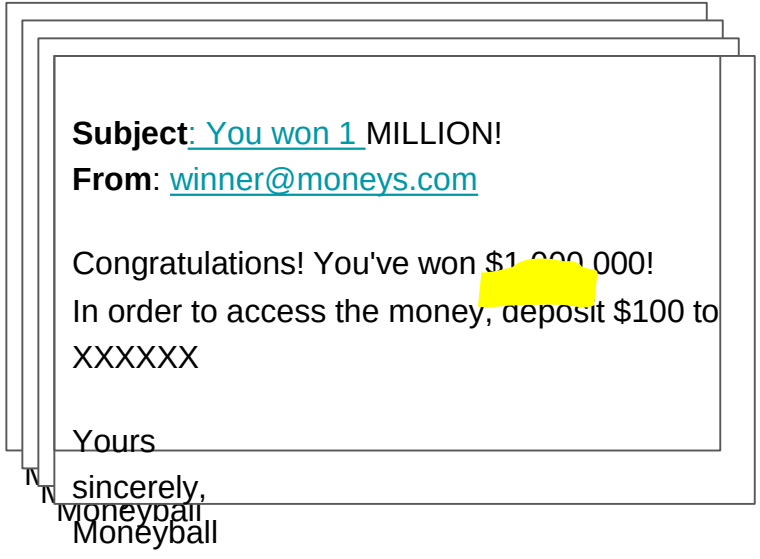
Data preparation

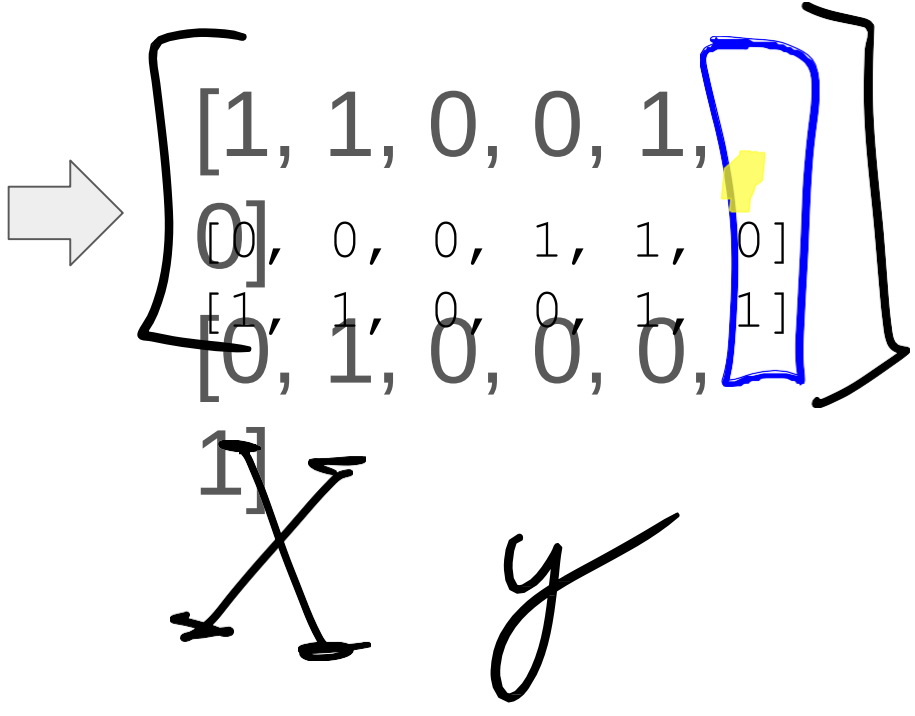
- Clean the data
- Build the pipelines
- Convert into tabular form

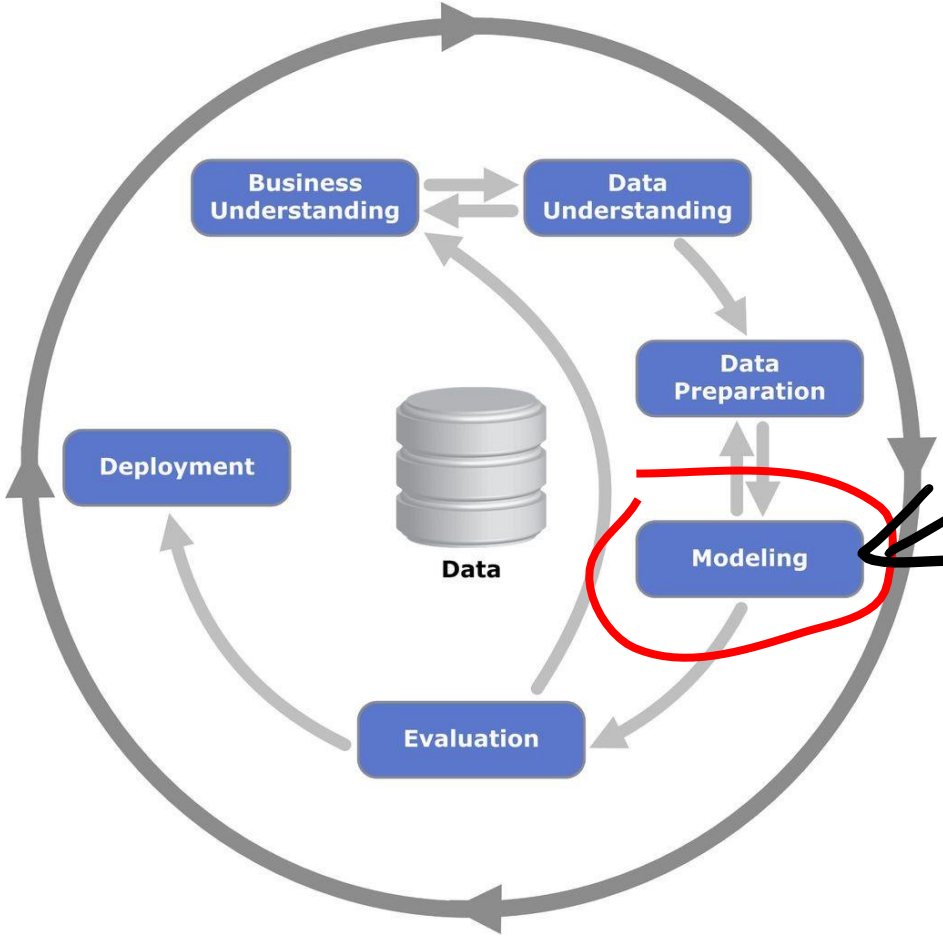


Data preparation




Subject: You won 1 MILLION!
From: winner@moneys.com
Congratulations! You've won $1,000,000!
In order to access the money, deposit $100 to XXXXXX
Yours sincerely,
Moneyball


[1, 1, 0, 0, 1, 0]
[0, 0, 0, 1, 1, 0]
[1, 1, 0, 0, 1, 1]
[0, 1, 0, 0, 0, 1]
X
y



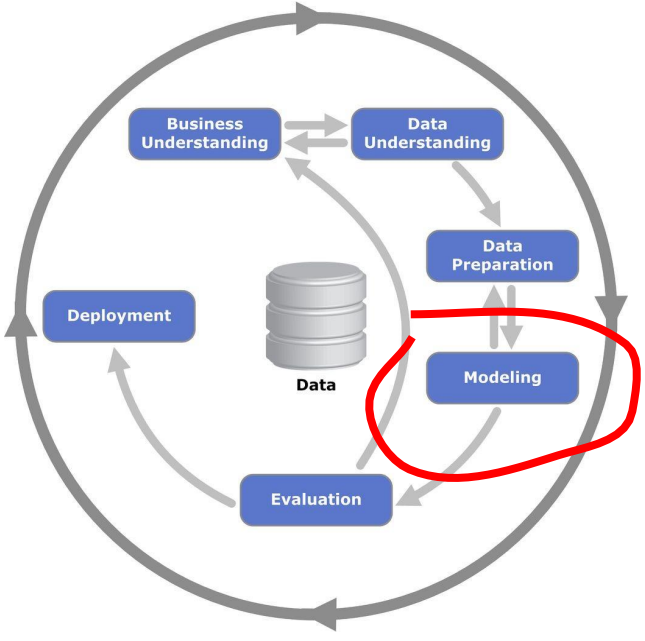
Training the models: the actual Machine Learning happens here

Picture: CRISP-DM

Modeling

Training a model:

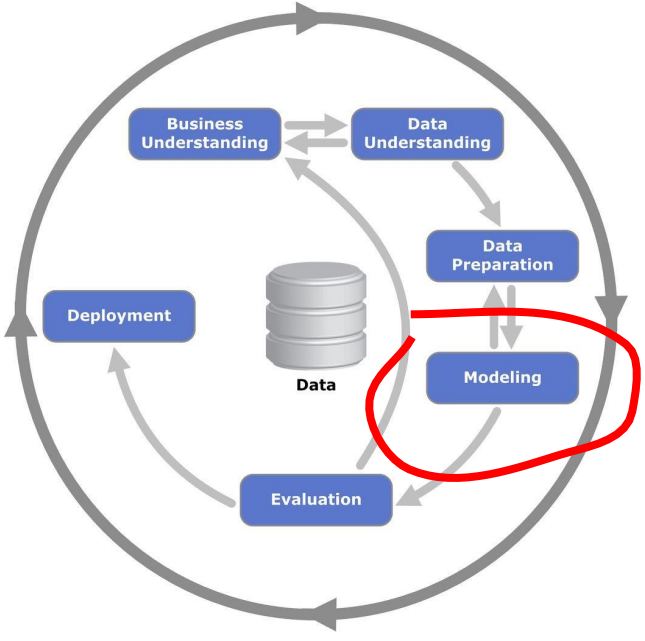
- Try different models
- Select the best one



Modeling

Which model to choose?

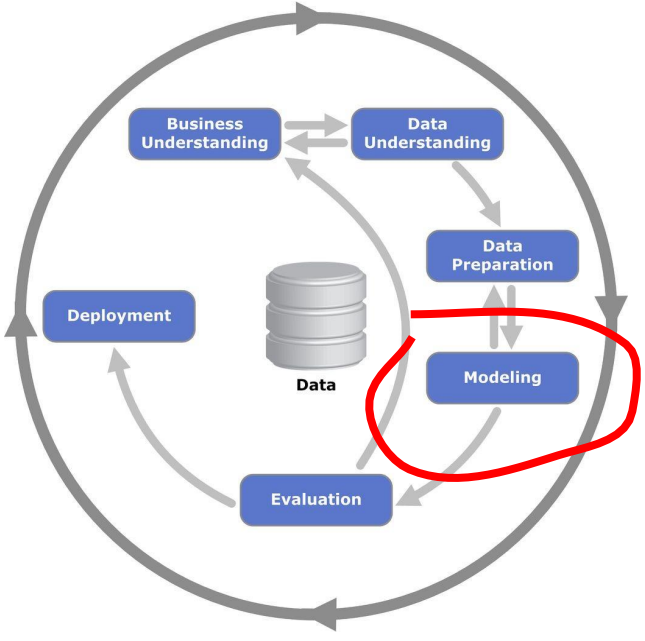
- Logistic regression
- Decision tree
- Neural network
- Or many others

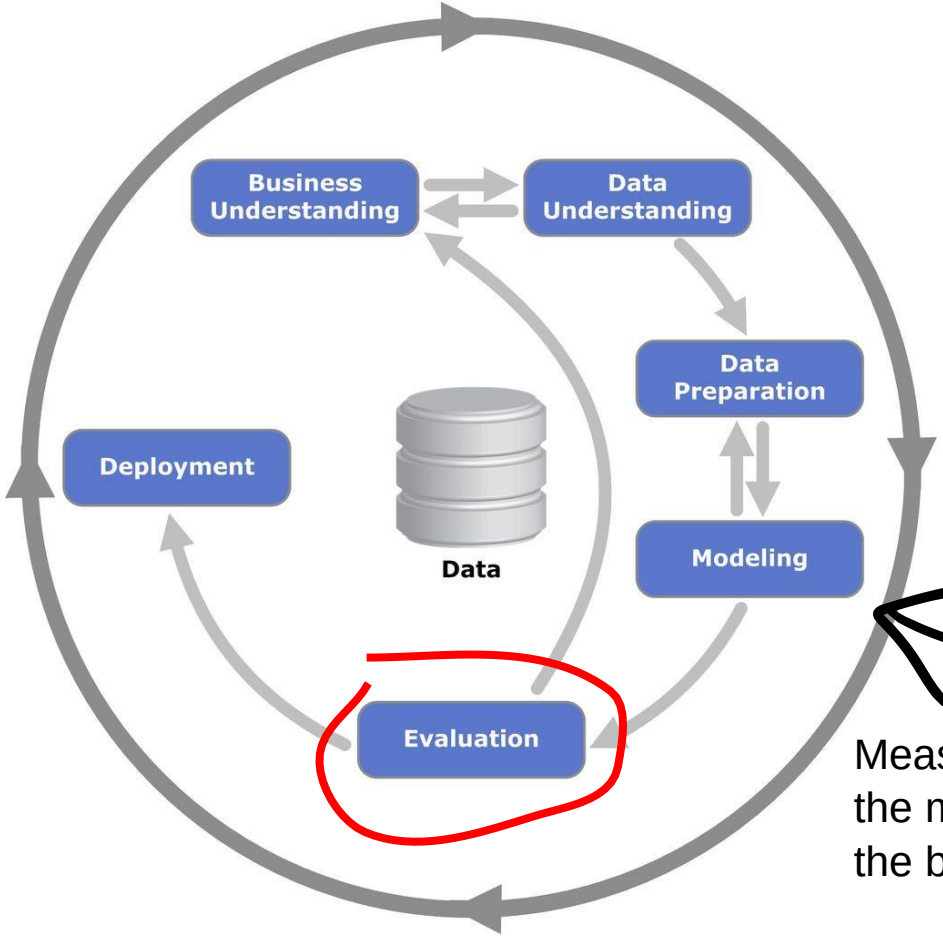


Modeling

Sometimes, we may go back to data preparation:

- Add new features
- Fix data issues





Measure how well
the model solves
the business problem

Picture: CRISP-DM

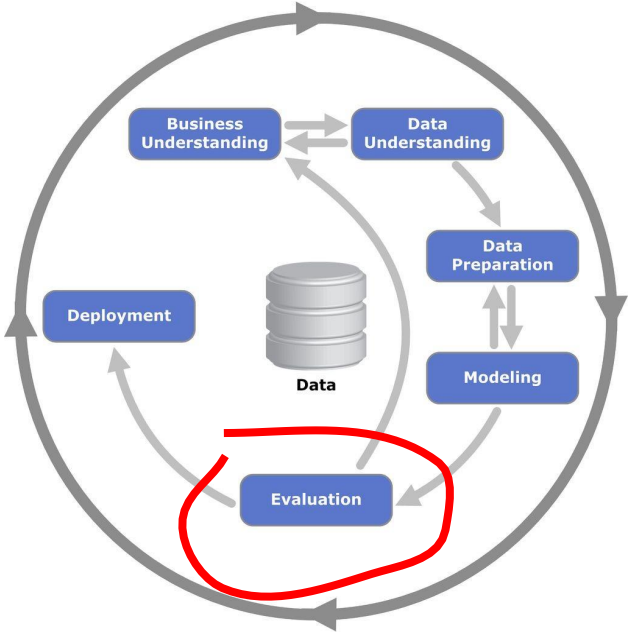
Evaluation

Is the model good enough?

- Have we reached the goal?
- Do our metrics improve?

Goal: Reduce the amount of spam by ~~50%~~ 30

- Have we reduced it? By how much?
- (Evaluate on the test group)



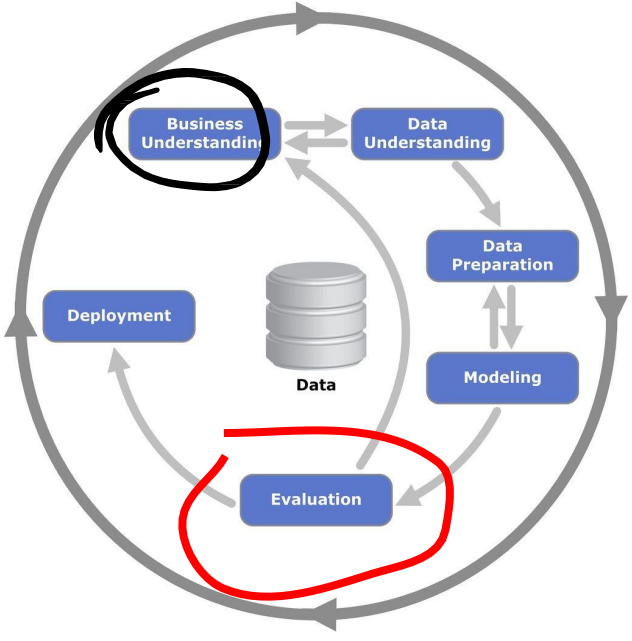
Evaluation

Do a retrospective:

- Was the goal achievable?
- Did we solve/measure the right thing?

After that, we may decide to:

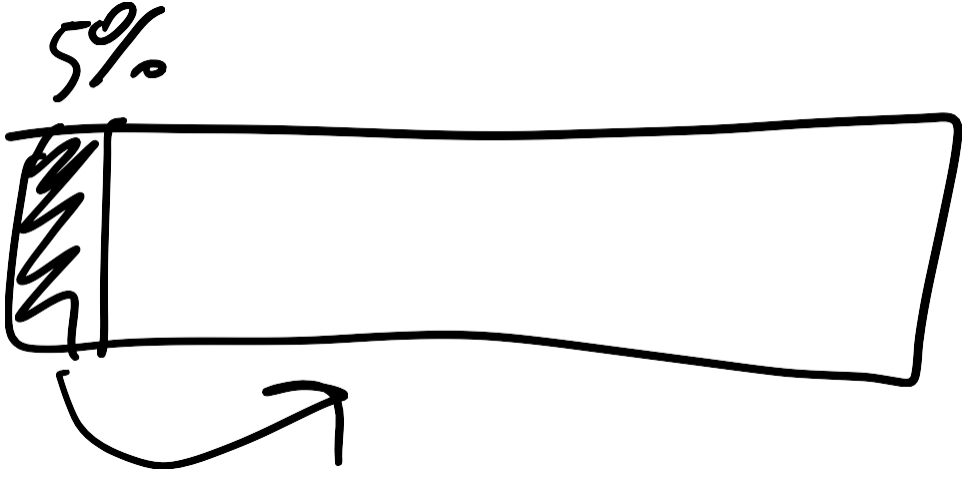
- Go back and adjust the goal
- Roll the model to more users/all users
- Stop working on the project

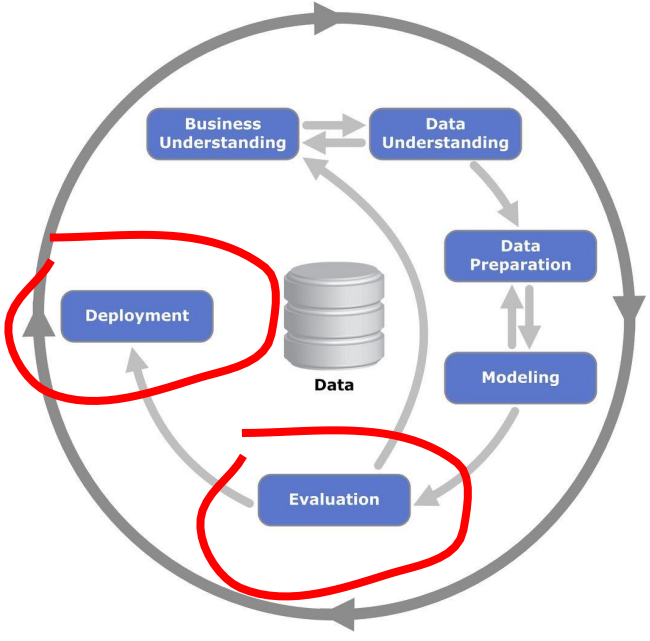


Evaluation + Deployment

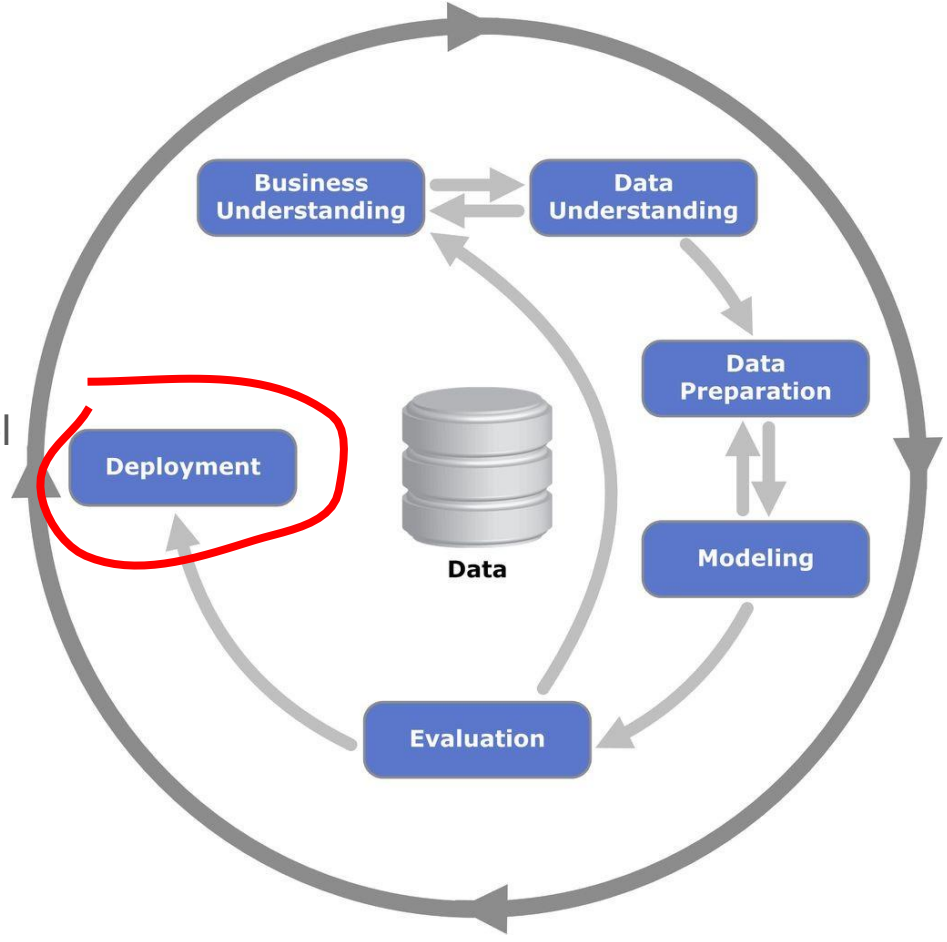
Often happens together:

- Online evaluation: evaluation of live users
- It means: deploy the model, evaluate it





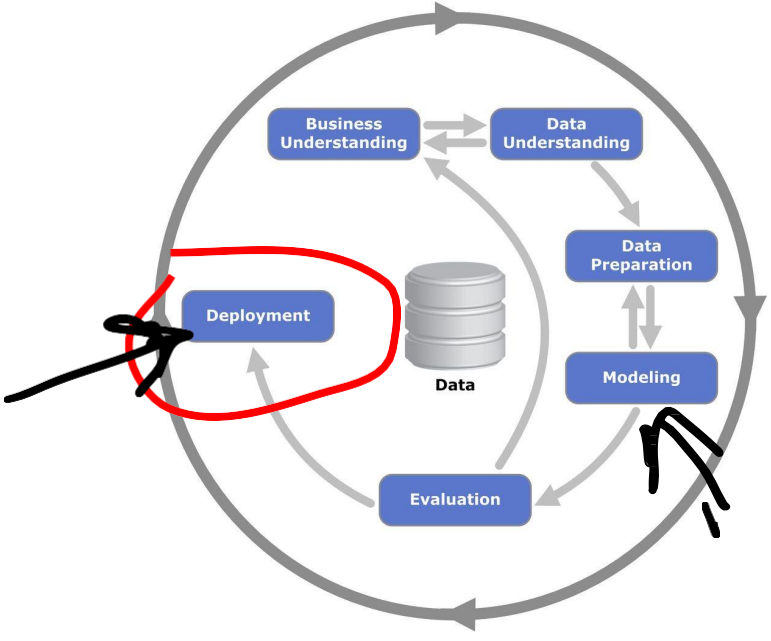
Deploy the model to production



Picture: CRISP-DM

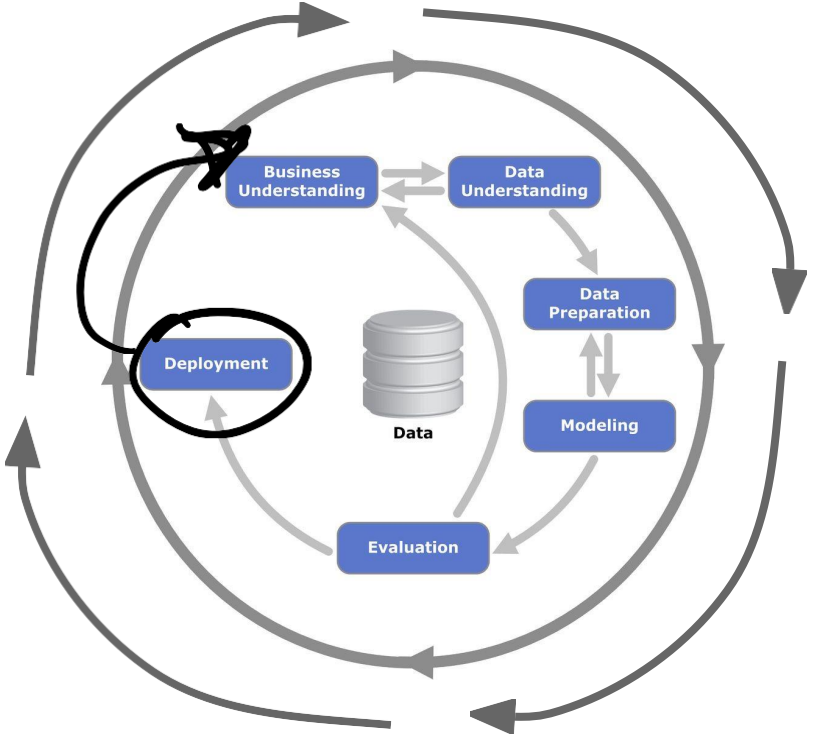
Deployment

- Roll the model to all users
- Proper monitoring
- Ensuring the quality and maintainability



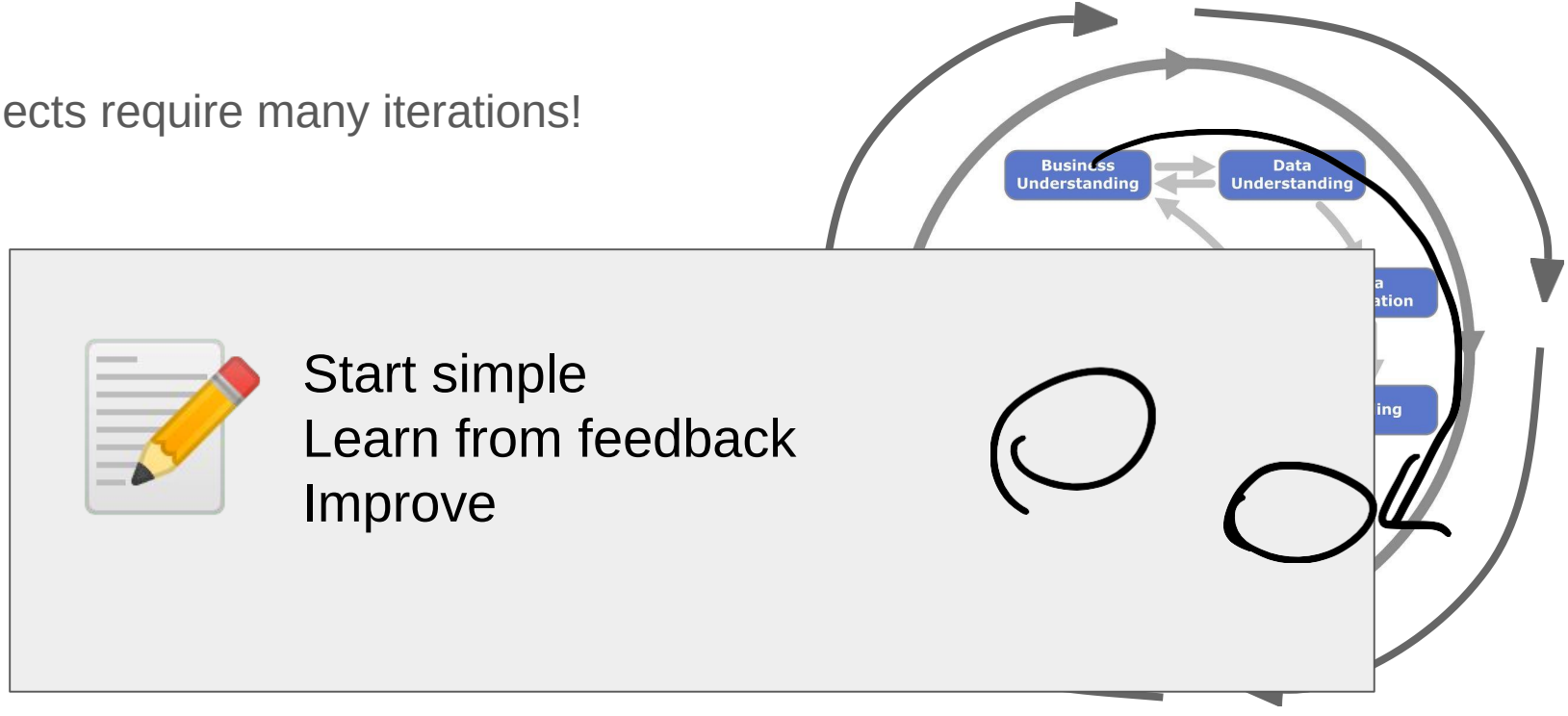
Iterate!

ML projects require many iterations!



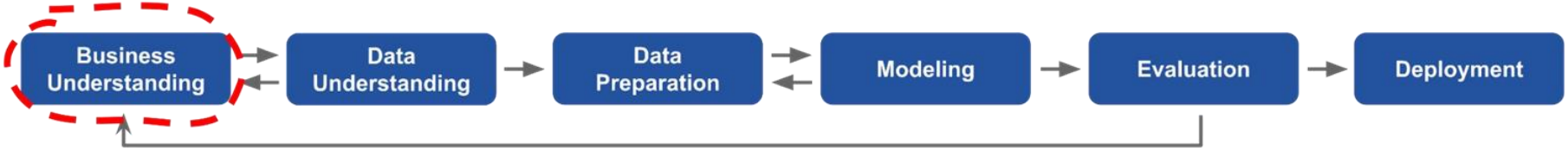
Iterate!

ML projects require many iterations!



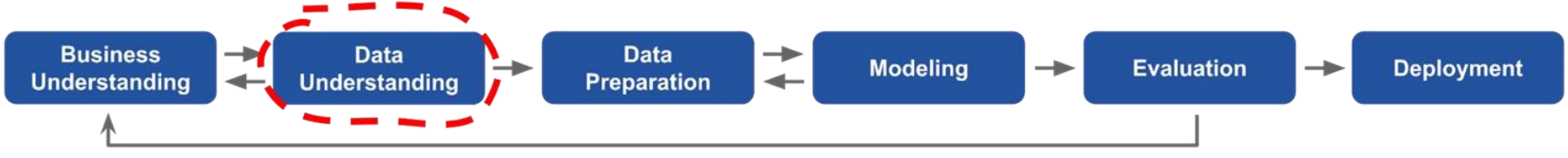
Summary

- Business understanding: define a measurable goal. Ask: do we need ML?



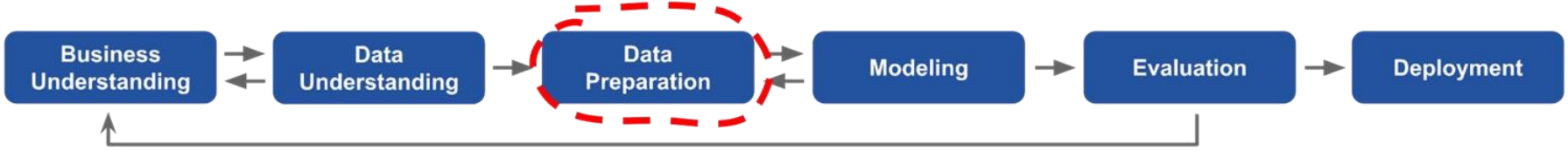
Summary

- Business understanding: define a measurable goal. Ask: do we need ML?
- Data understanding: do we have the data? Is it good?



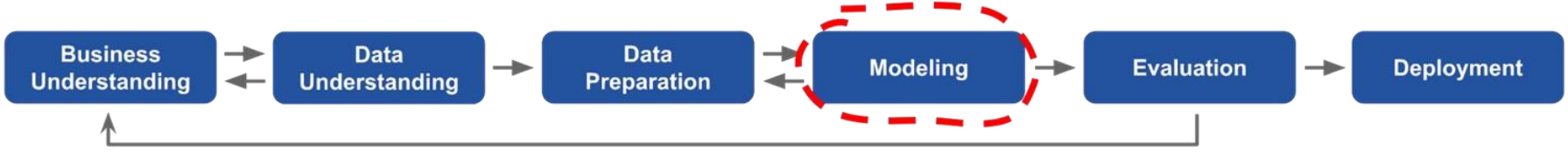
Summary

- Business understanding: define a measurable goal. Ask: do we need ML?
- Data understanding: do we have the data? Is it good?
- Data preparation: transform data into a table, so we can put it into ML



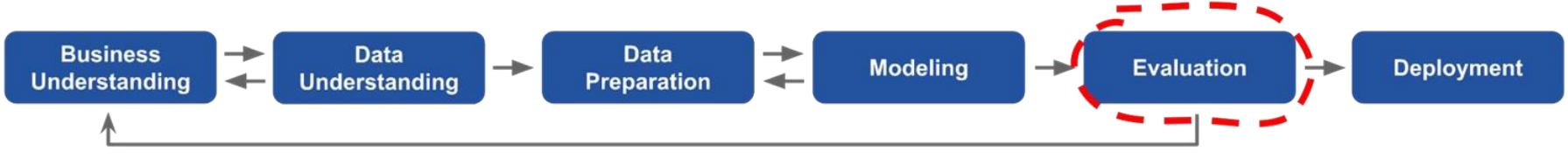
Summary

- Business understanding: define a measurable goal. Ask: do we need ML?
- Data understanding: do we have the data? Is it good?
- Data preparation: transform data into a table, so we can put it into ML
- Modelling: to select the best model, use the validation set



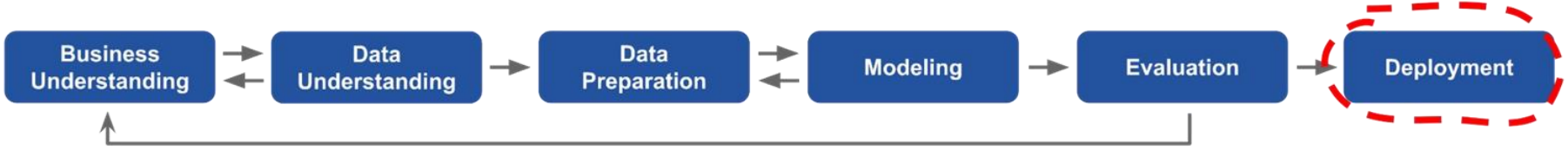
Summary

- Business understanding: define a measurable goal. Ask: do we need ML?
- Data understanding: do we have the data? Is it good?
- Data preparation: transform data into a table, so we can put it into ML
- Modelling: to select the best model, use the validation set
- Evaluation: validate that the goal is reached



Summary

- Business understanding: define a measurable goal. Ask: do we need ML?
- Data understanding: do we have the data? Is it good?
- Data preparation: transform data into a table, so we can put it into ML
- Modelling: to select the best model, use the validation set
- Evaluation: validate that the goal is reached
- Deployment: roll out to production to all the users



Summary

- Business understanding: define a measurable goal. Ask: do we need ML?
- Data understanding: do we have the data? Is it good?
- Data preparation: transform data into a table, so we can put it into ML
- Modelling: to select the best model, use the validation set
- Evaluation: validate that the goal is reached
- Deployment: roll out to production to all the users
- Iterate: start simple, learn from the feedback, improve

Next

The modelling step of CRISP-DM

Machine Learning

Session #1.5

Modelling step: Model Selection

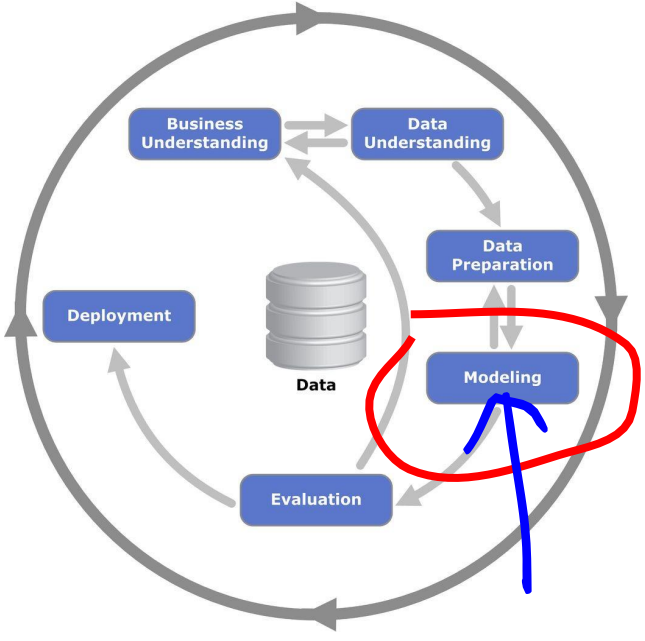
Session #1.5: Plan

- Process for selecting the best model
- Evaluating the model
- Multiple comparison problem

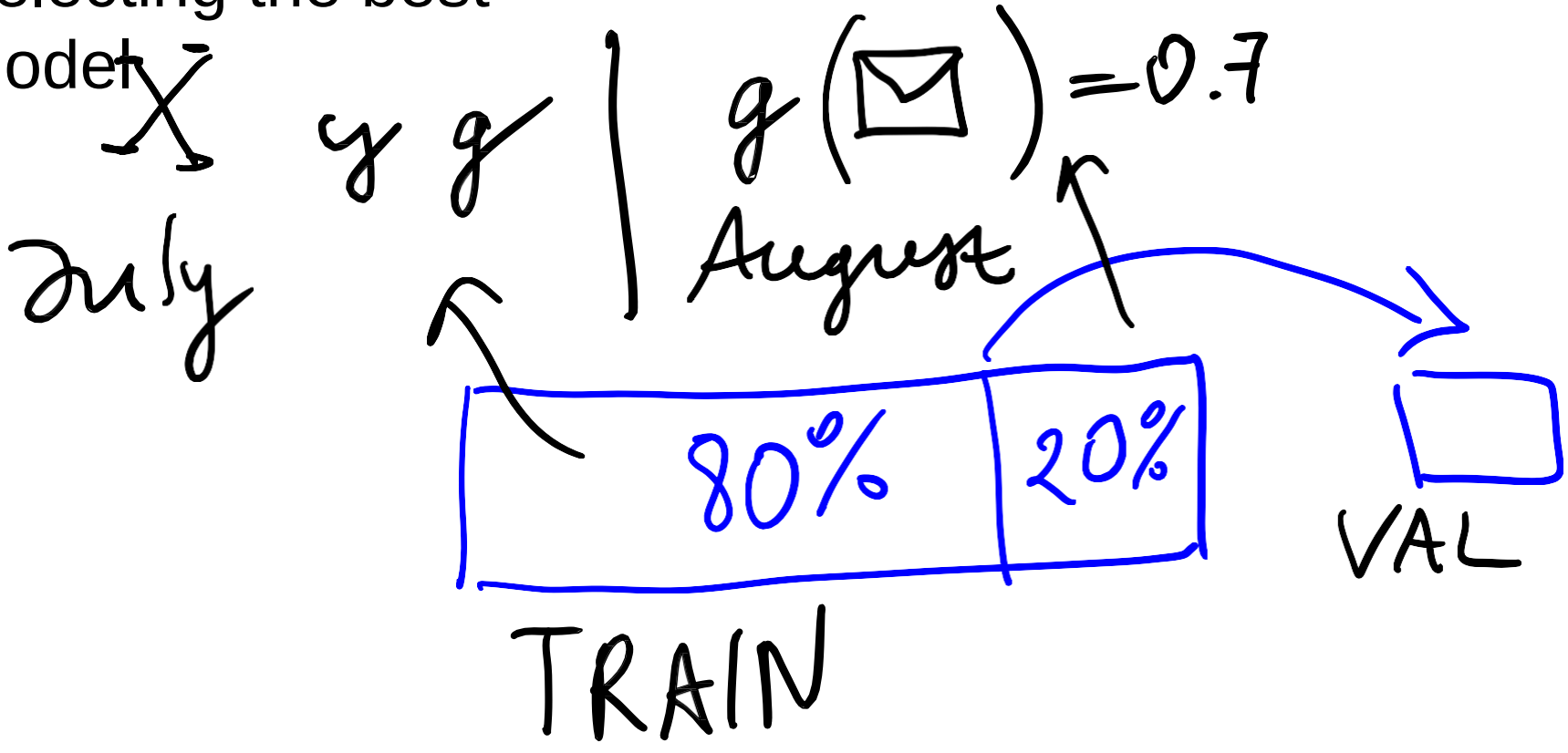
Modeling

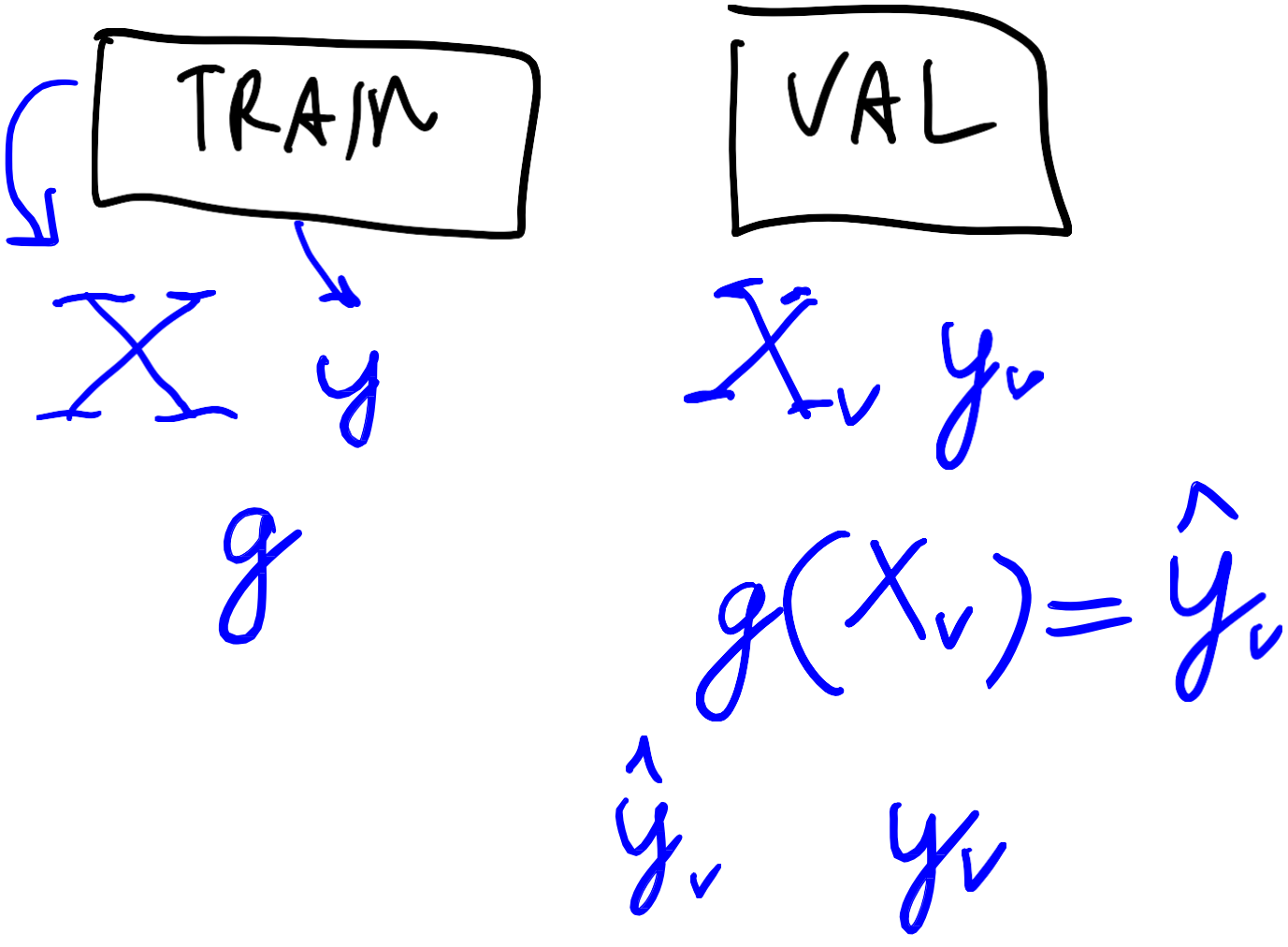
Which model to choose?

- Logistic regression
- Decision tree
- Neural network
- Or many others



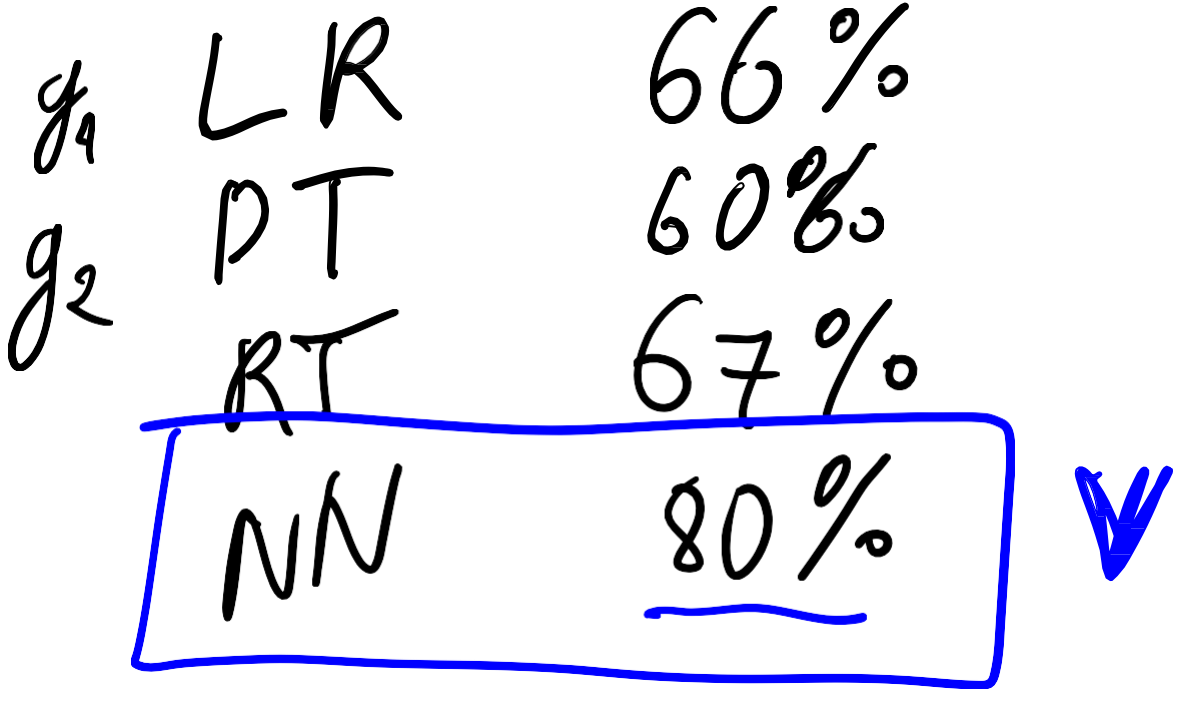
Selecting the best model



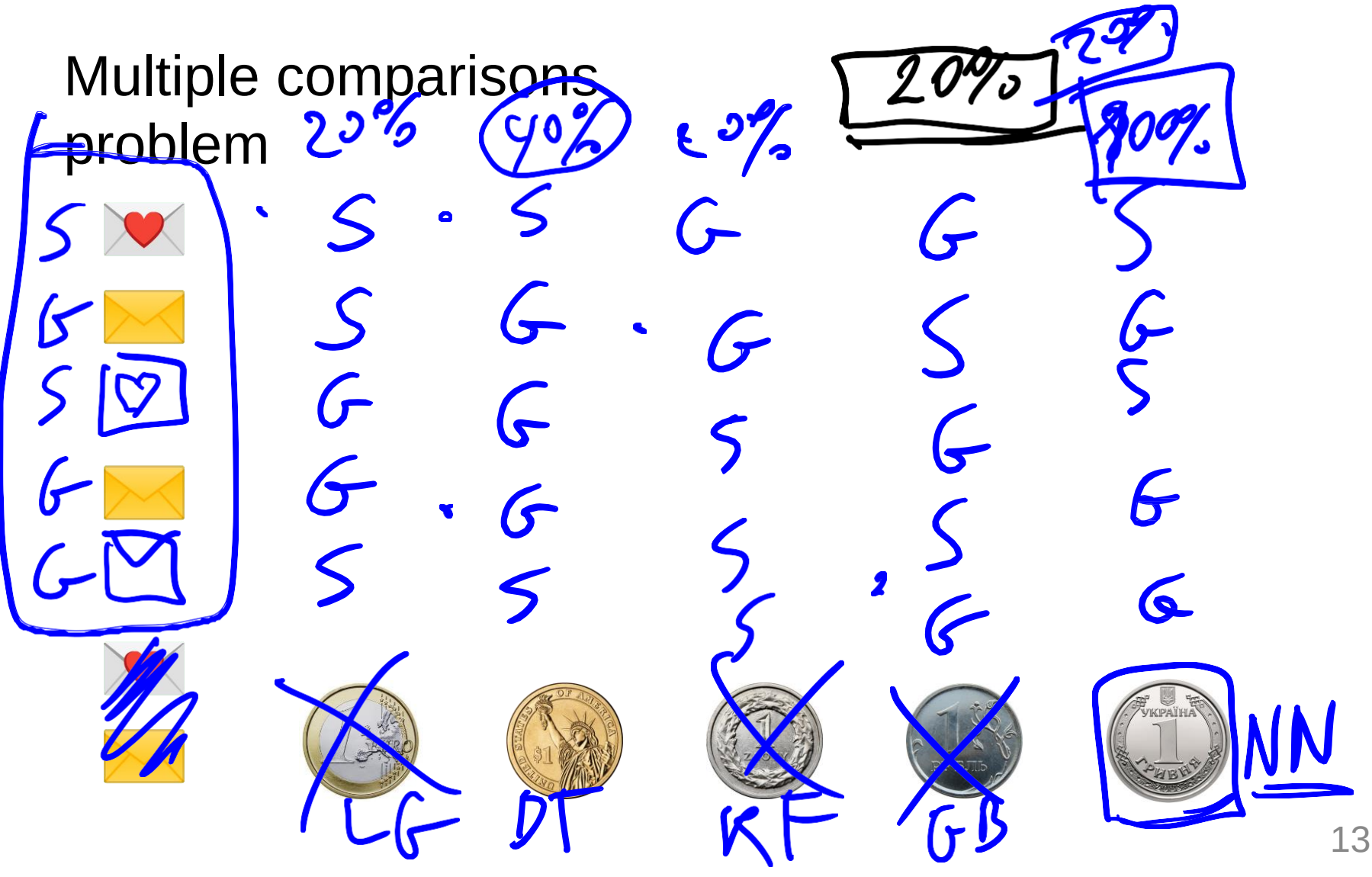

TRAIN
VAL
X
y
X_v
y_v
g
$g(X_v) = \hat{y}_v$
$\hat{y}_v$
y_v

$\hat{y}_v$			y_v
0.8	1	—	1
0.7	1	x	0
0.6	1	—	1
0.1	0	—	0
0.9	1	—	1
0.6	1	x	0
pred			target

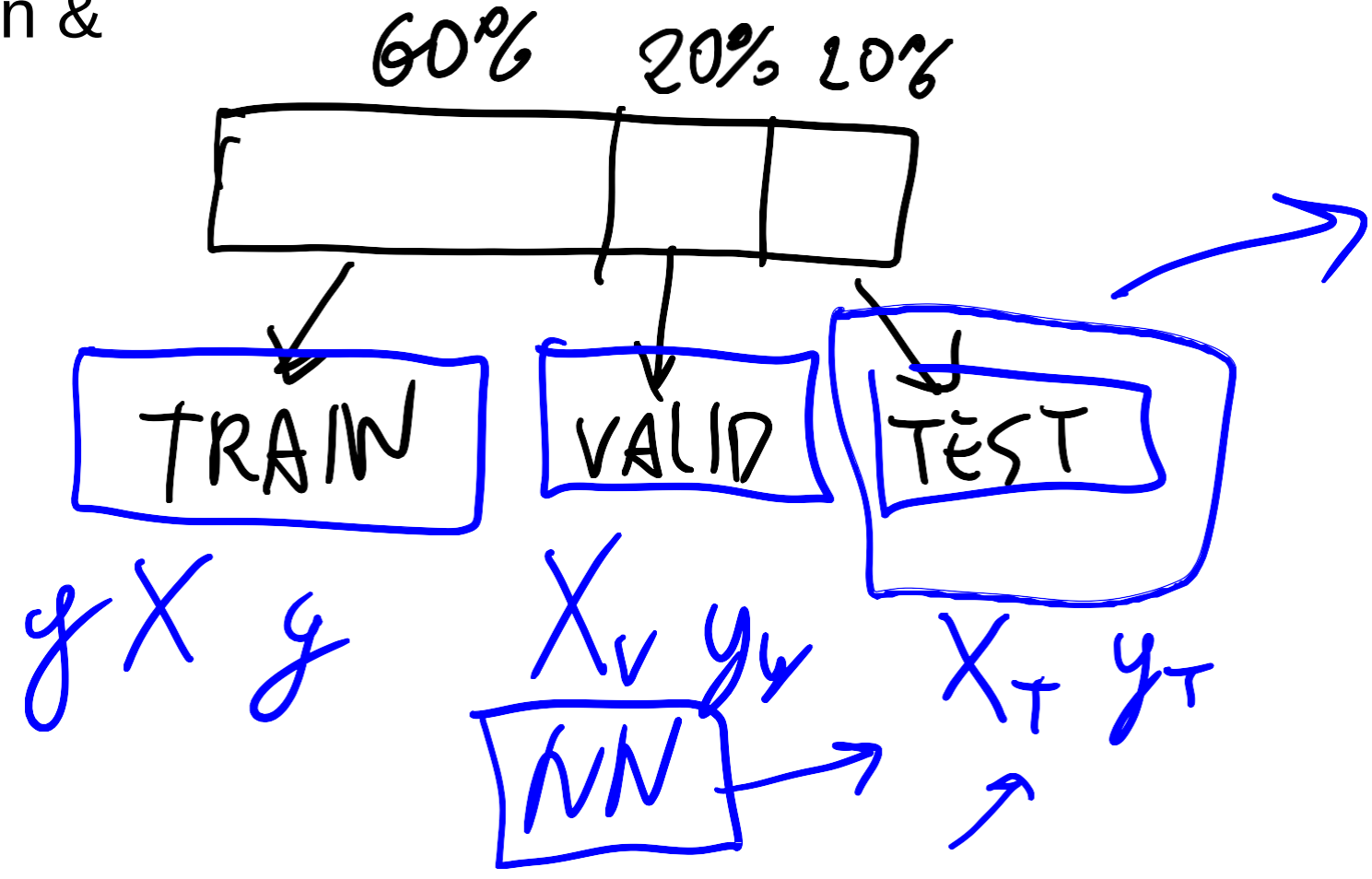
4/6 = 66%


g_1 LR 66%
g_2 DT 60%
RT 67%
NN 80%

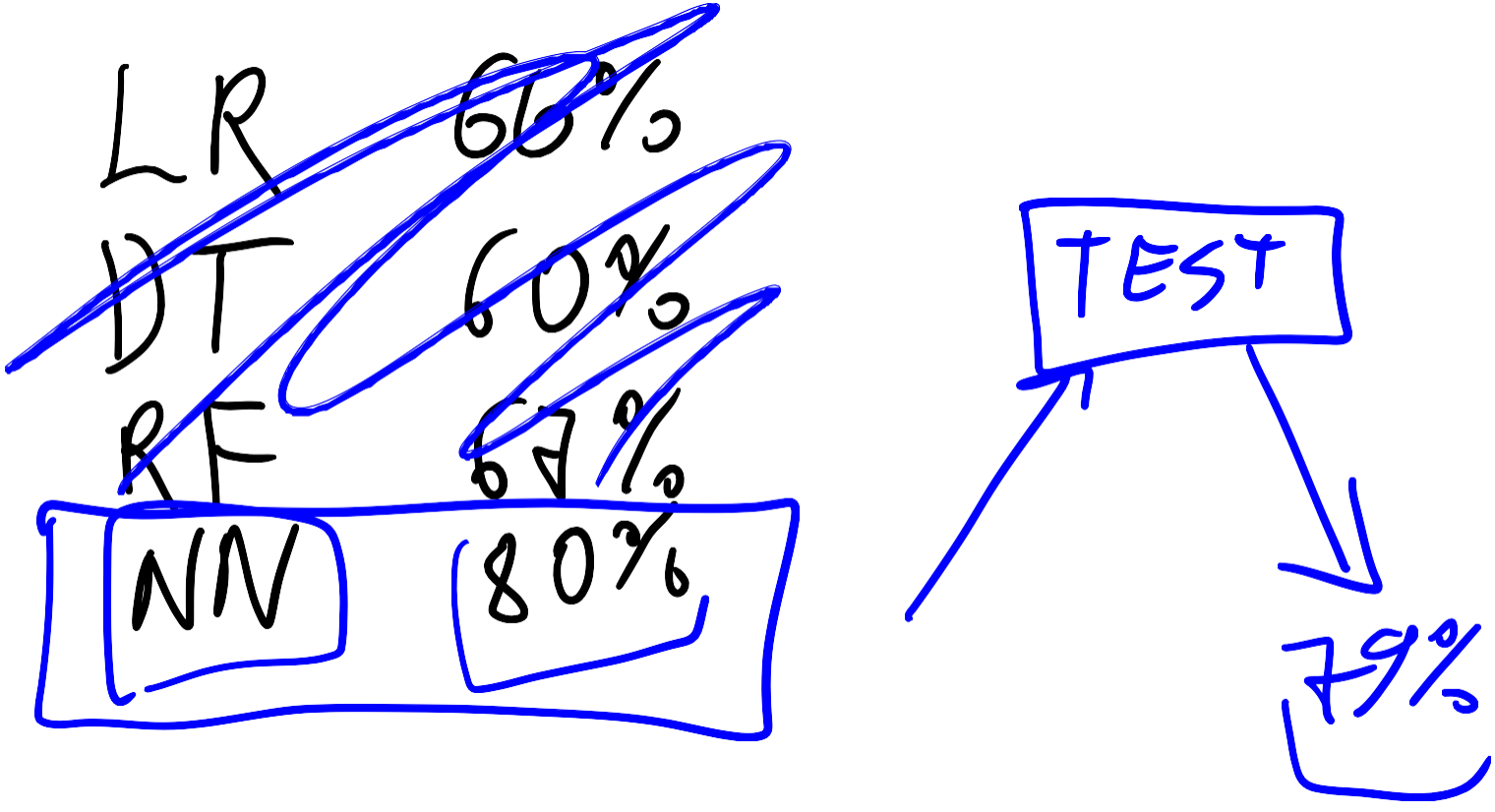
Multiple comparisons problem

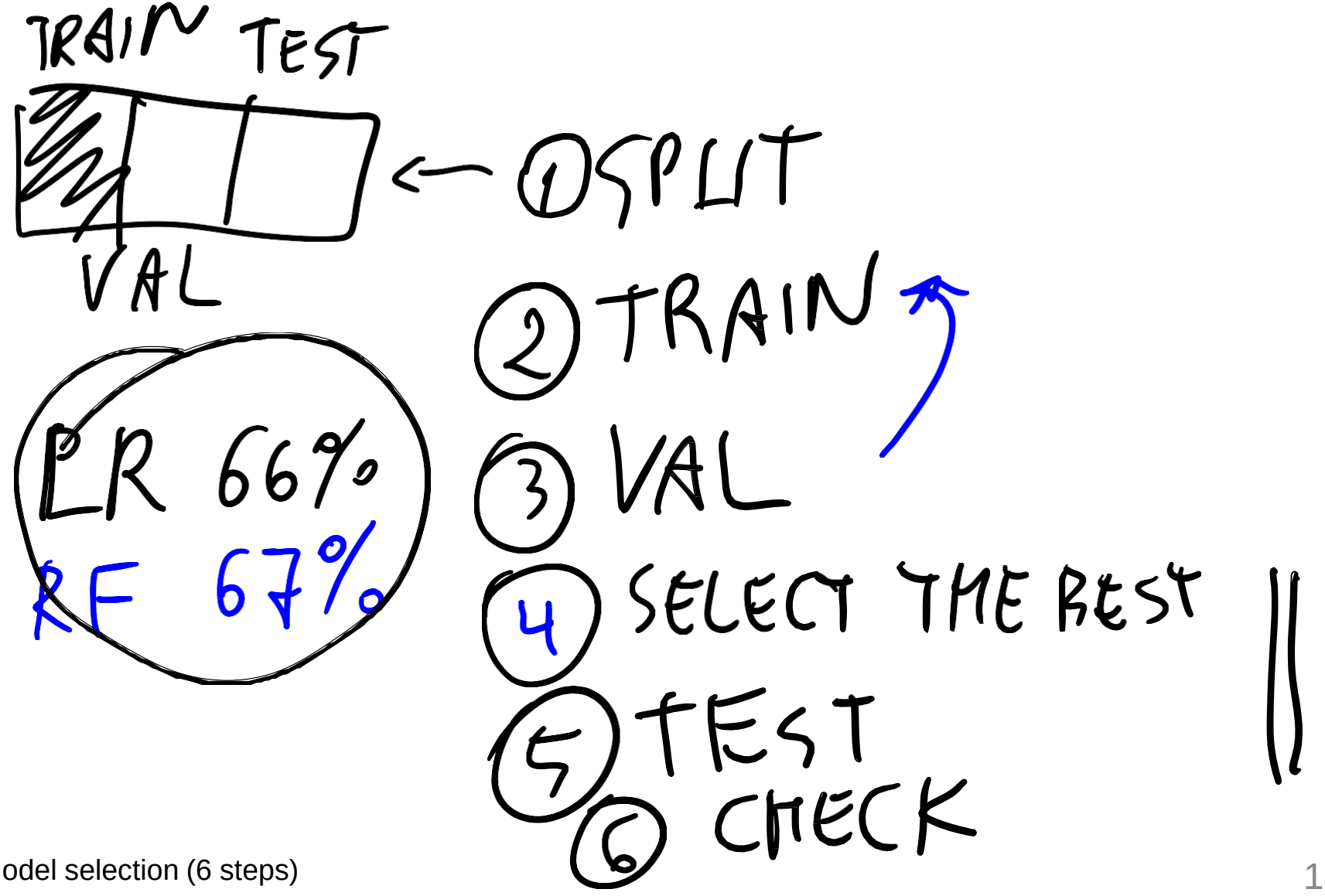


Validation & Test

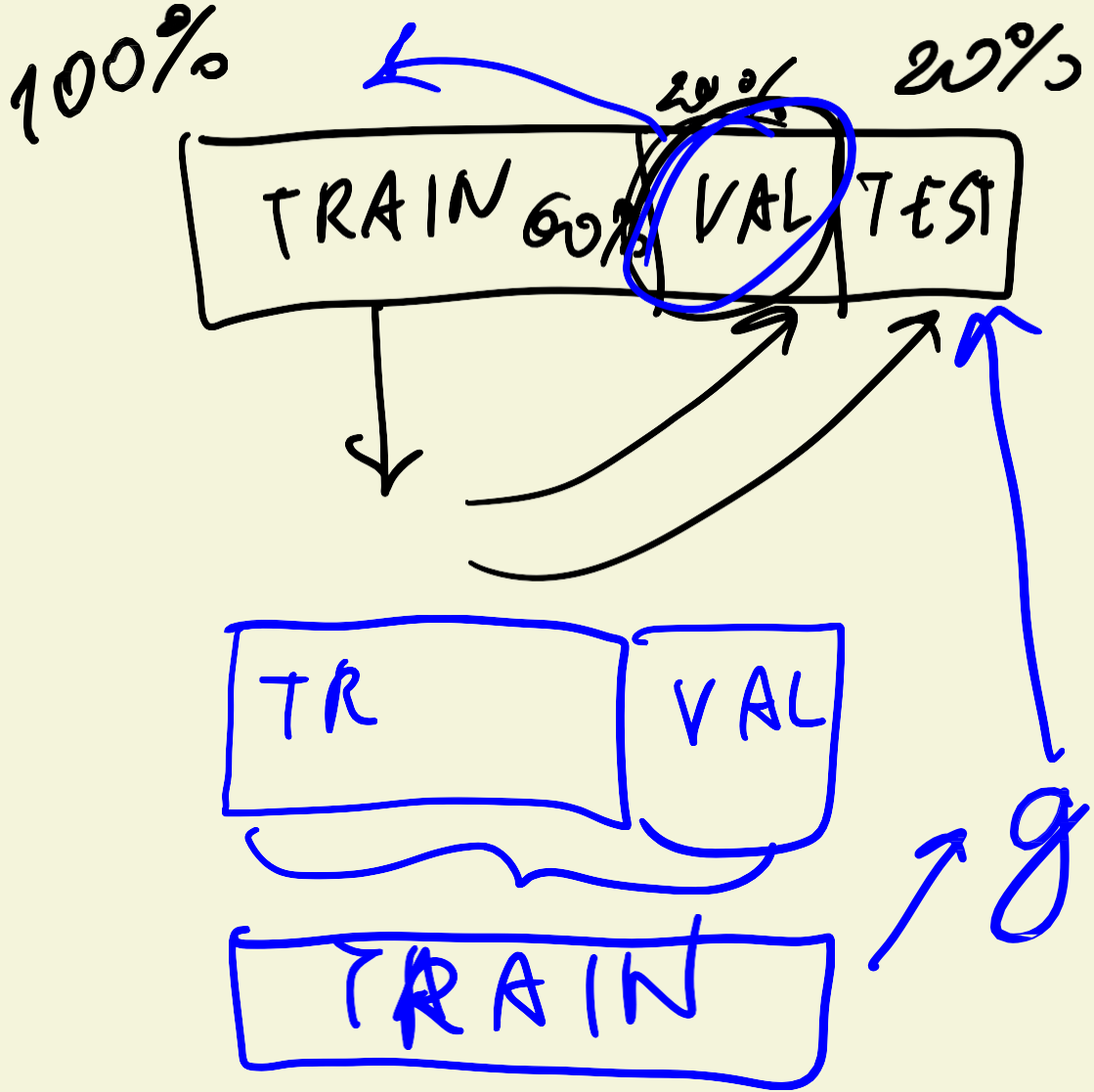


Train + validation + test


LR 66%
DT 60%
RF 67%
NN 80%
TEST
79%



Model selection (6 steps)


100%
20%
20%
TRAIN
60%
VAL
TEST
TR
VAL
TRAIN
g

Next

Introduction to NumPy

Session #1 Summary

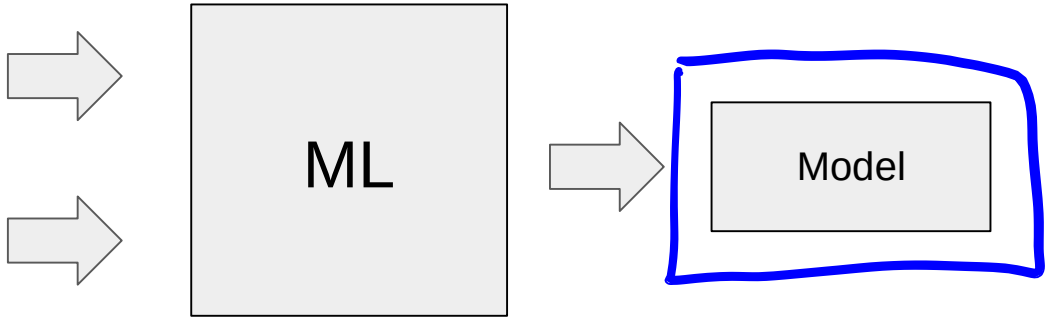
1.1 Introduction to ML

features

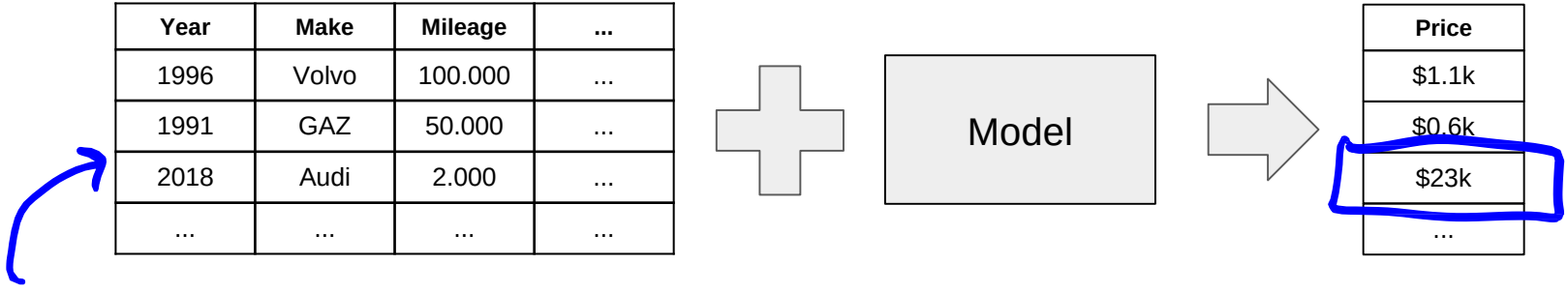
Year	Make	Mileage	...
1995	GAZ	200.000	...
1980	VAZ	100.000	...
2016	BWM	5.000	...
...	...	...	...

Price
$1.1k
$0.6k
$23k
...

target



1.1 Introduction to ML



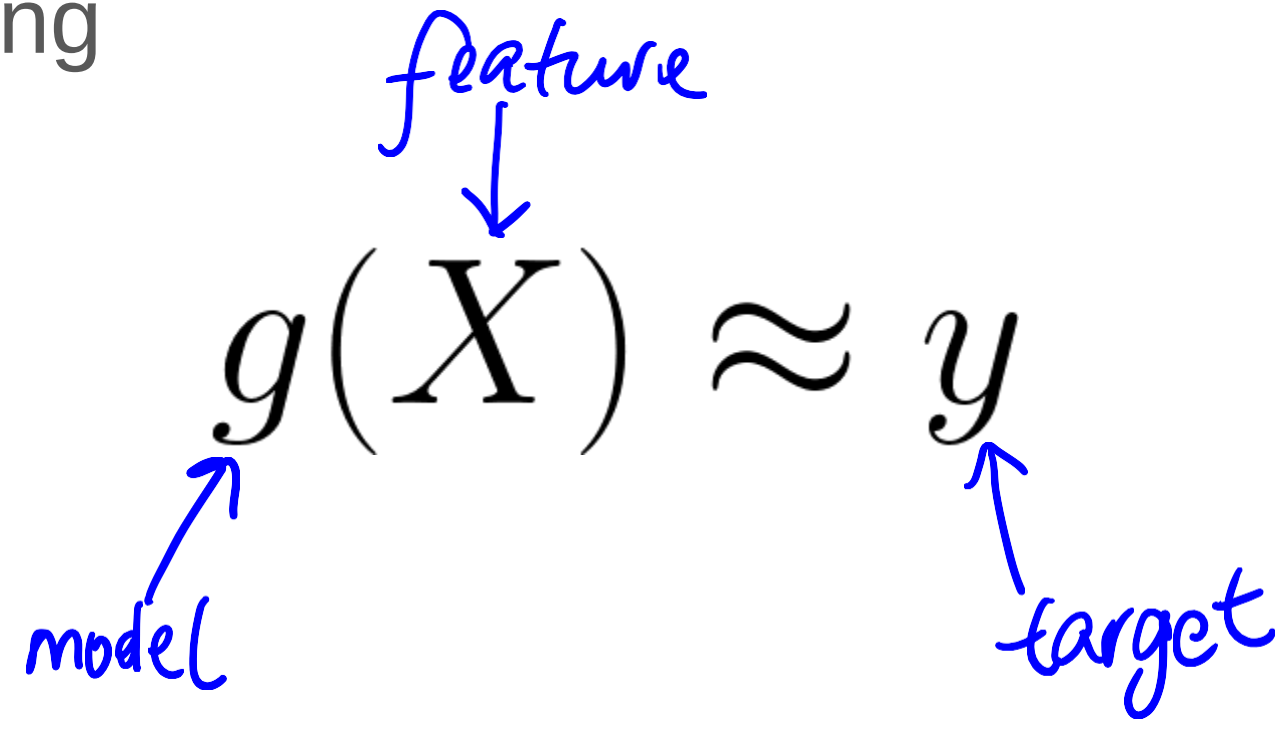
1.2 Rules vs ML

- If sender = promotions@online.com then “spam”
- If title contains “tax review” and sender domain is “online.com” then “spam”
- If body contains a word “deposit”
  - **If sender domain is “test.com” then “spam”**
  - **If body >= 100 words then spam**
- Otherwise, “good email”

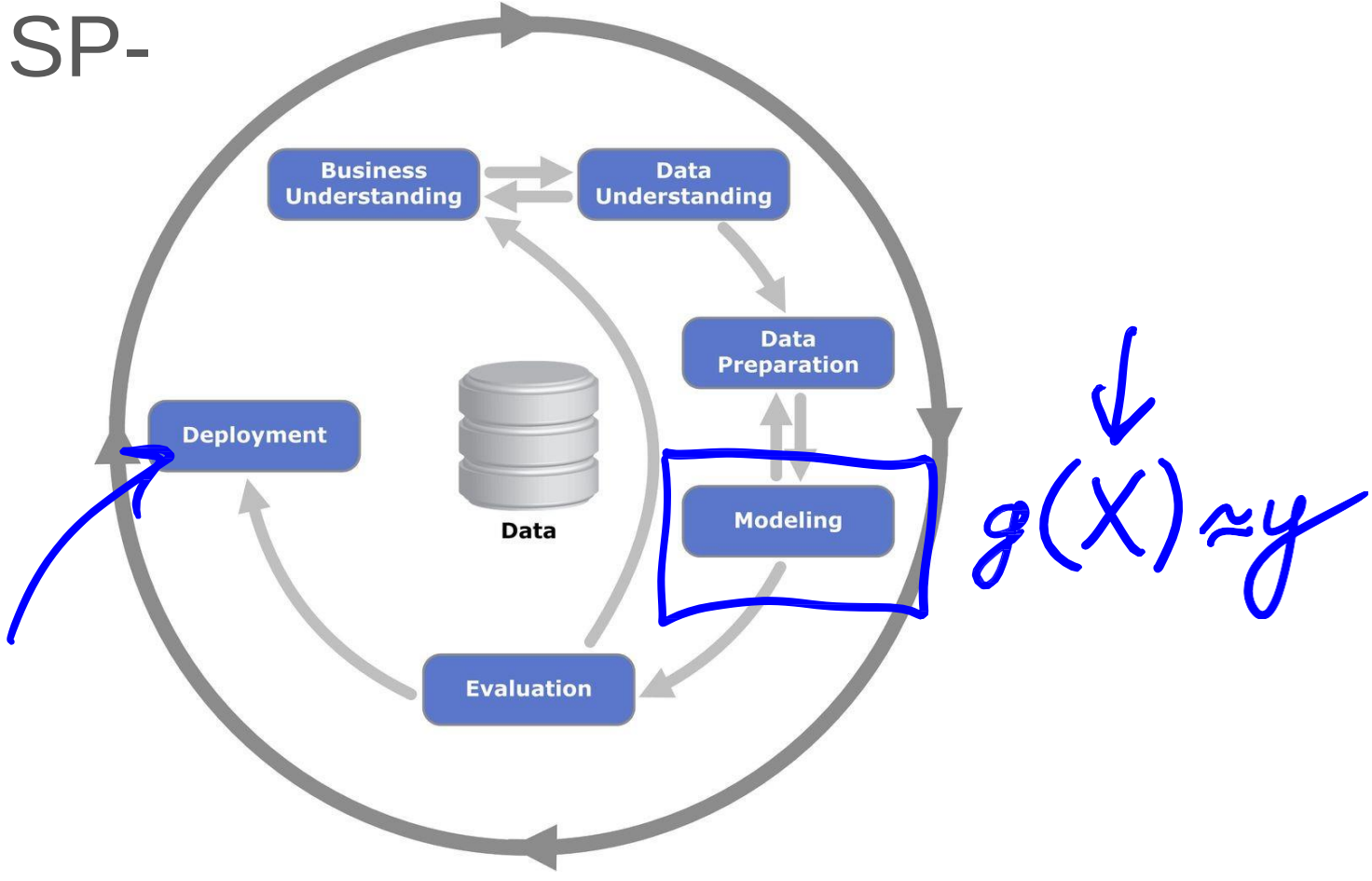
1.2 Rules vs ML

Features (data)						Target (desired output)
[1,	1,	0,	0,	1,	1]	1
[0,	0,	0,	1,	0,	1]	0
[1,	1,	1,	0,	1,	0]	1
[1,	0,	0,	0,	0,	1]	1
[0,	0,	0,	1,	1,	0]	0
[1,	0,	1,	0,	1,	1]	0

1.3 Supervised Machine Learning



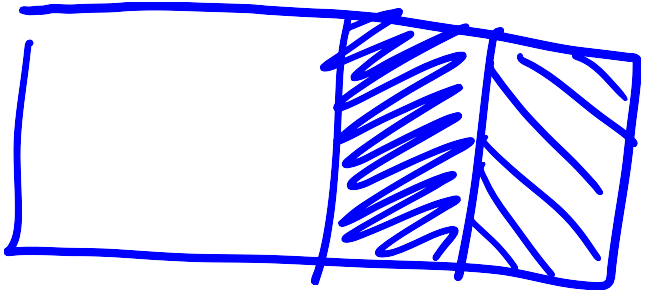
1.4 CRISP-DM



Picture: CRISP-DM

1.5 Model Selection

1. Split data into train/validation/test
2. Train a model
3. Validate it
4. Select the best model
5. Test it



1.6 Environment

- Install Python, Numpy, Pandas, Matplotlib and Scikit-Learn
- Anaconda is the easiest option
- Creating account on AWS

1.7 Introduction to NumPy

1.8 Linear algebra

- Multiplication
- Formulas are not scary when you implement them

1.9 Introduction to Pandas

Next

Session 2: Car price prediction project