Machine Learning

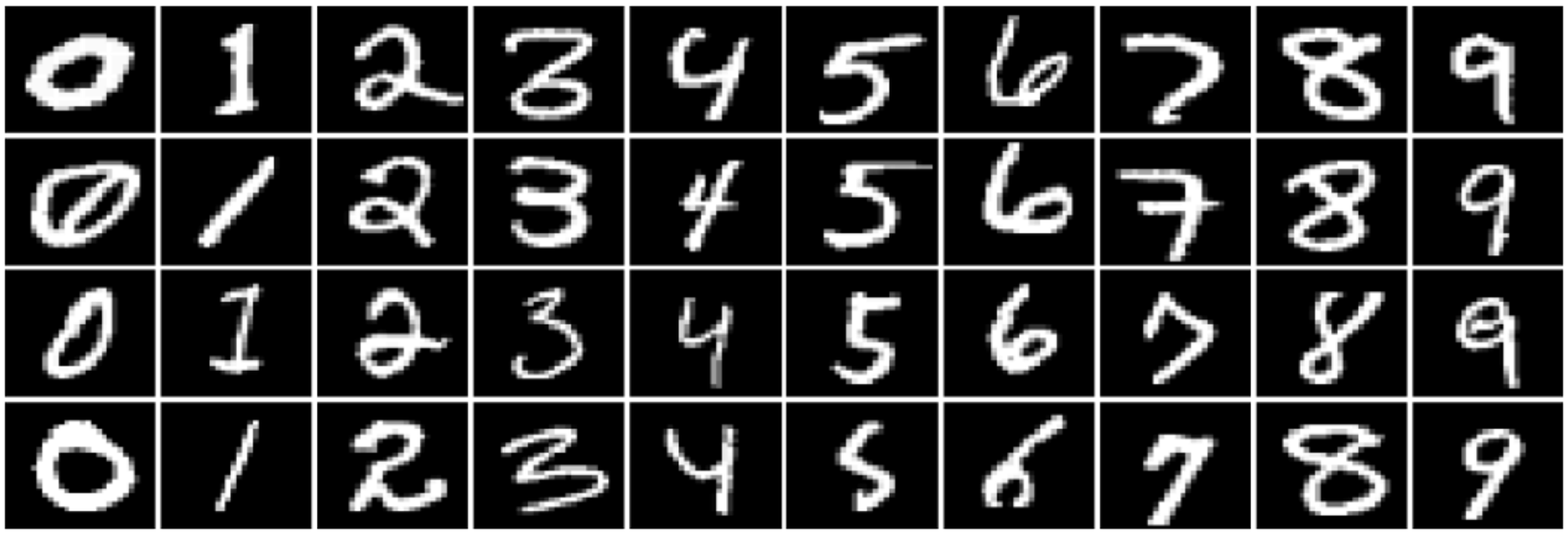
Essential concepts

by Ilya Kuzovkin
ilya.kuzovkin@gmail.com





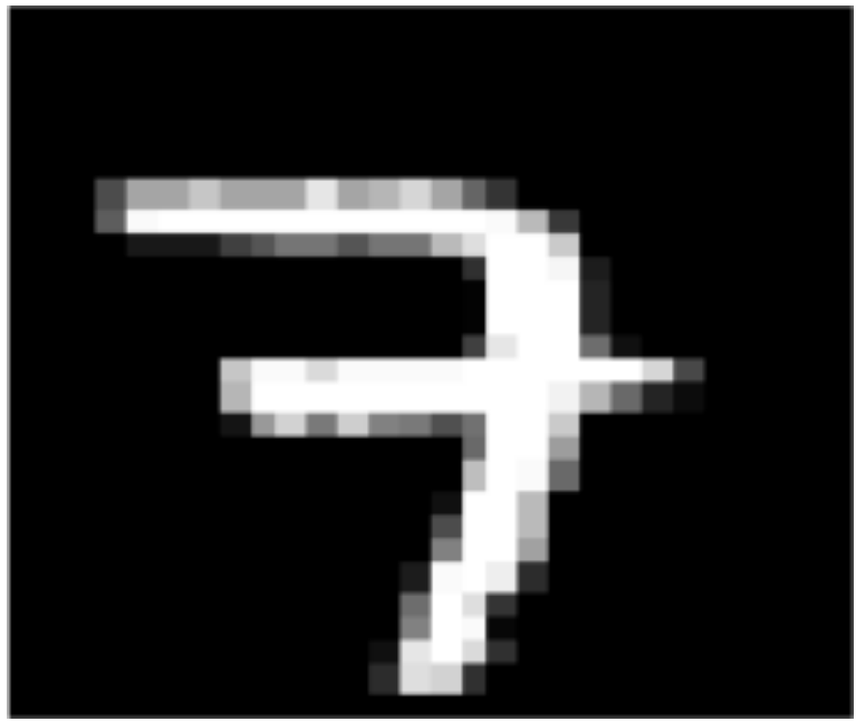
Mooncascade ML Camp
2016



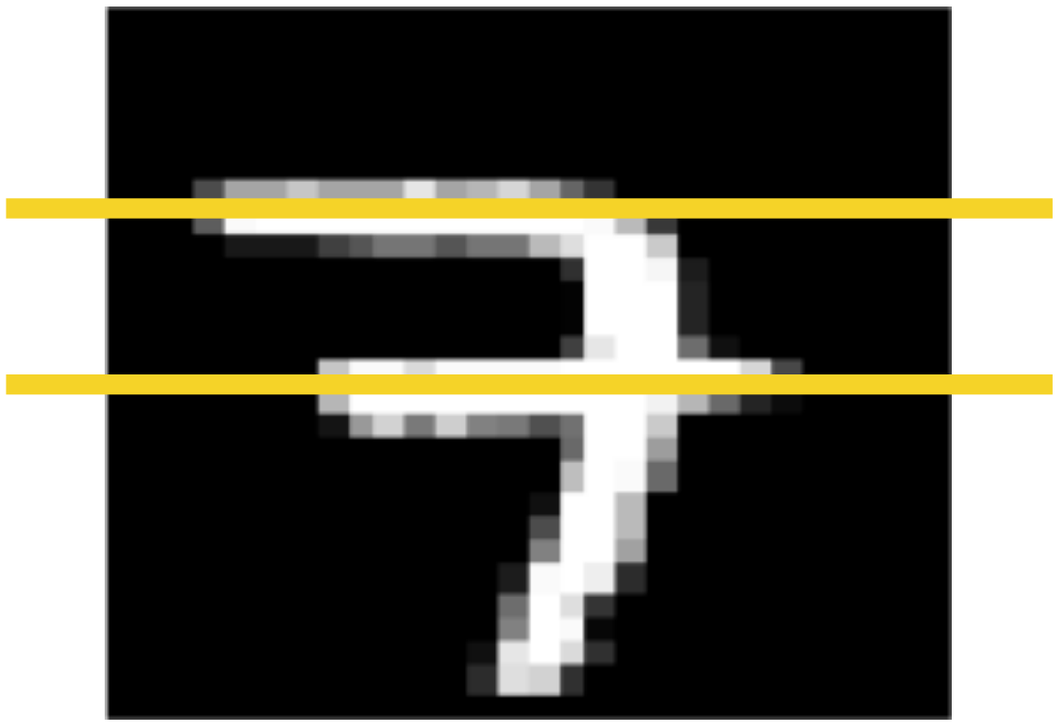
One machine learning use case



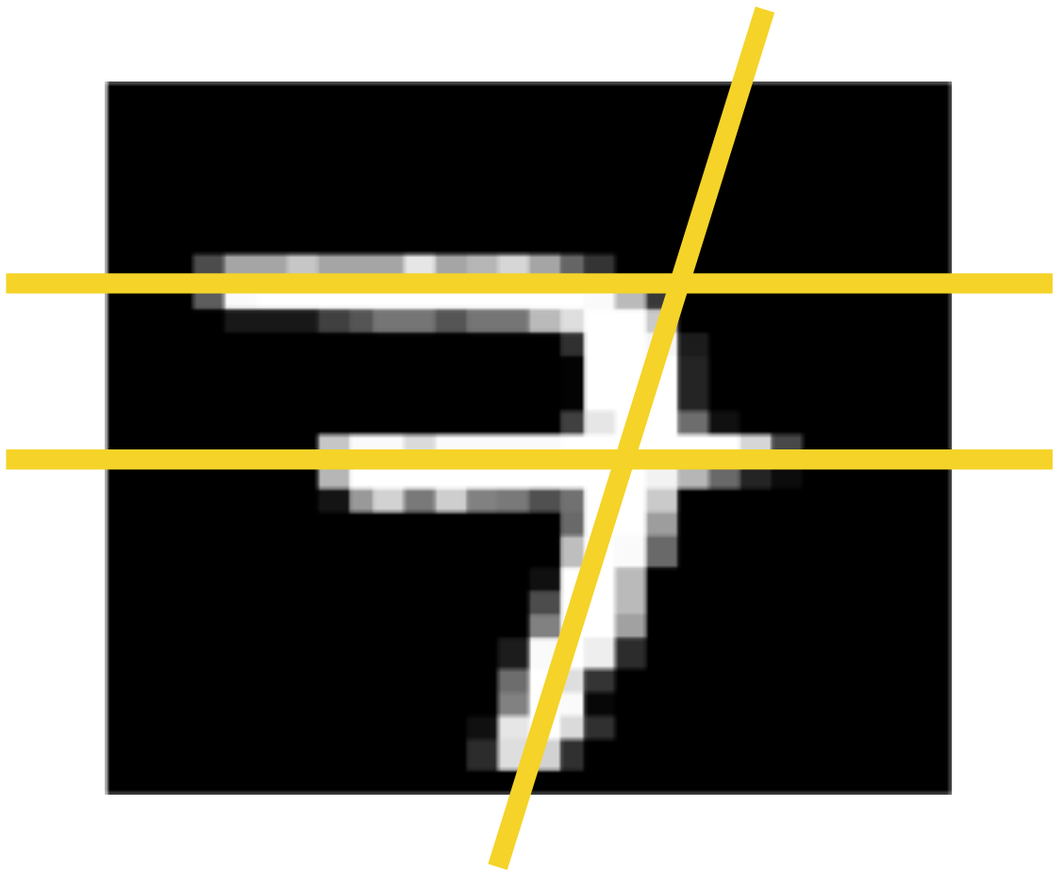




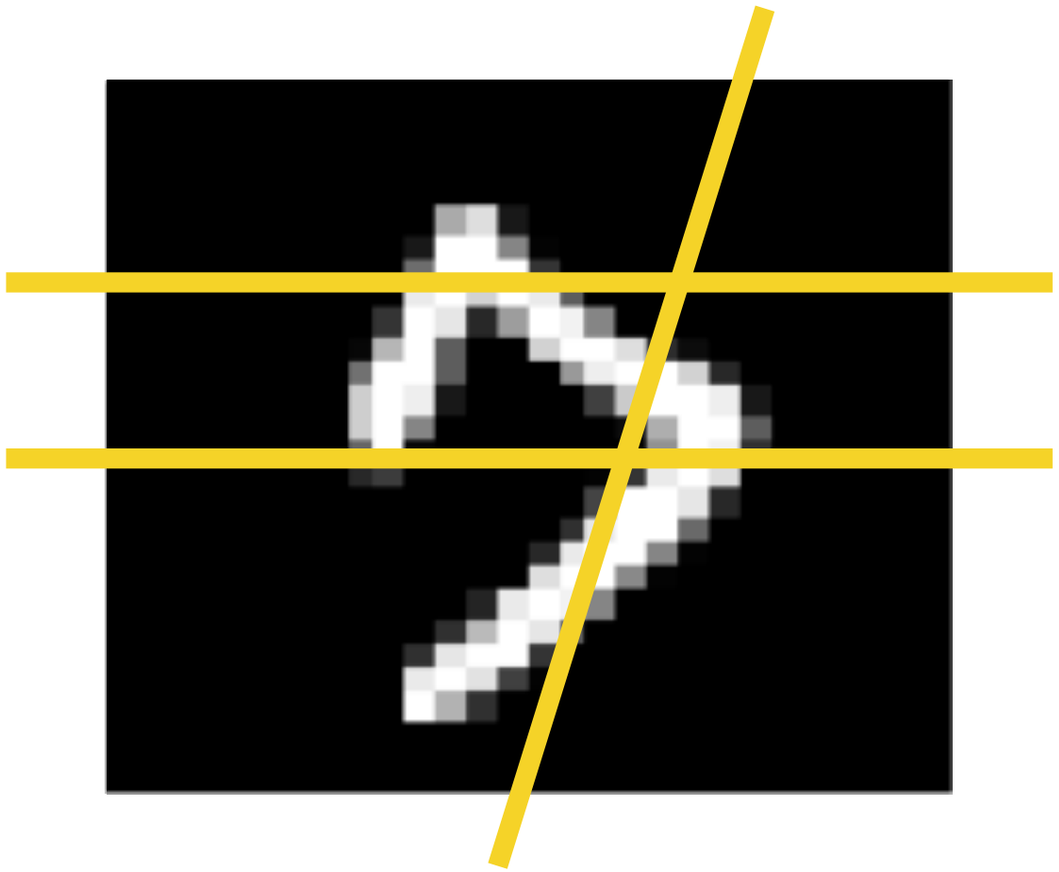




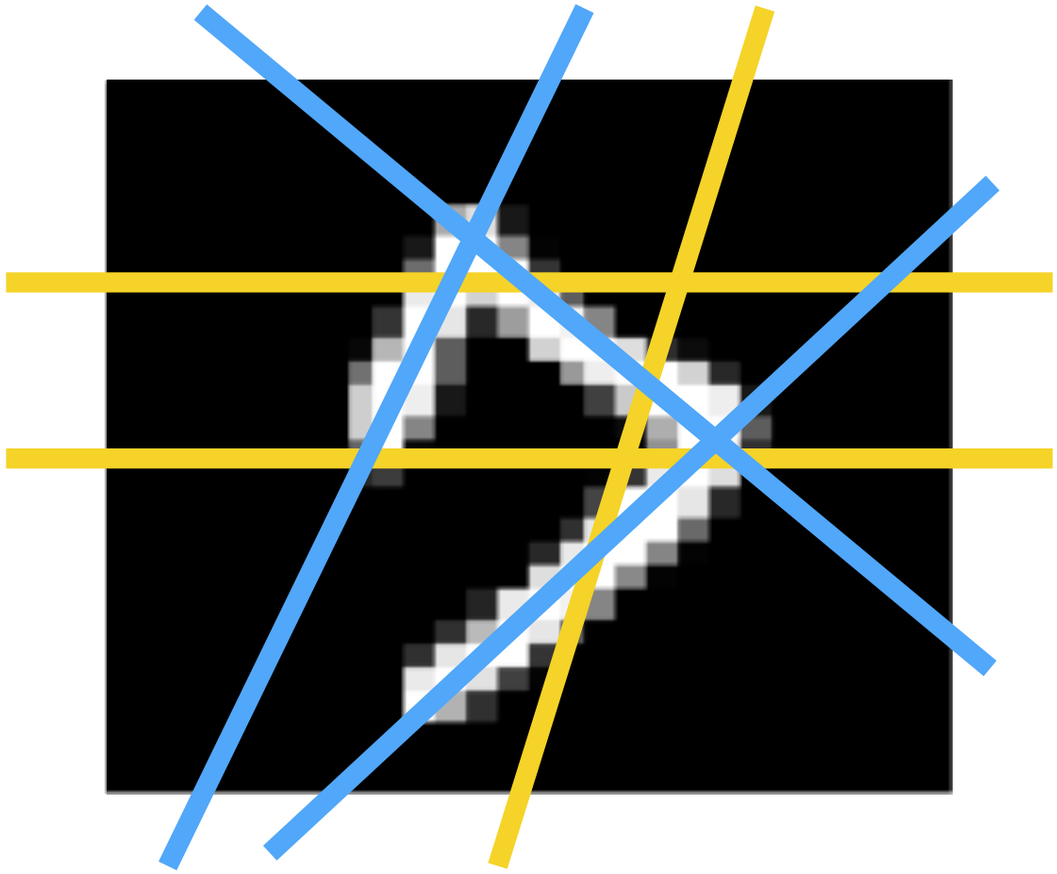


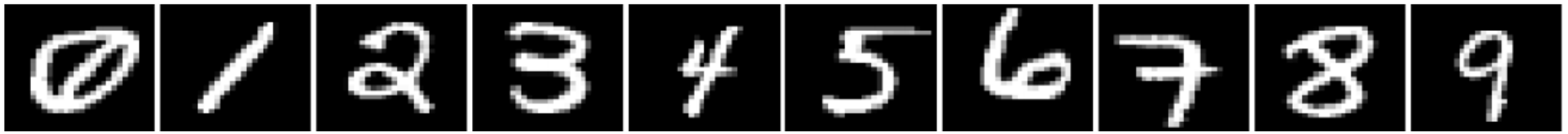


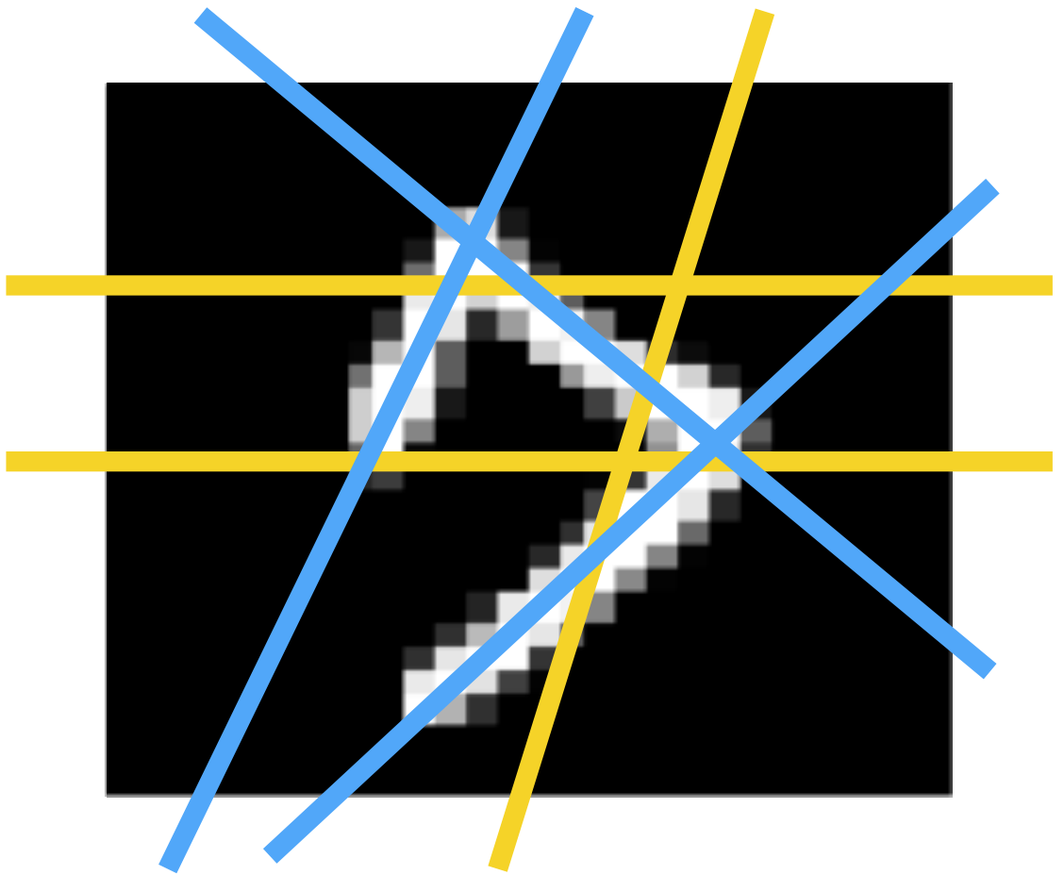


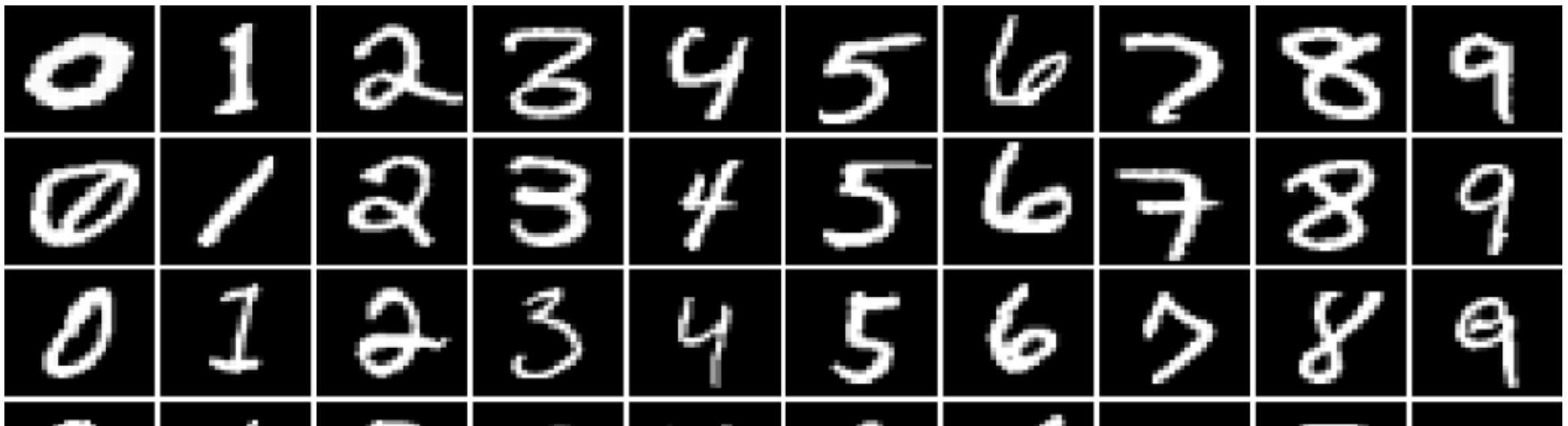






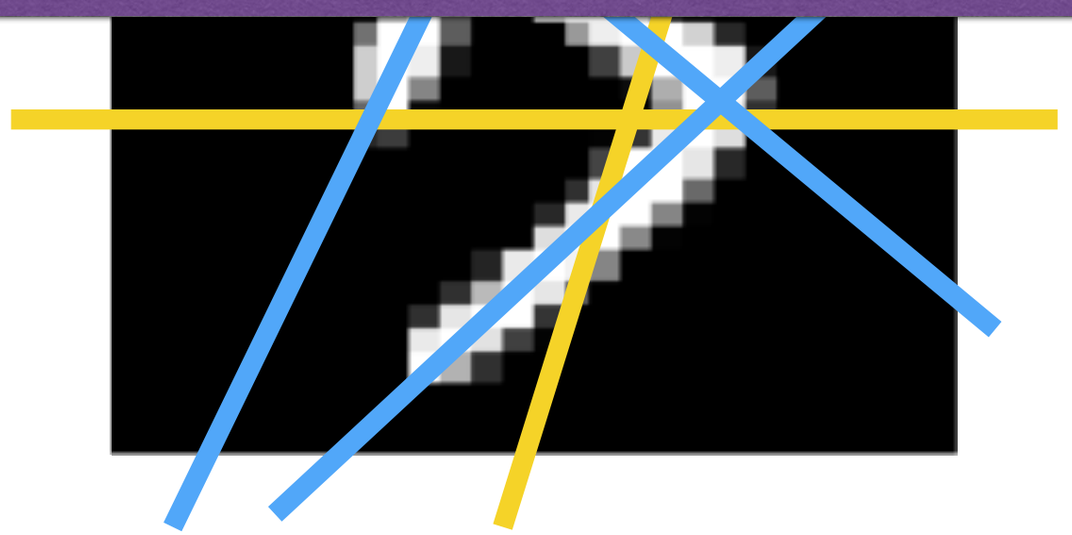


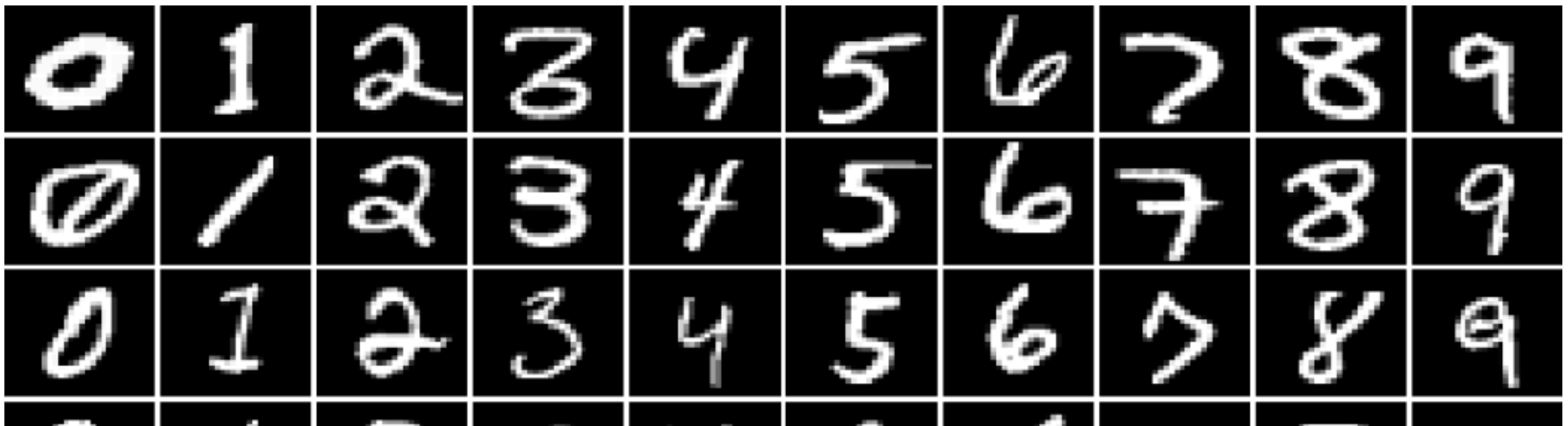






Can we ask a computer to create those patterns **automatically**?







Can we ask a computer to create those patterns **automatically**?

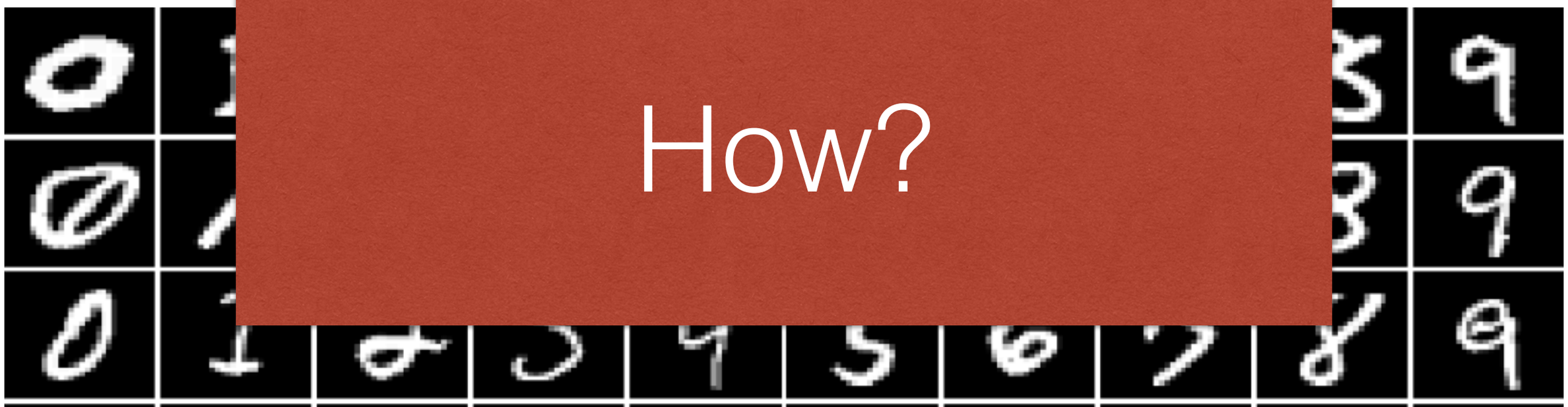
Yes



Can we ask a computer to create those patterns **automatically**?

Yes

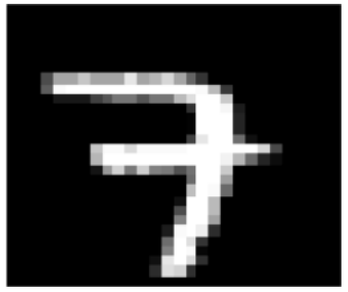
How?




Raw data

A data **sample**:

Instance



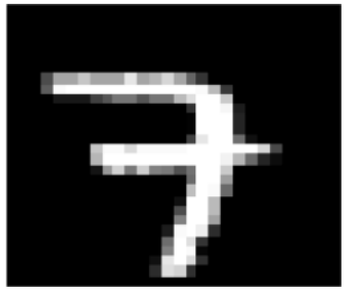
Class (label)

“7”



A data **sample**:

Instance



Class (label)

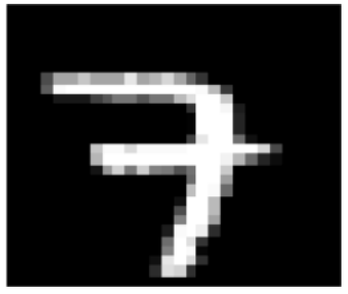
“7”



How to represent it in a machine-readable form?

A data **sample**:

Instance



Class (label)

“7”

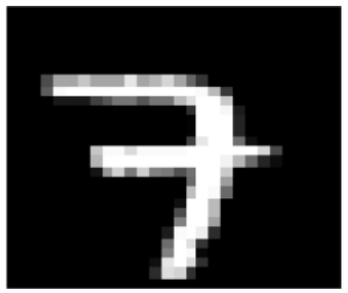


How to represent it in a machine-readable form?

Feature extraction

A data **sample**:

Instance



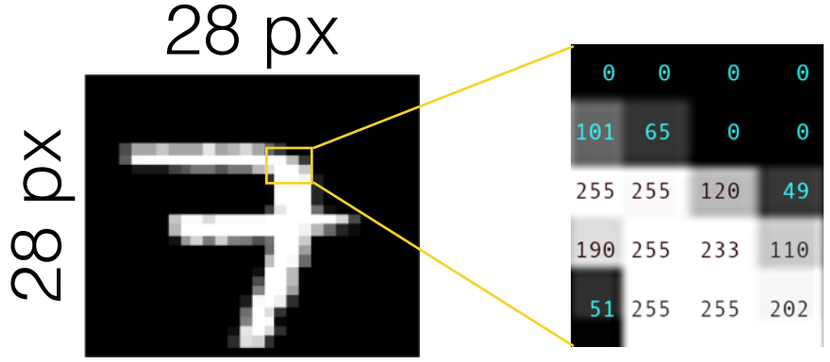
Class (label)

“7”



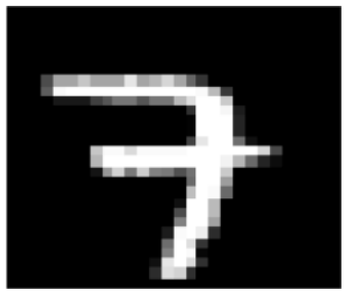
How to represent it in a machine-readable form?

Feature extraction



A data **sample**:

Instance



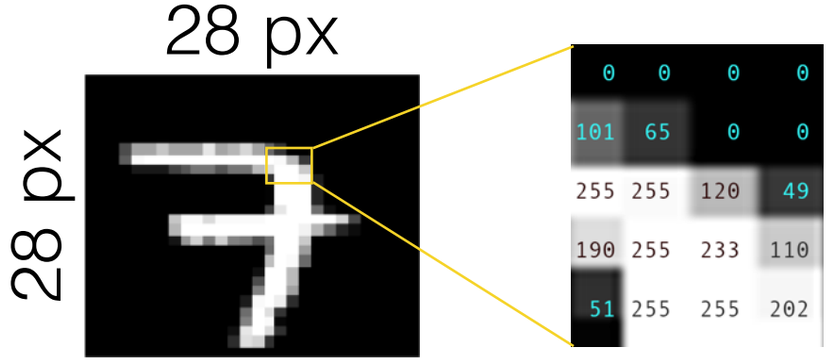
Class (label)

“7”



How to represent it in a machine-readable form?

Feature extraction



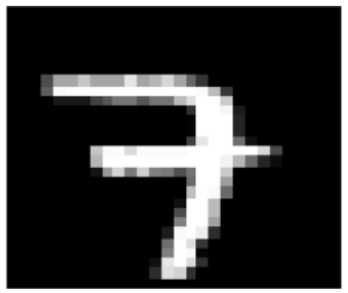
784 pixels in total

(0, 0, 0, …, 28, 65, 128, 255, 101, 38,… 0, 0, 0)

Feature vector

A data **sample**:

Instance



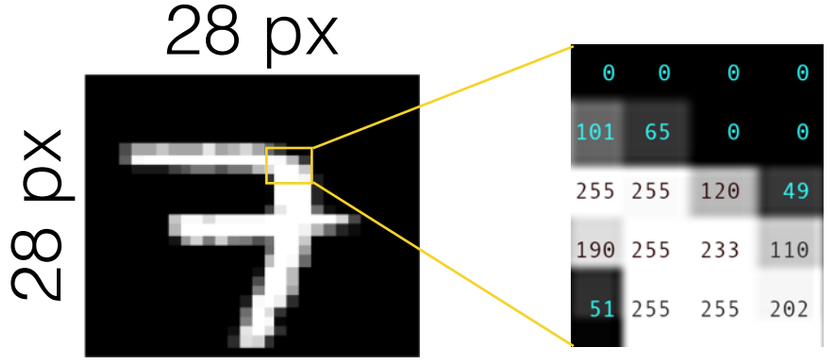
Class (label)

“7”



How to represent it in a machine-readable form?

Feature extraction



784 pixels in total

(0, 0, 0, …, 28, 65, 128, 255, 101, 38,… 0, 0, 0)

Feature vector

 → (0, 0, 0, …, 28, 65, 128, 255, 101, 38,… 0, 0, 0) “2”

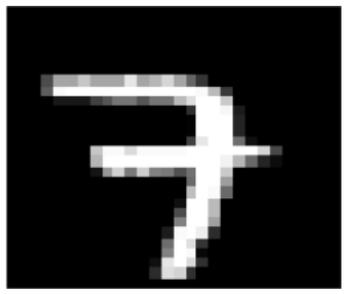
 → (0, 0, 0, …, 13, 48, 102, 0, 46, 255,… 0, 0, 0) “7”

 → (0, 0, 0, …, 17, 34, 12, 43, 122, 70,… 0, 7, 0) “8”

 → (0, 0, 0, …, 98, 21, 255, 255, 231, 140,… 0, 0, 0) “2”

A data **sample**:

Instance



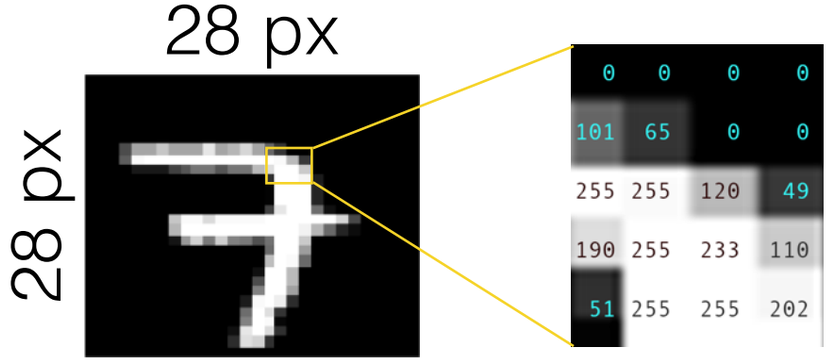
Class (label)

“7”



How to represent it in a machine-readable form?

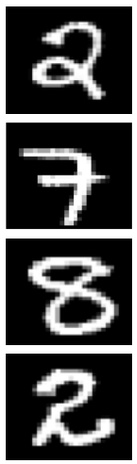
Feature extraction



784 pixels in total

(0, 0, 0, …, 28, 65, 128, 255, 101, 38,… 0, 0, 0)

Feature vector



(0, 0, 0, …, 28, 65, 128, 255, 101, 38,… 0, 0, 0) “2”

(0, 0, 0, …, 13, 48, 102, 0, 46, 255,… 0, 0, 0) “7”

(0, 0, 0, …, 17, 34, 12, 43, 122, 70,… 0, 7, 0) “8”

(0, 0, 0, …, 98, 21, 255, 255, 231, 140,… 0, 0, 0) “2”

Dataset

The data is in the right format — what's next?

The data is in the right format — what's next?

- C4.5
- Random forests
- Bayesian networks
- Hidden Markov models
- Artificial neural network
- Data clustering
- Expectation-maximization algorithm
- Self-organizing map
- Radial basis function network
- Vector Quantization
- Generative topographic map
- Information bottleneck method
- IBSEAD
- Apriori algorithm
- Eclat algorithm
- FP-growth algorithm
- Single-linkage clustering
- Conceptual clustering
- K-means algorithm
- Fuzzy clustering
- Temporal difference learning
- Q-learning
- Learning Automata
- AODE
- Artificial neural network
- Backpropagation
- Naive Bayes classifier
- Bayesian network
- Bayesian knowledge base
- Case-based reasoning
- Decision trees
- Inductive logic programming
- Gaussian process regression
- Gene expression programming
- Group method of data handling (GMDH)
- Learning Automata
- Learning Vector Quantization
- Logistic Model Tree
- Decision tree
- Decision graphs
- Lazy learning
- Monte Carlo Method
- SARSA
- Instance-based learning
- Nearest Neighbor Algorithm
- Analogical modeling
- Probably approximately correct learning (PACL)
- Symbolic machine learning algorithms
- Subsymbolic machine learning algorithms
- Support vector machines
- Random Forest
- Ensembles of classifiers
- Bootstrap aggregating (bagging)
- Boosting (meta-algorithm)
- Ordinal classification
- Regression analysis
- Information fuzzy networks (IFN)
- Linear classifiers
- Fisher's linear discriminant
- Logistic regression
- Naive Bayes classifier
- Perceptron
- Support vector machines
- Quadratic classifiers
- k-nearest neighbor
- Boosting

Pick an algorithm

The data is in the right format — what's next?

- C4.5
- Random forests
- Bayesian networks
- Hidden Markov models
- Artificial neural network
- Data clustering
- Expectation-maximization algorithm
- Self-organizing map
- Radial basis function network
- Vector Quantization
- Generative topographic map
- Information bottleneck method
- IBSEAD
- Apriori algorithm
- Eclat algorithm
- FP-growth algorithm
- Single-linkage clustering
- Conceptual clustering
- K-means algorithm
- Fuzzy clustering
- Temporal difference learning
- Q-learning
- Learning Automata
- AODE
- Artificial neural network
- Backpropagation
- Naive Bayes classifier
- Bayesian network
- Bayesian knowledge base
- Case-based reasoning
- Decision trees
- Inductive logic programming
- Gaussian process regression
- Gene expression programming
- Group method of data handling (GMDH)
- Learning Automata
- Learning Vector Quantization
- Logistic Model Tree
- Decision tree
- Decision graphs
- Lazy learning
- Monte Carlo Method
- SARSA
- Instance-based learning
- Nearest Neighbor Algorithm
- Analogical modeling
- Probably approximately correct learning (PACL)
- Symbolic machine learning algorithms
- Subsymbolic machine learning algorithms
- Support vector machines
- Random Forest
- Ensembles of classifiers
- Bootstrap aggregating (bagging)
- Boosting (meta-algorithm)
- Ordinal classification
- Regression analysis
- Information fuzzy networks (IFN)
- Linear classifiers
- Fisher's linear discriminant
- Logistic regression
- Naive Bayes classifier
- Perceptron
- Support vector machines
- Quadratic classifiers
- k-nearest neighbor
- Boosting

Pick an algorithm

Decision tree

 vs.

DECISION TREE

 vs. 

(0, …, 28, 65, …, 207, 101, 0, 0) 

(0, …, 19, 34, …, 254, 54, 0, 0) 

(0, …, 87, 59, …, 240, 52, 4, 0) 

(0, …, 87, 52, …, 240, 19, 3, 0) 

(0, …, 28, 64, …, 102, 101, 0, 0) 

(0, …, 19, 23, …, 105, 54, 0, 0) 

(0, …, 87, 74, …, 121, 51, 7, 0) 

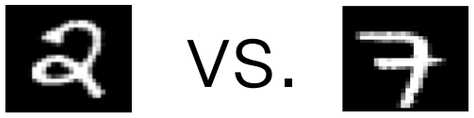
(0, …, 87, 112, …, 239, 52, 4, 0) 

Decision tree

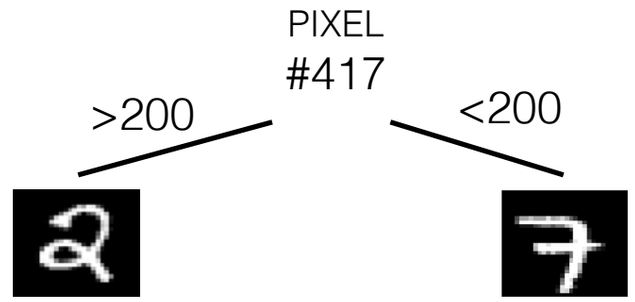
 vs.

			PIXEL #417		
(0, …,	28,	65, …,	207,	101, 0, 0)	
(0, …,	19,	34, …,	254,	54, 0, 0)	
(0, …,	87,	59, …,	240,	52, 4, 0)	
(0, …,	87,	52, …,	240,	19, 3, 0)	
(0, …,	28,	64, …,	102,	101, 0, 0)	
(0, …,	19,	23, …,	105,	54, 0, 0)	
(0, …,	87,	74, …,	121,	51, 7, 0)	
(0, …,	87,	112, …,	239,	52, 4, 0)	

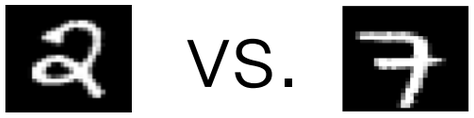
Decision tree



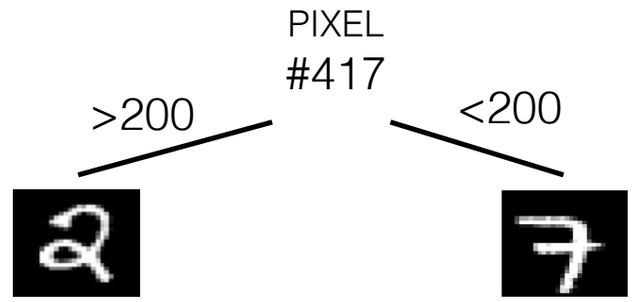
					PIXEL #417				
(0,	…,	28,	65,	…,	207,	101,	0,	0)	
(0,	…,	19,	34,	…,	254,	54,	0,	0)	
(0,	…,	87,	59,	…,	240,	52,	4,	0)	
(0,	…,	87,	52,	…,	240,	19,	3,	0)	
(0,	…,	28,	64,	…,	102,	101,	0,	0)	
(0,	…,	19,	23,	…,	105,	54,	0,	0)	
(0,	…,	87,	74,	…,	121,	51,	7,	0)	
(0,	…,	87,	112,	…,	239,	52,	4,	0)	



Decision tree



					PIXEL #417					
(0,	…,	28,	65,	…,	207,	101,	0,	0)	
(0,	…,	19,	34,	…,	254,	54,	0,	0)	
(0,	…,	87,	59,	…,	240,	52,	4,	0)	
(0,	…,	87,	52,	…,	240,	19,	3,	0)	
(0,	…,	28,	64,	…,	102,	101,	0,	0)	
(0,	…,	19,	23,	…,	105,	54,	0,	0)	
(0,	…,	87,	74,	…,	121,	51,	7,	0)	
(0,	…,	87,	112,	…,	239,	52,	4,	0)	



Decision tree



(0, …, 28, 65, …, 207, 101, 0, 0)

(0, …, 19, 34, …, 254, 54, 0, 0)

(0, …, 87, 59, …, 240, 52, 4, 0)

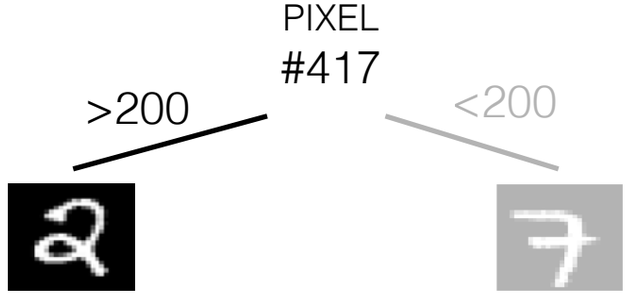
(0, …, 87, 52, …, 240, 19, 3, 0)

(0, …, 28, 64, …, 102, 101, 0, 0)

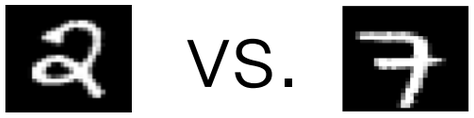
(0, …, 19, 23, …, 105, 54, 0, 0)

(0, …, 87, 74, …, 121, 51, 7, 0)

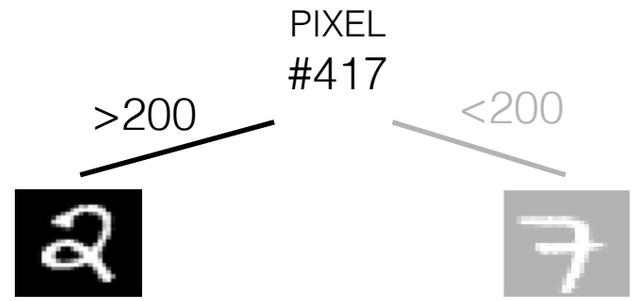
(0, …, 87, 112, …, 239, 52, 4, 0)



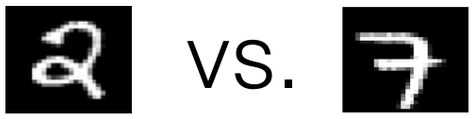
Decision tree



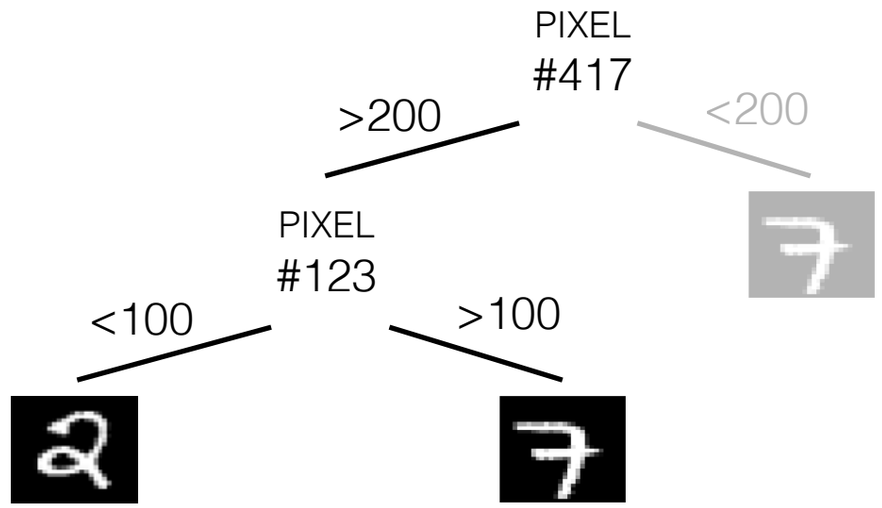
	PIXEL #123					
(0, …, 28,	65,	…,	207,	101,	0,	0)
(0, …, 19,	34,	…,	254,	54,	0,	0)
(0, …, 87,	59,	…,	240,	52,	4,	0)
(0, …, 87,	52,	…,	240,	19,	3,	0)
(0, …, 28,	64,	…,	102,	101,	0,	0)
(0, …, 19,	23,	…,	105,	54,	0,	0)
(0, …, 87,	74,	…,	121,	51,	7,	0)
(0, …, 87,	112,	…,	239,	52,	4,	0)



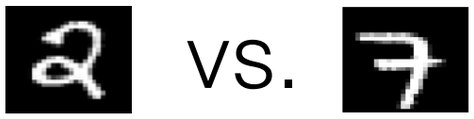
Decision tree


vs.

	PIXEL #123					
(0, …, 28,	65,	…,	207,	101,	0,	0) 
(0, …, 19,	34,	…,	254,	54,	0,	0) 
(0, …, 87,	59,	…,	240,	52,	4,	0) 
(0, …, 87,	52,	…,	240,	19,	3,	0) 
(0, …, 28,	64,	…,	102,	101,	0,	0) 
(0, …, 19,	23,	…,	105,	54,	0,	0) 
(0, …, 87,	74,	…,	121,	51,	7,	0) 
(0, …, 87,	112,	…,	239,	52,	4,	0) 



Decision tree


vs.

(0, …, 28, 65, …, 207, 101, 0, 0) 

(0, …, 19, 34, …, 254, 54, 0, 0) 

(0, …, 87, 59, …, 240, 52, 4, 0) 

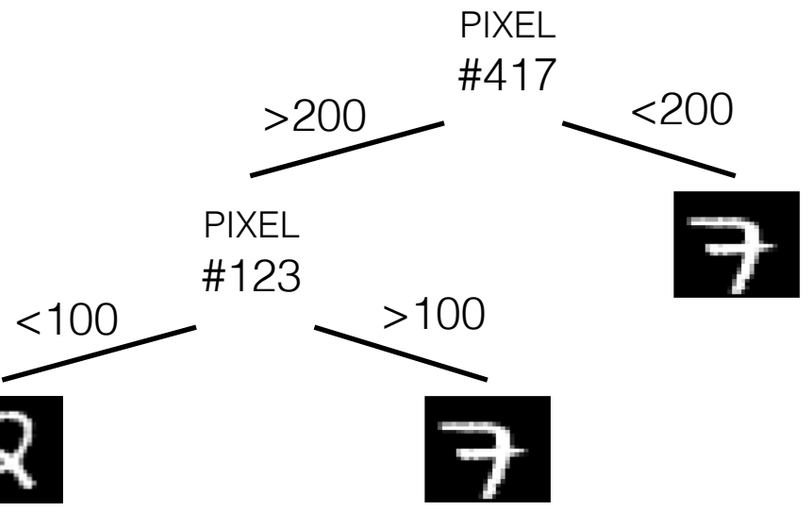
(0, …, 87, 52, …, 240, 19, 3, 0) 

(0, …, 28, 64, …, 102, 101, 0, 0) 

(0, …, 19, 23, …, 105, 54, 0, 0) 

(0, …, 87, 74, …, 121, 51, 7, 0) 

(0, …, 87, 112, …, 239, 52, 4, 0) 





Decision tree

```
from sklearn.datasets import fetch_mldata
from sklearn import tree

# load data
mnist = fetch_mldata('MNIST original')
print 'The training set has %d samples' % mnist.data.shape[0]

# train a decision tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(mnist.data, mnist.target)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
```

Decision tree

```
from sklearn.datasets import fetch_mldata
from sklearn import tree

# load data
mnist = fetch_mldata('MNIST original')
print 'The training set has %d samples' % mnist.data.shape[0]

# train a decision tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(mnist.data, mnist.target)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
```

```
The training set has 70000 samples
Accuracy on the training set is 1.0000
```

Accuracy

```
The training set has 70000 samples
Accuracy on the training set is 1.0000
```

ACCURACY

```
The training set has 70000 samples
Accuracy on the training set is 1.0000
```

Predicted class

True class

	0	1	2	3	4	5	6	7	8	9
0	973	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	10	9	996	2	0	0	0	13	2	0
3	0	2	4	976	1	13	1	7	3	3
4	1	6	0	0	950	0	4	2	0	19
5	6	1	0	11	2	859	5	1	3	4
6	5	3	0	0	3	3	944	0	0	0
7	0	21	5	0	1	0	0	991	0	10
8	8	2	4	16	8	11	3	4	914	4
9	4	5	2	8	9	2	1	8	2	968

Confusion matrix

Accuracy

```
The training set has 70000 samples
Accuracy on the training set is 1.0000
```

Predicted class

True class	0	1	2	3	4	5	6	7	8	9
0	973	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	10	9	996	2	0	0	0	13	2	0
3	0	2	4	976	1	13	1	7	3	3
4	1	6	0	0	950	0	4	2	0	19
5	6	1	0	11	2	859	5	1	3	4
6	5	3	0	0	3	3	944	0	0	0
7	0	21	5	0	1	0	0	991	0	10
8	8	2	4	16	8	11	3	4	914	4
9	4	5	2	8	9	2	1	8	2	968

Confusion matrix

$$\text{acc} = \frac{\text{correctly classified}}{\text{total number of samples}}$$

ACCURACY

```
The training set has 70000 samples
Accuracy on the training set is 1.0000
```

Predicted class

True class

	0	1	2	3	4	5	6	7	8	9
0	973	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	10	9	996	2	0	0	0	13	2	0
3	0	2	4	976	1	13	1	7	3	3
4	1	6	0	0	950	0	4	2	0	19
5	6	1	0	11	2	859	5	1	3	4
6	5	3	0	0	3	3	944	0	0	0
7	0	21	5	0	1	0	0	991	0	10
8	8	2	4	16	8	11	3	4	914	4
9	4	5	2	8	9	2	1	8	2	968

Confusion matrix

$$\text{acc} = \frac{\text{correctly classified}}{\text{total number of samples}}$$

Beware of an **imbalanced** dataset!

ACCURACY

```
The training set has 70000 samples
Accuracy on the training set is 1.0000
```

Predicted class

True class	0	1	2	3	4	5	6	7	8	9
0	973	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	10	9	996	2	0	0	0	13	2	0
3	0	2	4	976	1	13	1	7	3	3
4	1	6	0	0	950	0	4	2	0	19
5	6	1	0	11	2	859	5	1	3	4
6	5	3	0	0	3	3	944	0	0	0
7	0	21	5	0	1	0	0	991	0	10
8	8	2	4	16	8	11	3	4	914	4
9	4	5	2	8	9	2	1	8	2	968

Confusion matrix

$$\text{acc} = \frac{\text{correctly classified}}{\text{total number of samples}}$$

Beware of an **imbalanced** dataset!



Consider the following model:
“Always predict 2”

Accuracy

```
The training set has 70000 samples
Accuracy on the training set is 1.0000
```

Predicted class

True class

	0	1	2	3	4	5	6	7	8	9
0	973	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	10	9	996	2	0	0	0	13	2	0
3	0	2	4	976	1	13	1	7	3	3
4	1	6	0	0	950	0	4	2	0	19
5	6	1	0	11	2	859	5	1	3	4
6	5	3	0	0	3	3	944	0	0	0
7	0	21	5	0	1	0	0	991	0	10
8	8	2	4	16	8	11	3	4	914	4
9	4	5	2	8	9	2	1	8	2	968

Confusion matrix

$$\text{acc} = \frac{\text{correctly classified}}{\text{total number of samples}}$$

Beware of an **imbalanced** dataset!





Consider the following model:
“Always predict 2”

Accuracy 0.9

Decision tree

```
from sklearn.datasets import fetch_mldata
from sklearn import tree

# load data
mnist = fetch_mldata('MNIST original')
print 'The training set has %d samples' % mnist.data.shape[0]

# train a decision tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(mnist.data, mnist.target)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
```

```
The training set has 70000 samples
Accuracy on the training set is 1.0000
```

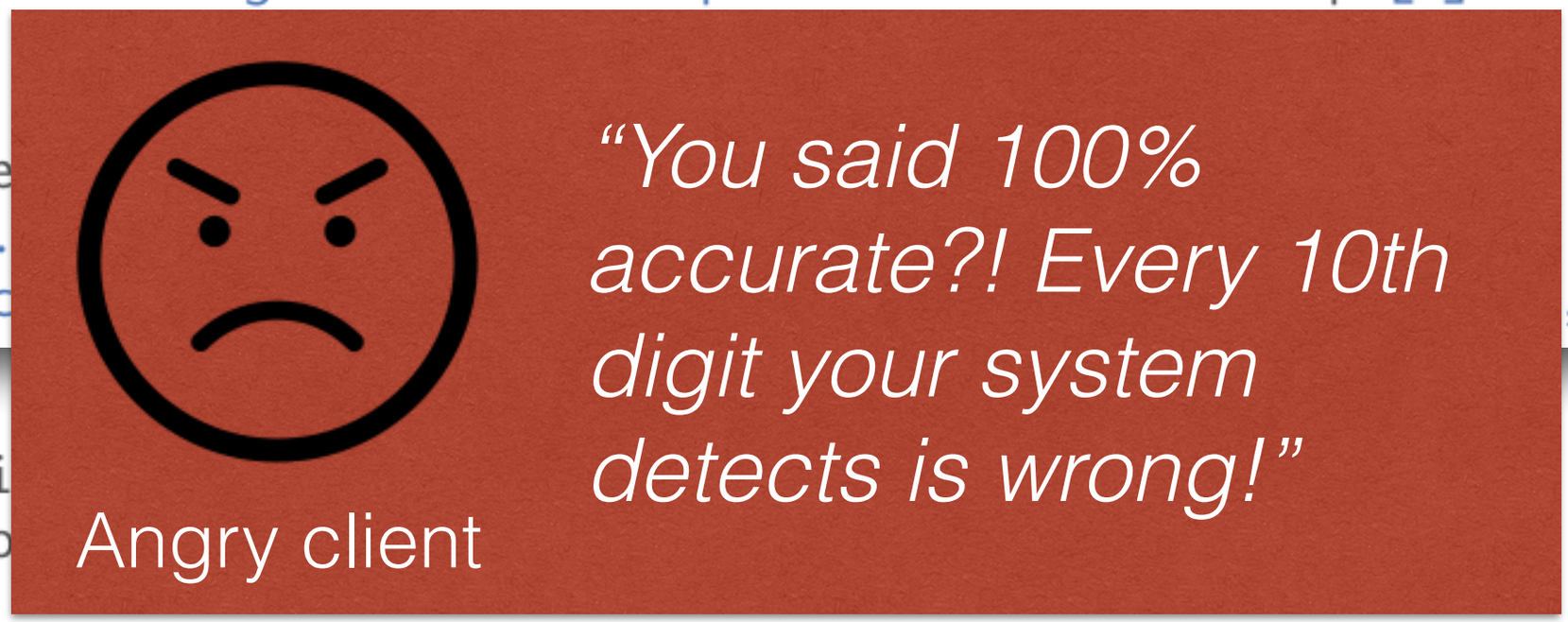
Decision tree

```
from sklearn.datasets import fetch_mldata
from sklearn import tree

# load data
mnist = fetch_mldata('MNIST original')
print 'The training set has %d samples' % mnist.data.shape[0]

# train a
clf = tree
clf = clf.
print 'Acc                                                    , y_train)
```

```
The traini
Accuracy o
```



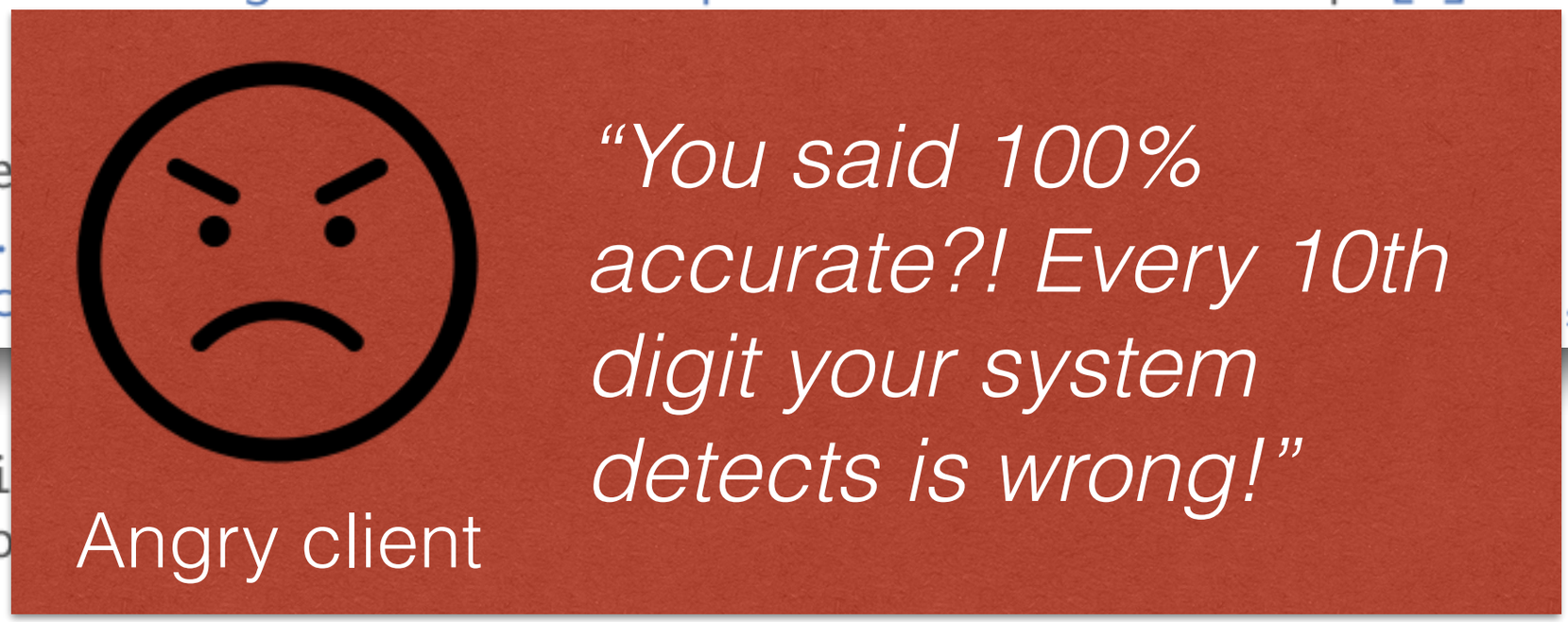
Decision tree

```
from sklearn.datasets import fetch_mldata
from sklearn import tree

# load data
mnist = fetch_mldata('MNIST original')
print 'The training set has %d samples' % mnist.data.shape[0]

# train a
clf = tree
clf = clf.
print 'Acc                                                    , y_train)
```

```
The traini
Accuracy o
```



We’ve trained our system on the data the client gave us. But our system has never seen the **new** data the client applied it to.
And in the real life — it never will…

OVERFITTING

Simulate the real-life situation — split the dataset

OVERFITTING

Simulate the real-life situation — split the dataset

```
from sklearn.datasets import fetch_mldata
from sklearn.cross_validation import train_test_split
from sklearn import tree

# load data
mnist = fetch_mldata('MNIST original')

# split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(mnist.data, mnist.target,
                                                    test_size=0.2, random_state=42)
print 'The training set has %d samples' % X_train.shape[0]
print 'The test set has %d samples' % X_test.shape[0]

# train a decision tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

OVERFITTING

Simulate the real-life situation — split the dataset

```
from sklearn.datasets import fetch_mldata
from sklearn.cross_validation import train_test_split
from sklearn import tree

# load data
mnist = fetch_mldata('MNIST original')

# split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(mnist.data, mnist.target,
                                                    test_size=0.2, random_state=42)
print 'The training set has %d samples' % X_train.shape[0]
print 'The test set has %d samples' % X_test.shape[0]

# train a decision tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

```
The training set has 56000 samples
The test set has 14000 samples
Accuracy on the training set is 1.0000
Accuracy on the test set is 0.8740
```

OVERFITTING

Simulate the real-life situation — split the dataset

```
from sklearn.datasets import fetch_mldata
from sklearn.cross_validation import train_test_split
from sklearn import tree

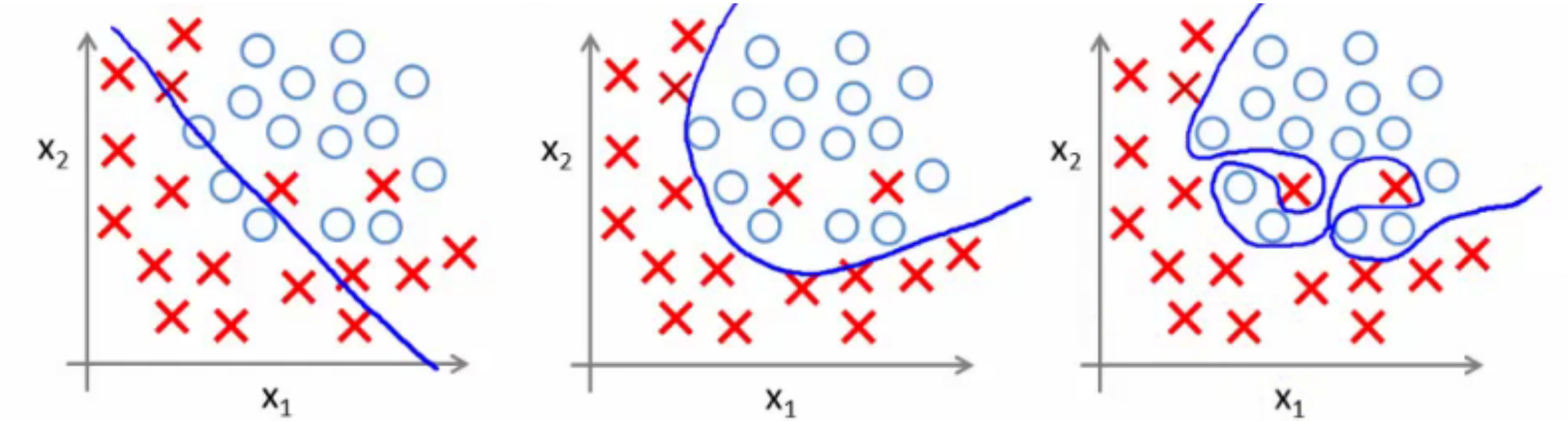
# load data
mnist = fetch_mldata('MNIST original')

# split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(mnist.data, mnist.target,
                                                    test_size=0.2, random_state=42)
print 'The training set has %d samples' % X_train.shape[0]
print 'The test set has %d samples' % X_test.shape[0]

# train a decision tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

```
The training set has 56000 samples
The test set has 14000 samples
Accuracy on the training set is 1.0000
Accuracy on the test set is 0.8740
```

OVERFITTING

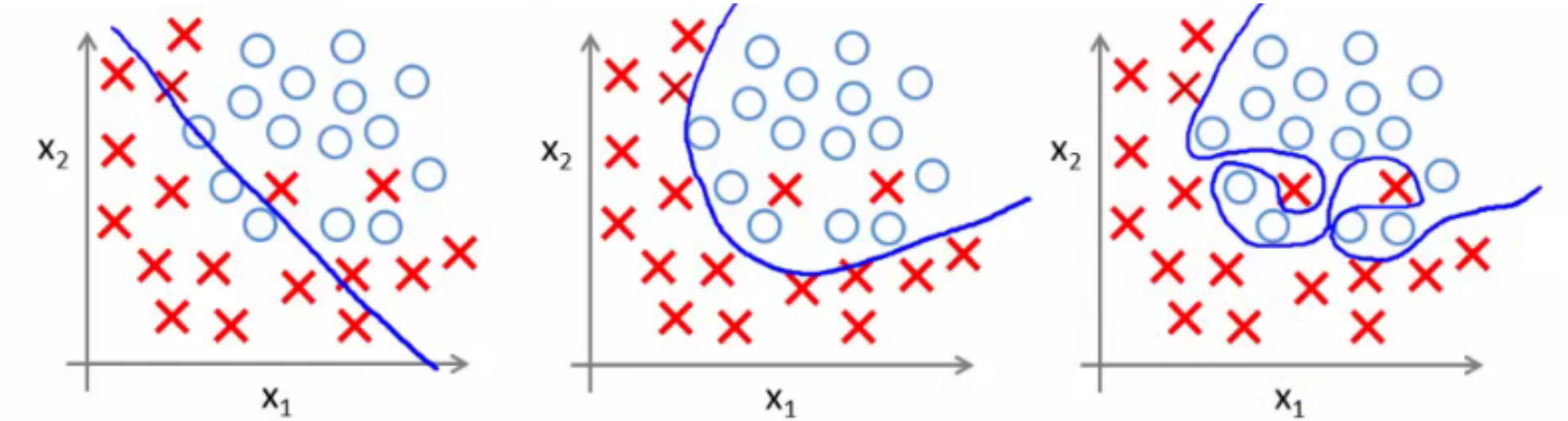


Underfitting
"Too stupid"

OK

Overfitting
"Too smart"

OVERFITTING



Underfitting
"Too stupid"

OK

Overfitting
"Too smart"

Our current decision tree has too much capacity,
it just has memorized all of the data.
Let's make it less complex.

```
clf = tree.DecisionTreeClassifier(min_samples_leaf=100)
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

```
clf = tree.DecisionTreeClassifier(min_samples_leaf=100)
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

```
Accuracy on the training set is 0.8139
Accuracy on the test set is 0.8084
```

```
clf = tree.DecisionTreeClassifier(min_samples_leaf=100)
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

```
Accuracy on the training set is 0.8139
Accuracy on the test set is 0.8084
```

```
clf = tree.DecisionTreeClassifier(min_samples_leaf=10)
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

```
Accuracy on the training set is 0.9197
Accuracy on the test set is 0.8704
```

```
clf = tree.DecisionTreeClassifier(min_samples_leaf=100)
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

```
Accuracy on the training set is 0.8139
Accuracy on the test set is 0.8084
```

```
clf = tree.DecisionTreeClassifier(min_samples_leaf=10)
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

```
Accuracy on the training set is 0.9197
Accuracy on the test set is 0.8704
```

You probably did not notice, but we are overfitting again :(

The whole dataset

Training set 60%	Validation set 20%	Test set 20%

The whole dataset

Training set 60%	Validation set 20%	Test set 20%

Fit various models and parameter combinations on this subset

The whole dataset

Training set 60%	Validation set 20%	Test set 20%

Fit various models and parameter combinations on this subset

- **Evaluate** the models created with different parameters

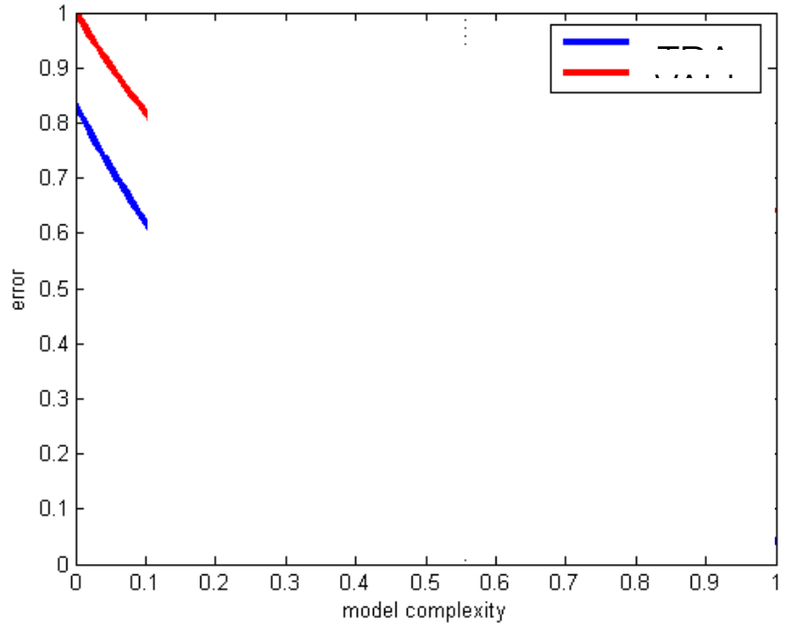
The whole dataset

Training set 60%	Validation set 20%	Test set 20%

Fit various models and parameter combinations on this subset

- **Evaluate** the models created with different parameters

- Estimate **overfitting**



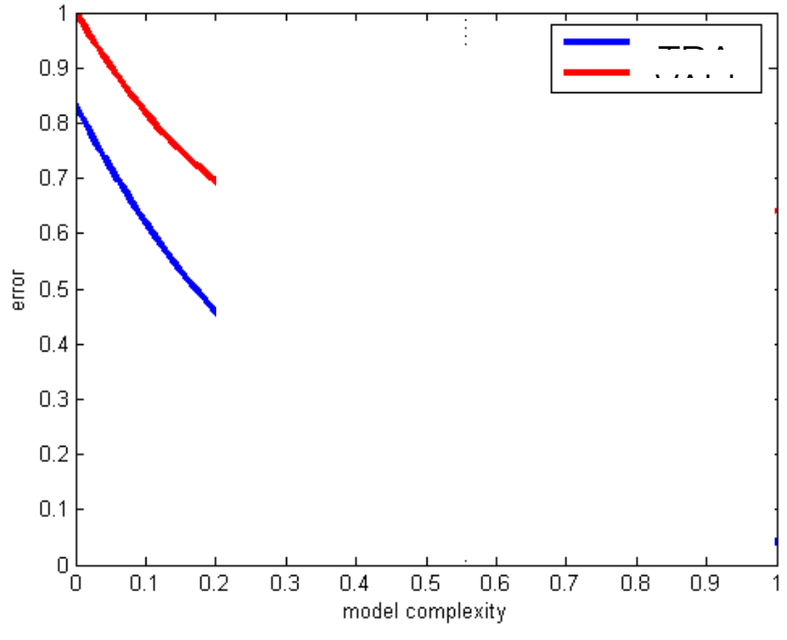
The whole dataset

Training set 60%	Validation set 20%	Test set 20%

Fit various models and parameter combinations on this subset

- **Evaluate** the models created with different parameters

- Estimate **overfitting**



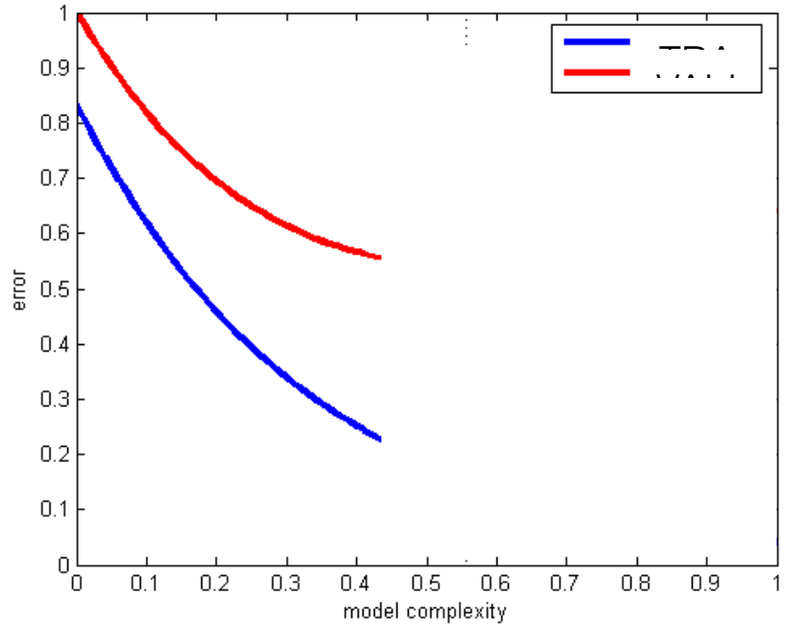
The whole dataset

Training set 60%	Validation set 20%	Test set 20%

Fit various models and parameter combinations on this subset

- **Evaluate** the models created with different parameters

- Estimate **overfitting**



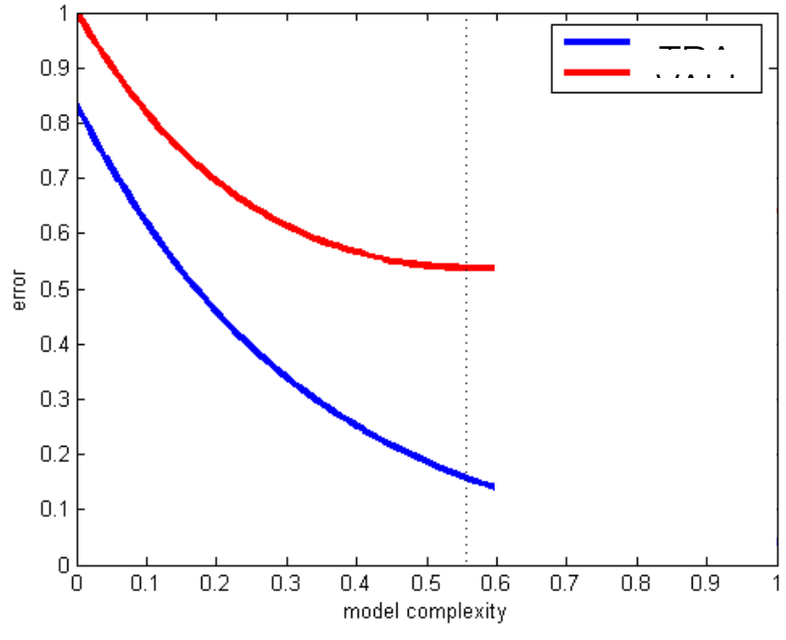
The whole dataset

Training set 60%	Validation set 20%	Test set 20%

Fit various models and parameter combinations on this subset

- **Evaluate** the models created with different parameters

- Estimate **overfitting**



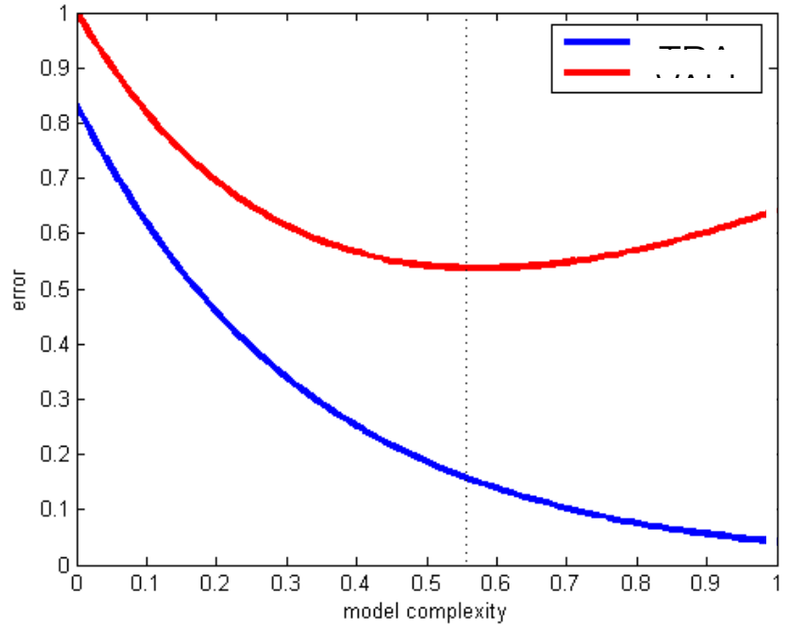
The whole dataset

Training set 60%	Validation set 20%	Test set 20%

Fit various models and parameter combinations on this subset

- **Evaluate** the models created with different parameters

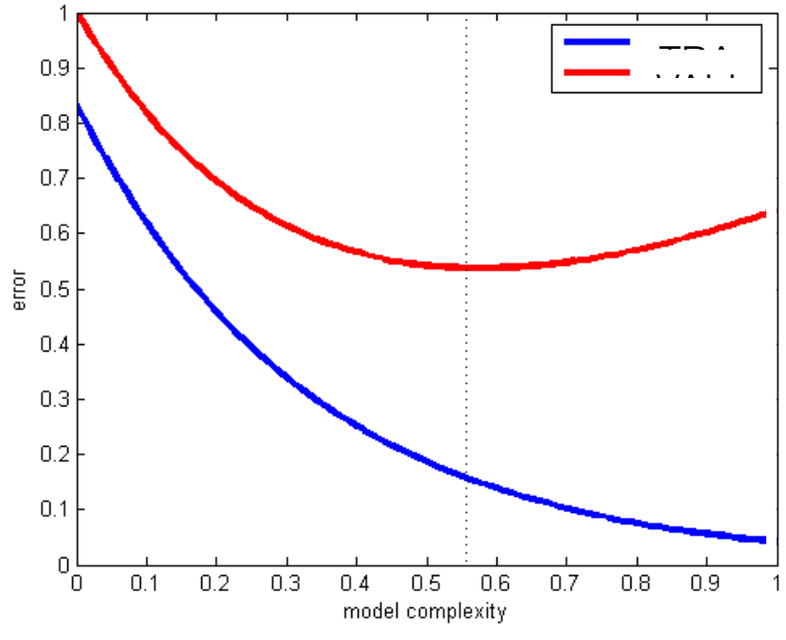
- Estimate **overfitting**



The whole dataset

Training set 60%	Validation set 20%	Test set 20%
Fit various models and parameter combinations on this subset	• **Evaluate** the models created with different parameters	Use only **once** to get the final performance estimate

- Estimate **overfitting**



Training set 60%	Validation set 20%	Test set 20%

```
from sklearn.datasets import fetch_mldata
from sklearn.cross_validation import train_test_split
from sklearn import tree

# load data
mnist = fetch_mldata('MNIST original')

# split into training, validation and test set
X_trainall, X_test, y_trainall, y_test = train_test_split(mnist.data, mnist.target,
                                                          test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_trainall, y_trainall,
                                                  test_size=0.2, random_state=42)
print 'The training set has %d samples' % X_train.shape[0]
print 'The validation set has %d samples' % X_val.shape[0]
print 'The test set has %d samples' % X_test.shape[0]

# go over parameters to pick the best
best_score = 0.0
best_msl = None
for msl in [5, 10, 15]:
    clf = tree.DecisionTreeClassifier(min_samples_leaf=msl)
    clf = clf.fit(X_train, y_train)
    validation_score = clf.score(X_val, y_val)
    if validation_score > best_score:
        best_msl = msl
        best_score = validation_score
    print 'With msl=%d accuracy on the validation set is %.4f' % (msl, validation_score)

# train the final decition tree
print 'Best msl is %d' % best_msl
clf = tree.DecisionTreeClassifier(min_samples_leaf=best_msl)
clf = clf.fit(X_trainall, y_trainall)
print 'Accuracy on the training set is %.4f' % clf.score(X_trainall, y_trainall)

# estimate the model performance on the holdout set
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

Training set 60%	Validation set 20%	Test set 20%

```
from sklearn.datasets import fetch_mldata
from sklearn.cross_validation import train_test_split
from sklearn import tree

# load data
mnist = fetch_mldata('MNIST original')

# split into training, validation and test set
X_trainall, X_test, y_trainall, y_test = train_test_split(mnist.data, mnist.target,
                                                          test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_trainall, y_trainall,
                                                  test_size=0.2, random_state=42)
print 'The training set has %d samples' % X_train.shape[0]
print 'The validation set has %d samples' % X_val.shape[0]
print 'The test set has %d samples' % X_test.shape[0]

# go over parameters to pick the best
best_score = 0.0
best_msl = None
for msl in [5, 10, 15]:
    clf = tree.DecisionTreeClassifier(min_samples_leaf=msl)
    clf = clf.fit(X_train, y_train)
    validation_score = clf.score(X_val, y_val)
    if validation_score > best_score:
        best_msl = msl
        best_score = validation_score
    print 'With msl=%d accuracy on the valida

# train the final decition tree
print 'Best msl is %d' % best_msl
clf = tree.DecisionTreeClassifier(min_samples
clf = clf.fit(X_trainall, y_trainall)
print 'Accuracy on the training set is %.4f'

# estimate the model performance on the holdo
print 'Accuracy on the test set is %.4f' % cl
```

```
The training set has 44800 samples
The validation set has 11200 samples
The test set has 14000 samples
With msl=5 accuracy on the validation set is 0.8648
With msl=10 accuracy on the validation set is 0.8562
With msl=15 accuracy on the validation set is 0.8498
Best msl is 5
Accuracy on the training set is 0.9460
Accuracy on the test set is 0.8766
```

Cross-validation

The whole dataset

Training set 60%

Validation set 20%

Cross-validation

The whole dataset

Training set 60%

Validation set 20%

What if we got too
optimistic validation set?

Cross-validation

The whole dataset

Training set 60%

Validation set 20%

What if we got too
optimistic validation set?

Training set 80%

Cross-validation

The whole dataset

Training set 60%

Validation set 20%

What if we got too **optimistic** validation set?

Training set 80%

Fix the parameter value you ned to evaluate, say `msl=15`

Cross-validation

The whole dataset

Training set 60%

Validation set 20%

What if we got too **optimistic** validation set?

Training set 80%

Fix the parameter value you ned to evaluate, say `msl=15`



Repeat 10 times



Cross-validation

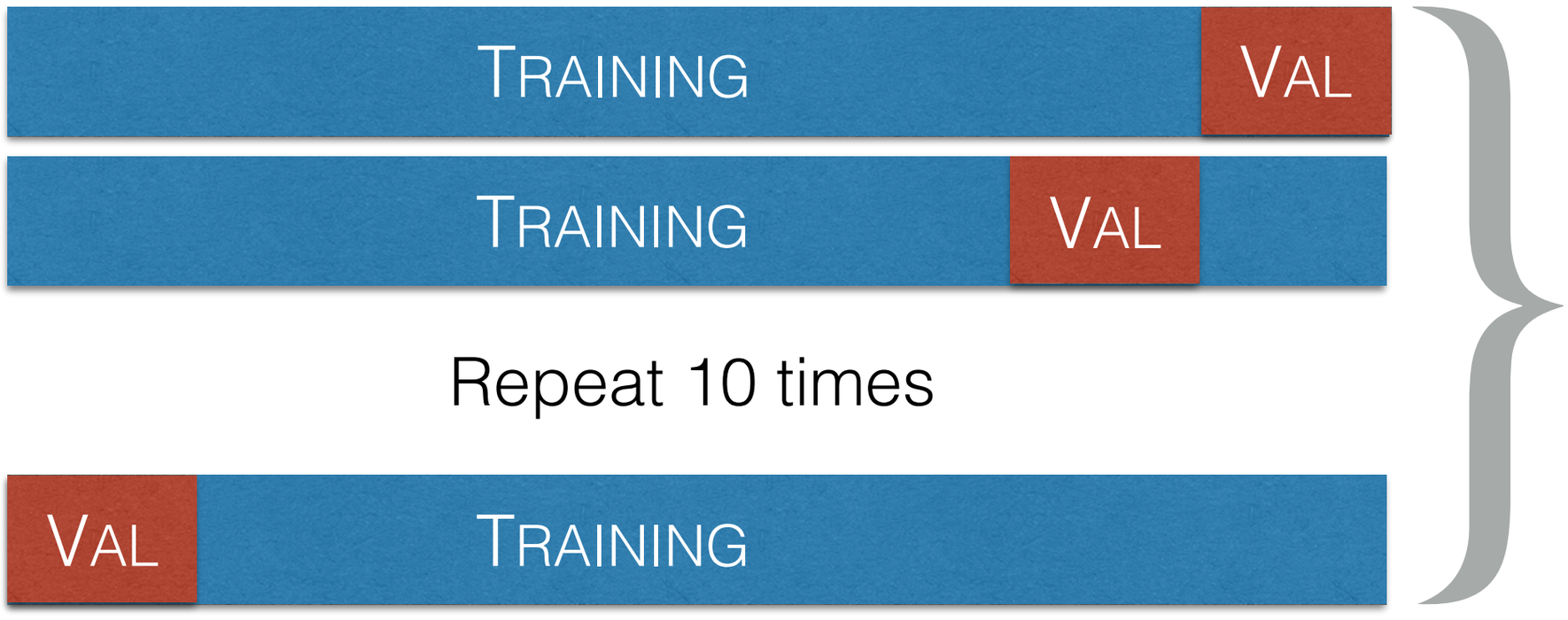
The whole dataset

Training set 60% | Validation set 20%

What if we got too **optimistic** validation set?

Training set 80%

Fix the parameter value you ned to evaluate, say `msl=15`



Take average validation score over 10 runs — it is a more **stable** estimate.

```
# go over parameters to pick the best
best_score = 0.0
best_msl = None
for msl in [5, 10, 15]:
    clf = tree.DecisionTreeClassifier(min_samples_leaf=msl)
    clf = clf.fit(X_train, y_train)
    validation_score = clf.score(X_val, y_val)
    if validation_score > best_score:
        best_msl = msl
        best_score = validation_score
    print 'With msl=%d accuracy on the validation set is %.4f' % (msl, validation_score)
```

```
# go over parameters to pick the best
best_score = 0.0
best_msl = None
for msl in [5, 10, 15]:
    clf = tree.DecisionTreeClassifier(min_samples_leaf=msl)
    clf = clf.fit(X_train, y_train)
    validation_score = clf.score(X_val, y_val)
    if validation_score > best_score:
        best_msl = msl
        best_score = validation_score
    print 'With msl=%d accuracy on the validation set is %.4f' % (msl, validation_score)
```

```
# go over parameters to pick the best
best_score = 0.0
best_msl = None
for msl in [5, 10, 15]:
    clf = tree.DecisionTreeClassifier(min_samples_leaf=msl)
    scores = cross_val_score(clf, X_train, y_train, cv=5)
    if scores.mean() > best_score:
        best_msl = msl
        best_score = scores.mean()
    print 'With msl=%d accuracy on the validation set is %0.4f (+/- %0.4f)' % \
          (msl, scores.mean(), scores.std() * 2)
```

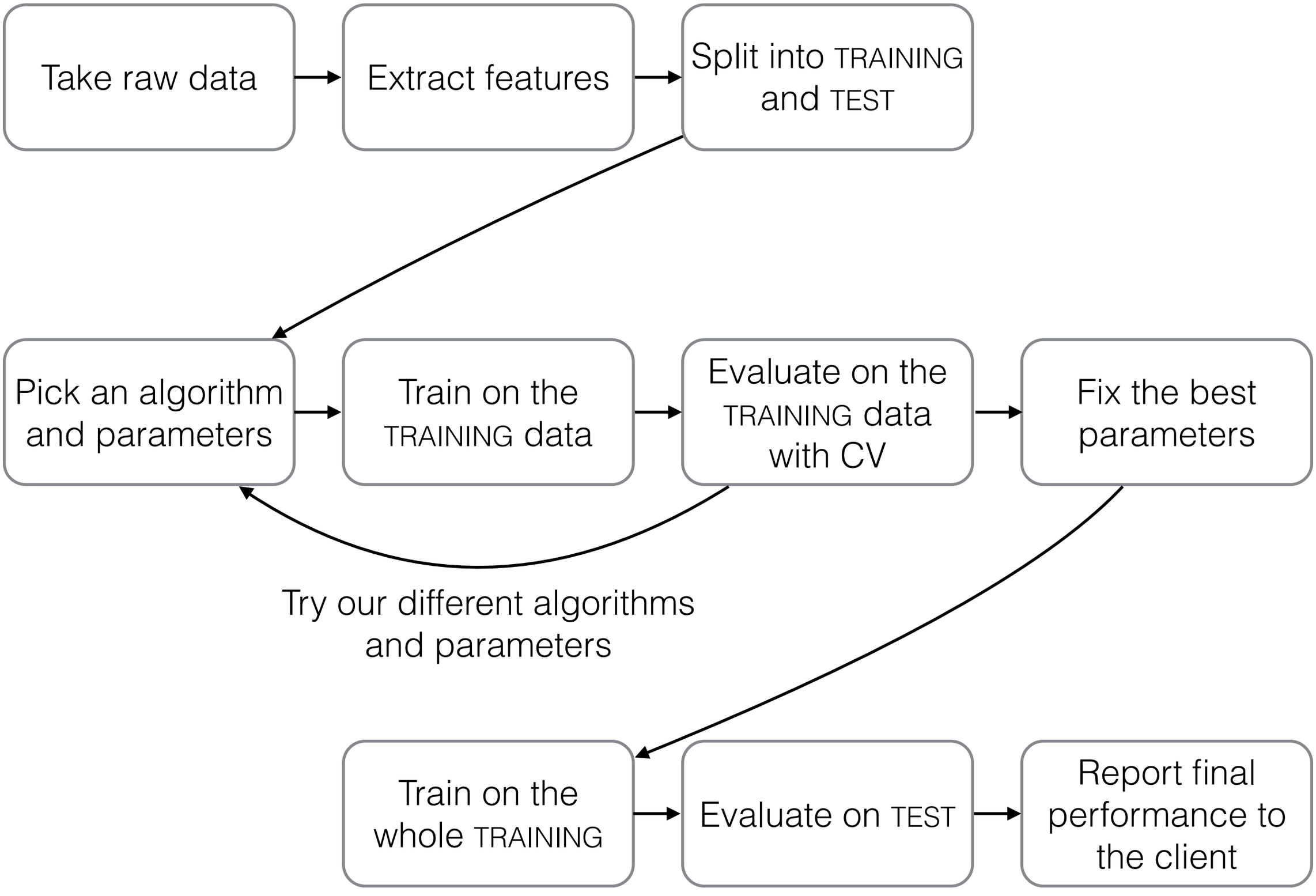
```
# go over parameters to pick the best
best_score = 0.0
best_msl = None
for msl in [5, 10, 15]:
    clf = tree.DecisionTreeClassifier(min_samples_leaf=msl)
    clf = clf.fit(X_train, y_train)
    validation_score = clf.score(X_val, y_val)
    if validation_score > best_score:
        best_msl = msl
        best_score = validation_score
    print 'With msl=%d accuracy on the validation set is %.4f' % (msl, validation_score)
```

```
With msl=5 accuracy on the validation set is 0.8648
With msl=10 accuracy on the validation set is 0.8562
With msl=15 accuracy on the validation set is 0.8498
```

```
# go over parameters to pick the best
best_score = 0.0
best_msl = None
for msl in [5, 10, 15]:
    clf = tree.DecisionTreeClassifier(min_samples_leaf=msl)
    scores = cross_val_score(clf, X_train, y_train, cv=5)
    if scores.mean() > best_score:
        best_msl = msl
        best_score = scores.mean()
    print 'With msl=%d accuracy on the validation set is %0.4f (+/- %0.4f)' % \
          (msl, scores.mean(), scores.std() * 2)
```

```
With msl=5 accuracy on the validation set is 0.8577 (+/- 0.0081)
With msl=10 accuracy on the validation set is 0.8525 (+/- 0.0060)
With msl=15 accuracy on the validation set is 0.8486 (+/- 0.0042)
```

Machine Learning Pipeline



Machine Learning Pipeline

Take raw data → Extract features → Split into TRAINING and TEST



"So it is ~87%…erm… Could you do better?"

Pick an algo… and parame… → … → …with CV → …x the best …arameters

Try our different algorithms and parameters

Train on the whole TRAINING → Evaluate on TEST → Report final performance to the client

Machine Learning Pipeline

Take raw data → Extract features → Split into TRAINING and TEST



"So it is ~87%…erm… Could you do better?"

Yes

Pick an algo… and parame… … the best … parameters

Train on the whole TRAINING → Evaluate on TEST → Report final performance to the client

Pick another algorithm

- C4.5
- Random forests
- Bayesian networks
- Hidden Markov models
- Artificial neural network
- Data clustering
- Expectation-maximization algorithm
- Self-organizing map
- Radial basis function network
- Vector Quantization
- Generative topographic map
- Information bottleneck method
- IBSEAD
- Apriori algorithm
- Eclat algorithm
- FP-growth algorithm
- Single-linkage clustering
- Conceptual clustering
- K-means algorithm
- Fuzzy clustering
- Temporal difference learning
- Q-learning
- Learning Automata
- AODE
- Artificial neural network
- Backpropagation
- Naive Bayes classifier
- Bayesian network
- Bayesian knowledge base
- Case-based reasoning
- Decision trees
- Inductive logic programming
- Gaussian process regression
- Gene expression programming
- Group method of data handling (GMDH)
- Learning Automata
- Learning Vector Quantization
- Logistic Model Tree
- Decision tree
- Decision graphs
- Lazy learning
- Monte Carlo Method
- SARSA
- Instance-based learning
- Nearest Neighbor Algorithm
- Analogical modeling
- Probably approximately correct learning (PACL)
- Symbolic machine learning algorithms
- Subsymbolic machine learning algorithms
- Support vector machines
- Random Forest
- Ensembles of classifiers
- Bootstrap aggregating (bagging)
- Boosting (meta-algorithm)
- Ordinal classification
- Regression analysis
- Information fuzzy networks (IFN)
- Linear classifiers
- Fisher's linear discriminant
- Logistic regression
- Naive Bayes classifier
- Perceptron
- Support vector machines
- Quadratic classifiers
- k-nearest neighbor
- Boosting

Pick another algorithm

- C4.5
- Random forests
- Bayesian networks
- Hidden Markov models
- Artificial neural network
- Data clustering
- Expectation-maximization algorithm
- Self-organizing map
- Radial basis function network
- Vector Quantization
- Generative topographic map
- Information bottleneck method
- IBSEAD
- Apriori algorithm
- Eclat algorithm
- FP-growth algorithm
- Single-linkage clustering
- Conceptual clustering
- K-means algorithm
- Fuzzy clustering
- Temporal difference learning
- Q-learning
- Learning Automata
- AODE
- Artificial neural network
- Backpropagation
- Naive Bayes classifier
- Bayesian network
- Bayesian knowledge base
- Case-based reasoning
- Decision trees
- Inductive logic programming
- Gaussian process regression
- Gene expression programming
- Group method of data handling (GMDH)
- Learning Automata
- Learning Vector Quantization
- Logistic Model Tree
- Decision tree
- Decision graphs
- Lazy learning
- Monte Carlo Method
- SARSA
- Instance-based learning
- Nearest Neighbor Algorithm
- Analogical modeling
- Probably approximately correct learning (PACL)
- Symbolic machine learning algorithms
- Subsymbolic machine learning algorithms
- Support vector machines
- Random Forest
- Ensembles of classifiers
- Bootstrap aggregating (bagging)
- Boosting (meta-algorithm)
- Ordinal classification
- Regression analysis
- Information fuzzy networks (IFN)
- Linear classifiers
- Fisher's linear discriminant
- Logistic regression
- Naive Bayes classifier
- Perceptron
- Support vector machines
- Quadratic classifiers
- k-nearest neighbor
- Boosting

Random Forest

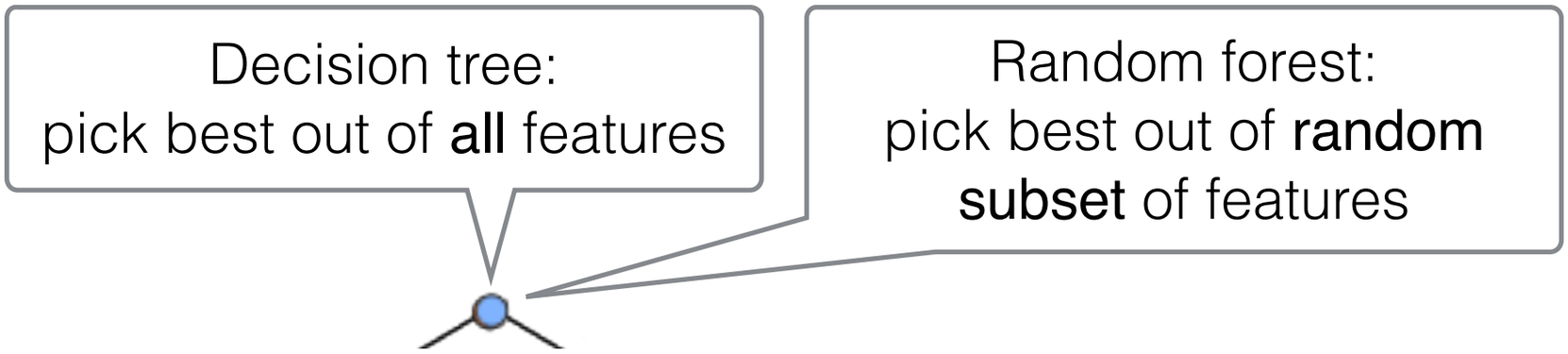


Random Forest

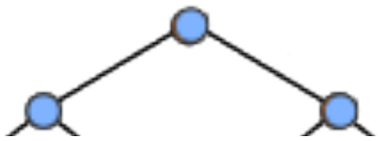
Decision tree:
pick best out of **all** features



Random Forest

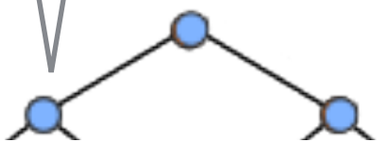


Random Forest



Random Forest

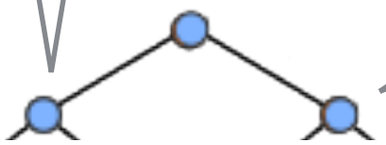
pick best out of **another** random subset of features



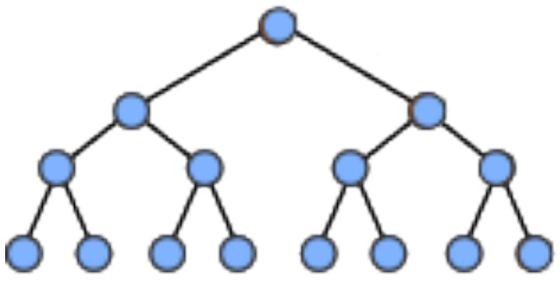
Random Forest

pick best out of **another** random subset of features

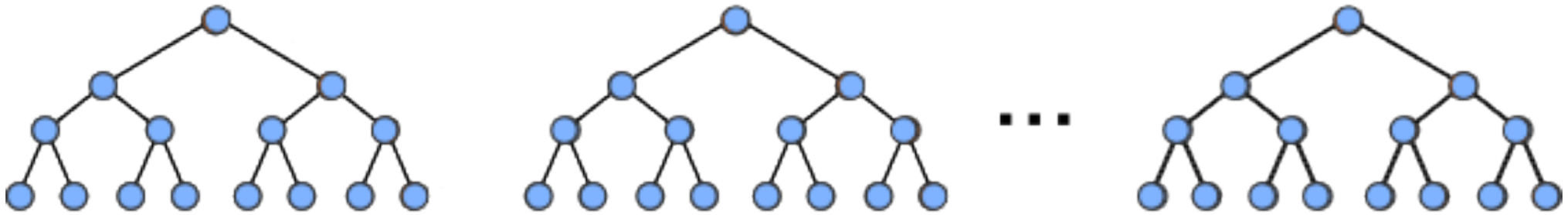
pick best out of **yet another** random subset of features



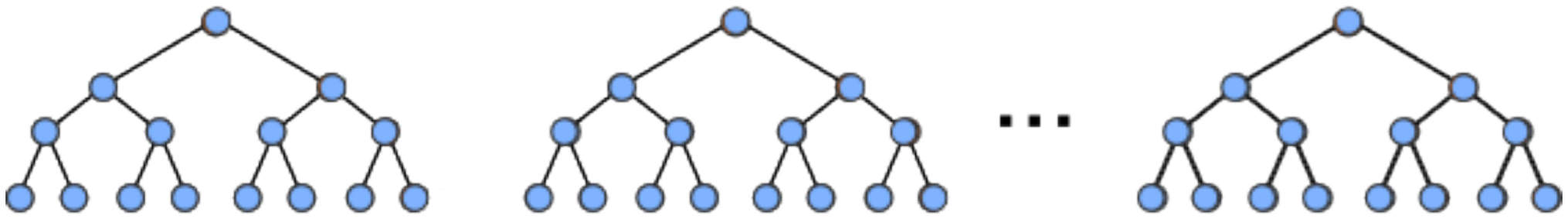
Random Forest



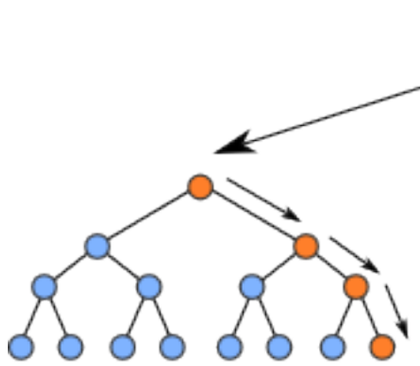
Random Forest



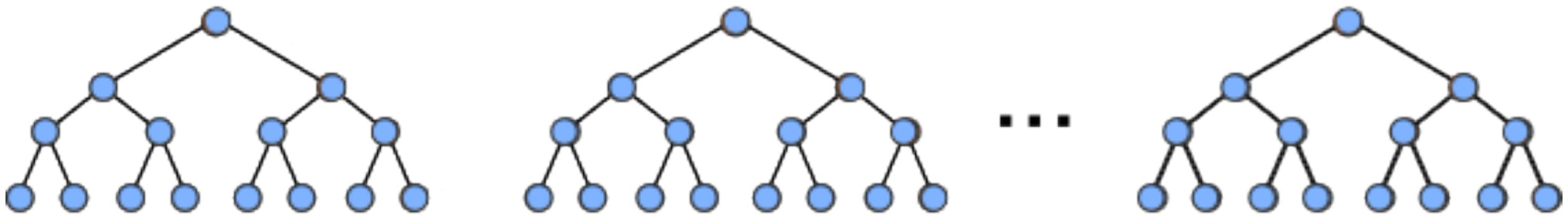
RANDOM FOREST



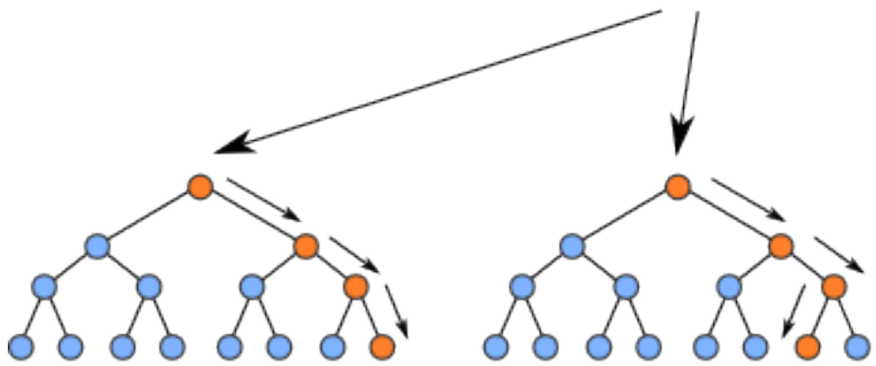
instance



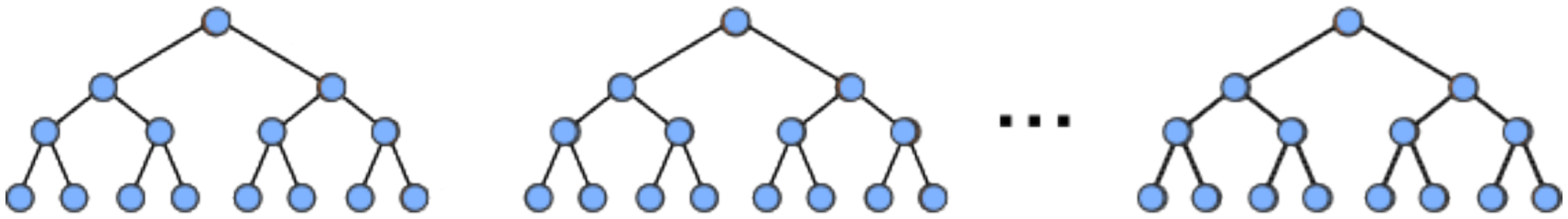
Random Forest



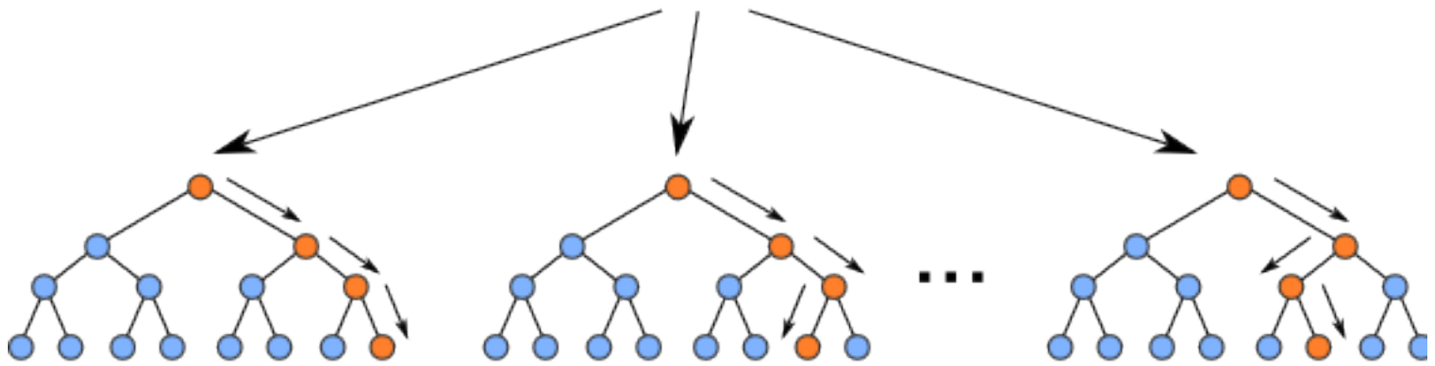
instance



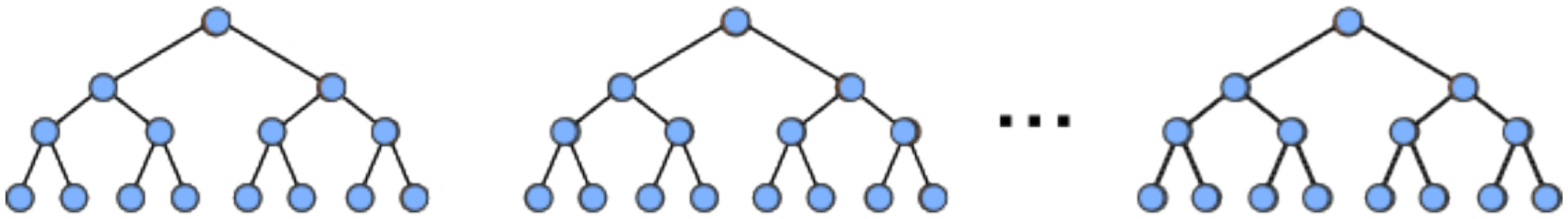
Random Forest

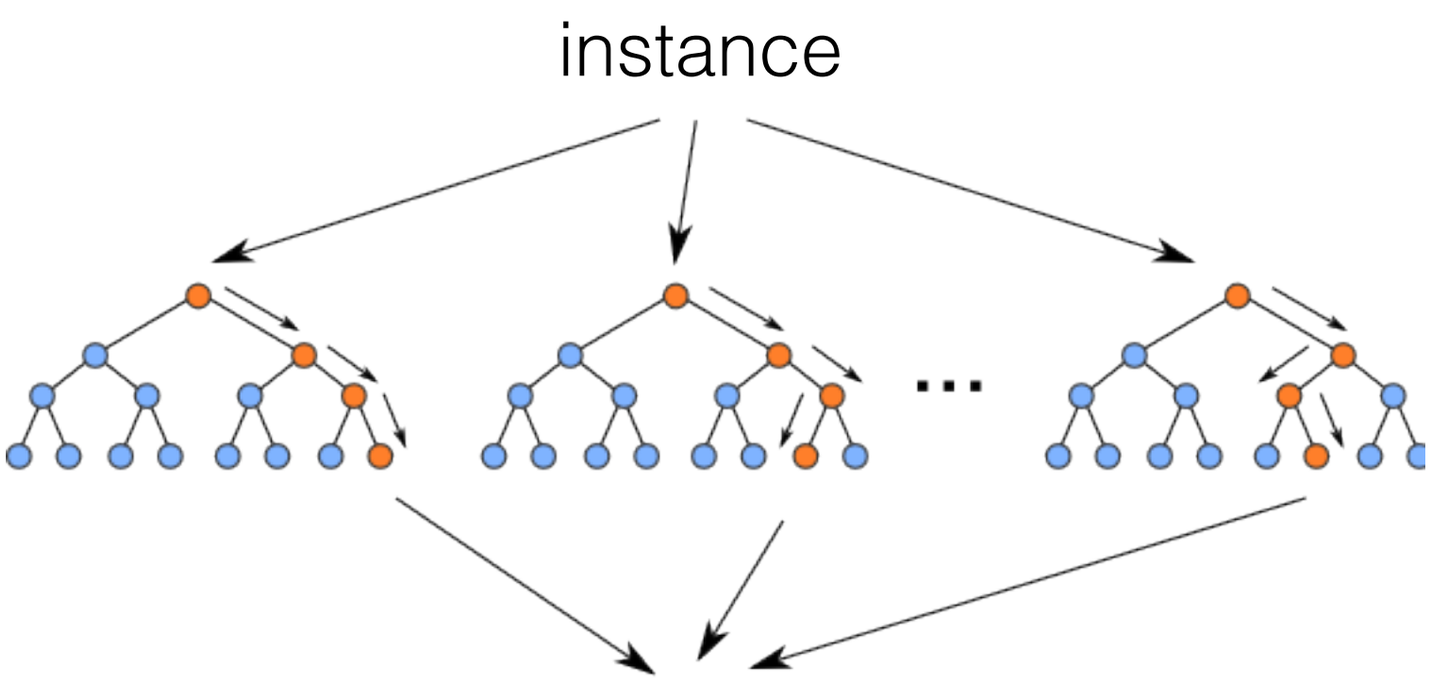


instance



RANDOM FOREST





```
from sklearn.datasets import fetch_mldata
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier

# load data
mnist = fetch_mldata('MNIST original')

# split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(mnist.data, mnist.target,
                                                    test_size=0.2, random_state=42)
print 'The training set has %d samples' % X_train.shape[0]
print 'The test set has %d samples' % X_test.shape[0]

# train a random forest
clf = RandomForestClassifier(n_estimators=100)
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

```
from sklearn.datasets import fetch_mldata
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier

# load data
mnist = fetch_mldata('MNIST original')

# split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(mnist.data, mnist.target,
                                                    test_size=0.2, random_state=42)
print 'The training set has %d samples' % X_train.shape[0]
print 'The test set has %d samples' % X_test.shape[0]

# train a random forest
clf = RandomForestClassifier(n_estimators=100)
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

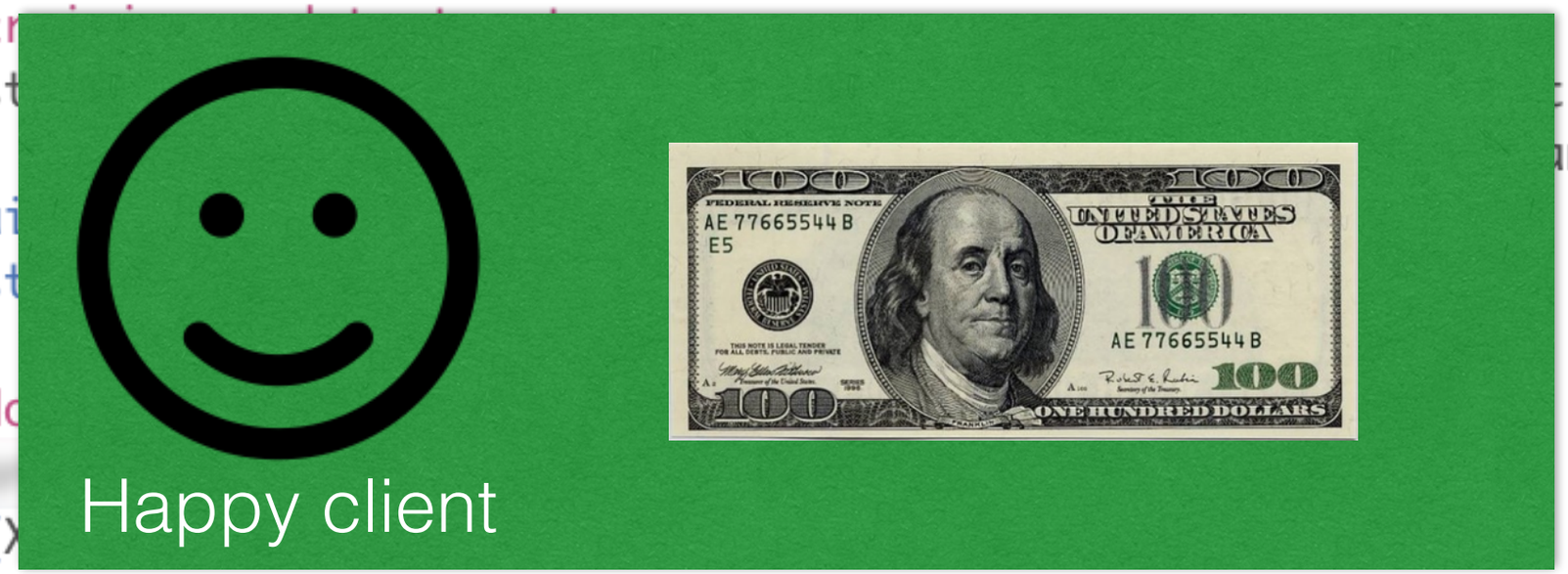
```
The training set has 56000 samples
The test set has 14000 samples
Accuracy on the training set is 1.0000
Accuracy on the test set is 0.9690
```

```
from sklearn.datasets import fetch_mldata
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier

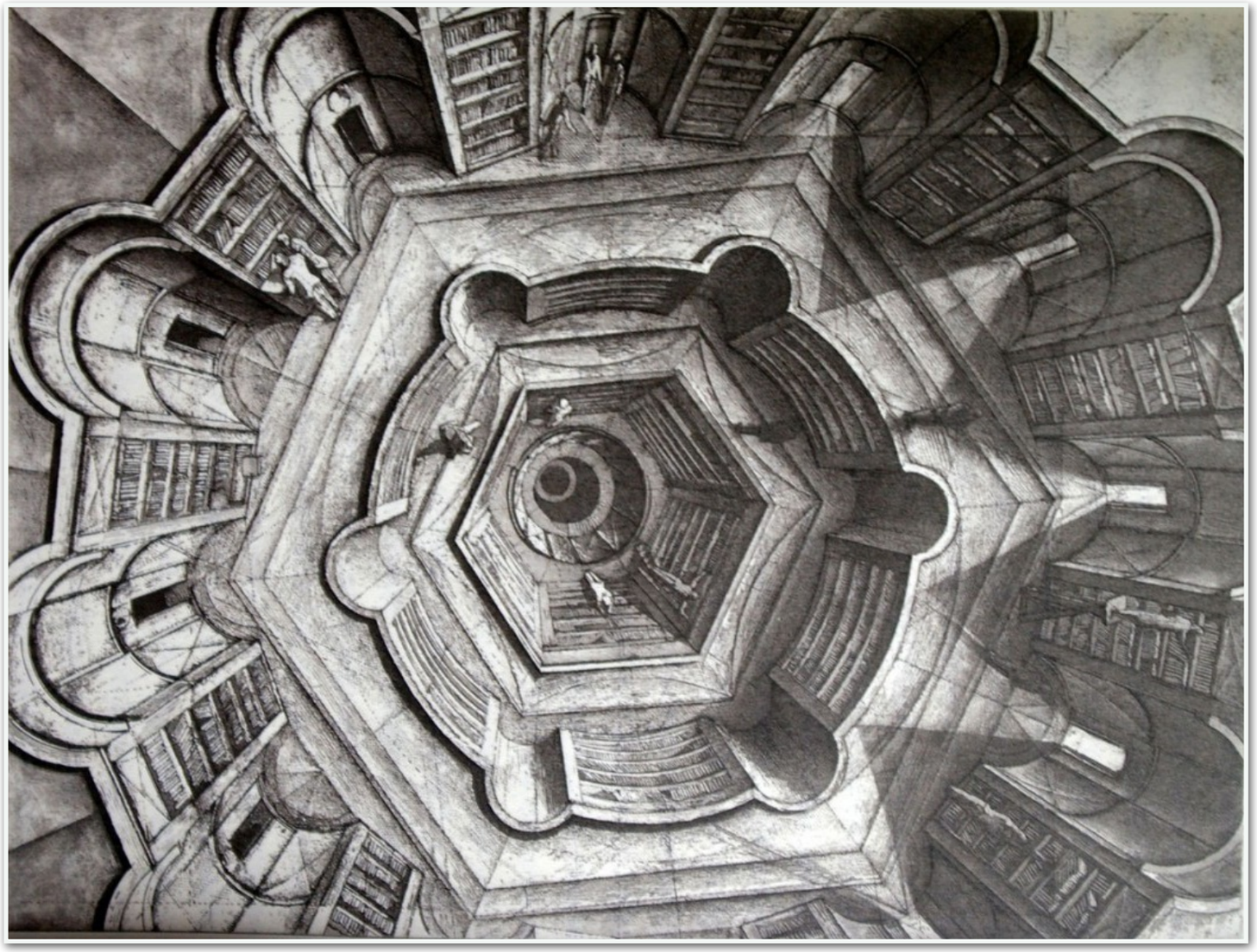
# load data
mnist = fetch_mldata('MNIST original')

# split into tr
X_train, X_test                                          t.target,
                                                         andom_state=42)
print 'The trai
print 'The test

# train a rando
clf =
clf = clf.fit(X
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

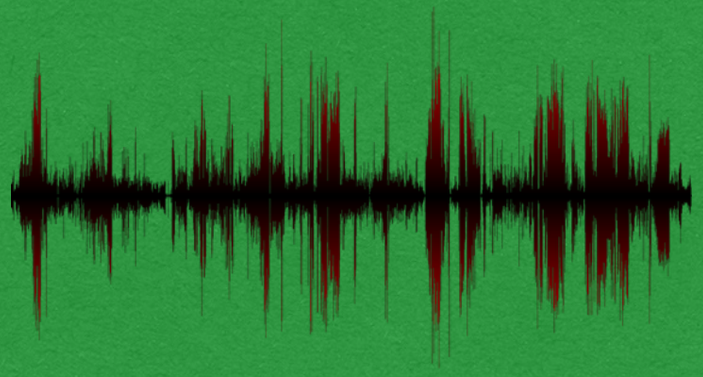



```
The training set has 56000 samples
The test set has 14000 samples
Accuracy on the training set is 1.0000
Accuracy on the test set is 0.9690
```



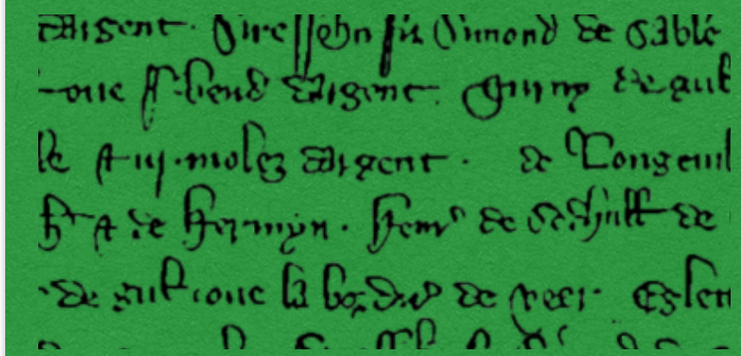
All other use cases

Sound



Frequency components → Genre

Text



Bag of words → Topic

Image



Pixel values → Cat or dog

Video



Frame pixels → Walking or running

Database records

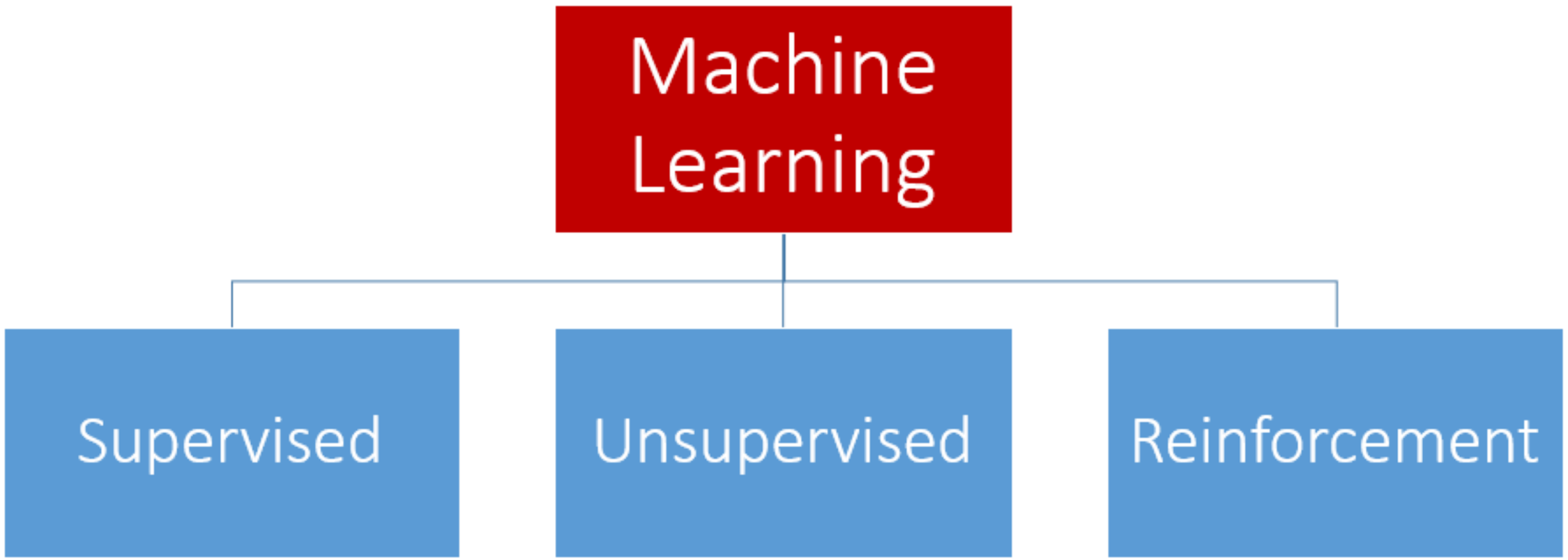
First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

Census data → Average salary

Biometric data



… → Dead or alive


Machine Learning
Supervised
Unsupervised
Reinforcement
Task driven (Regression / Classification)
Data driven (Clustering)
Algorithm learns to react to an environment

Machine Learning

Supervised

Task driven
(Regression / Classification)



Unsupervised

Data driven
(Clustering)

Reinforcement

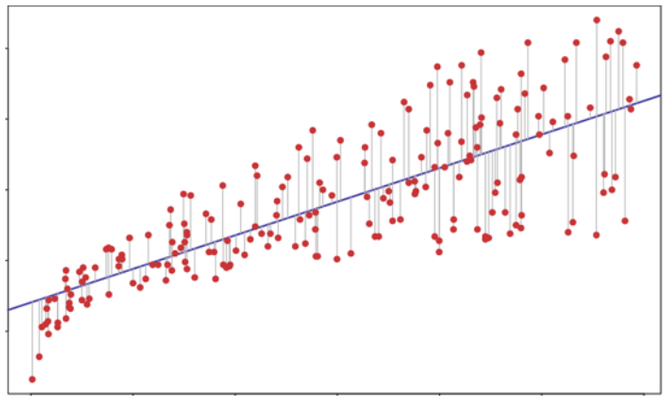
Algorithm learns to react to an environment

Machine Learning

Supervised

Task driven
(Regression / Classification)





Unsupervised

Data driven
(Clustering)

Reinforcement

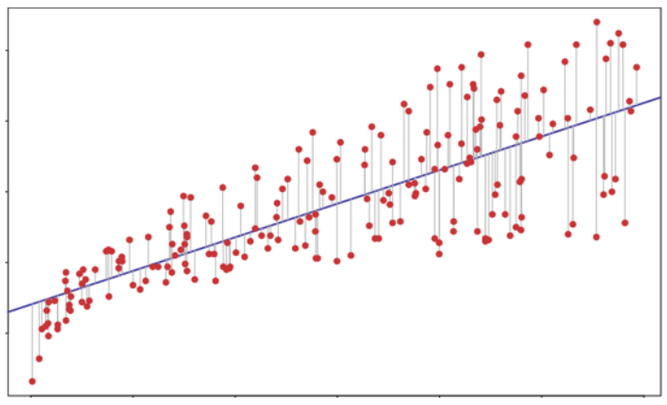
Algorithm learns to react to an environment

Machine Learning

Supervised

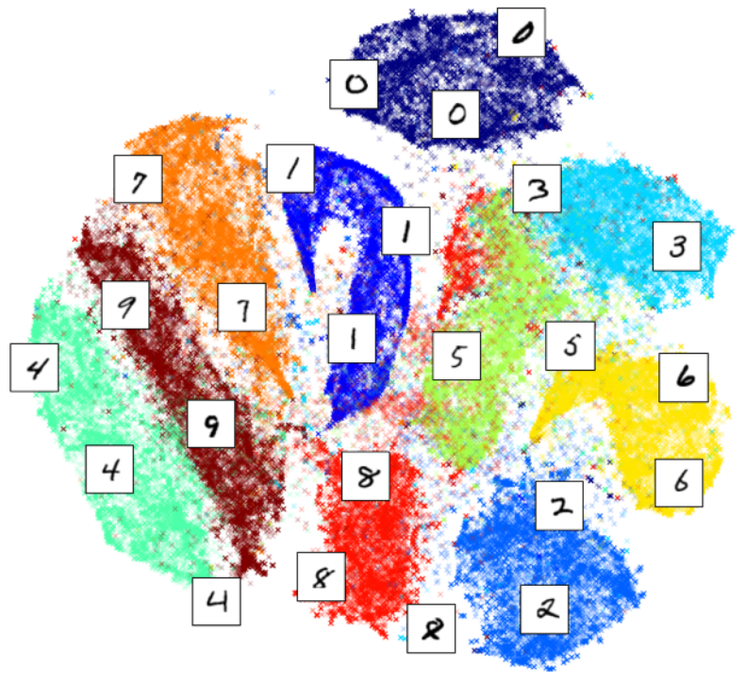
Task driven
(Regression / Classification)





Unsupervised

Data driven
(Clustering)



Reinforcement

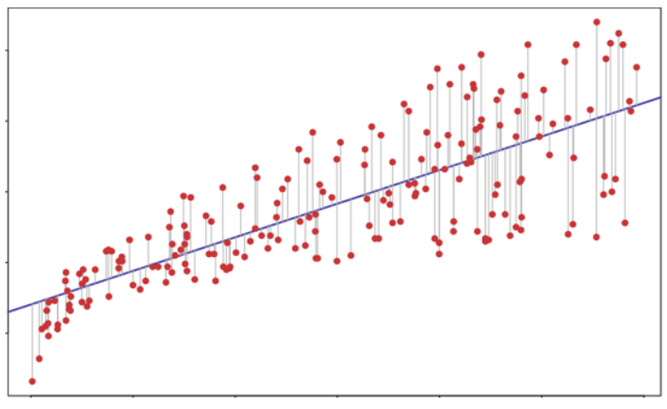
Algorithm learns to react to an environment

Machine Learning

Supervised

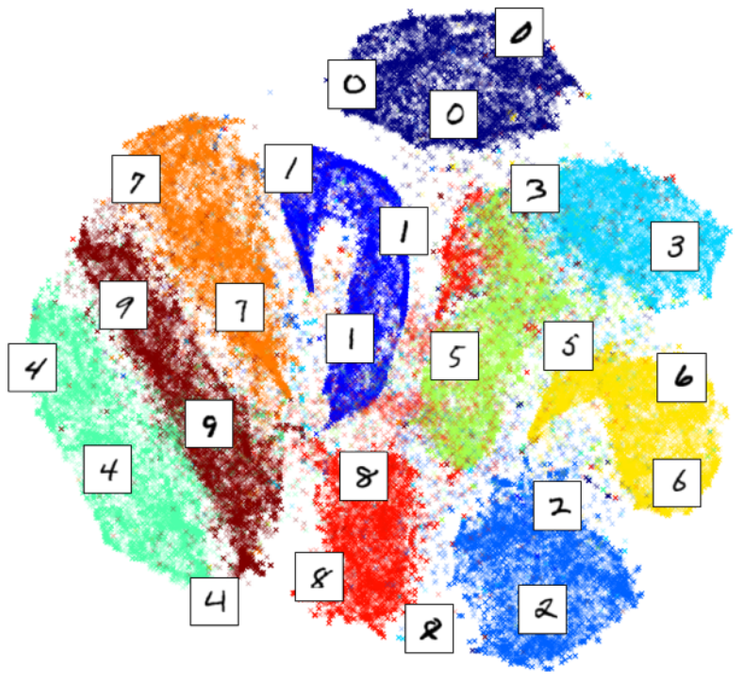
Task driven
(Regression / Classification)





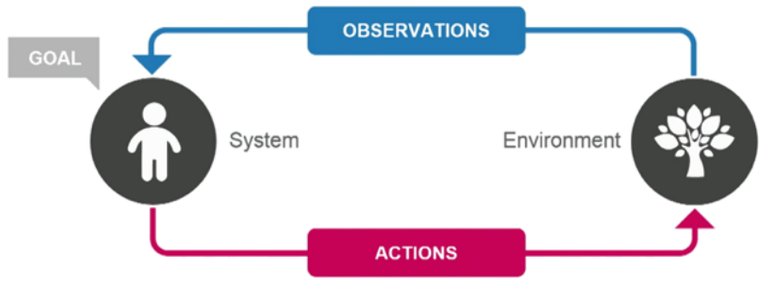
Unsupervised

Data driven
(Clustering)



Reinforcement

Algorithm learns to react to an environment





HANDS-ON SESSION

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

```
from sklearn.datasets import fetch_mldata
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier

# load data
mnist = fetch_mldata('MNIST original')

# split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(mnist.data, mnist.target,
                                                    test_size=0.2, random_state=42)
print 'The training set has %d samples' % X_train.shape[0]
print 'The test set has %d samples' % X_test.shape[0]

# train a random forest
clf = RandomForestClassifier(n_estimators=100)
clf = clf.fit(X_train, y_train)
print 'Accuracy on the training set is %.4f' % clf.score(X_train, y_train)
print 'Accuracy on the test set is %.4f' % clf.score(X_test, y_test)
```

```
The training set has 56000 samples
The test set has 14000 samples
Accuracy on the training set is 1.0000
Accuracy on the test set is 0.9690
```

Welcome to Kaggle's data science competitions.

New to Data Science?
Tutorials on the Titanic competition »

Want to learn from other's code?
Kaggler's top rated scripts »

Download

Choose a competition & download the training data.

Build

Build a model using whatever methods and tools you prefer.

Submit

Upload your predictions. Kaggle scores your solution and shows your score on the leaderboard.

Active Competitions

All Competitions

Active Competitions



Ultrasound Nerve Segmentation
Identify nerve structures in ultrasound images of the neck

2 months
92 teams
132 scripts
$100,000

Draper Satellite Image Chronology
Can you put order to space and time?

33 days
261 teams
332 scripts
$75,000

State Farm Distracted Driver Detection
Can computer vision spot distracted drivers?

2 months
854 teams
563 scripts
$65,000

Expedia Hotel Recommendations
Which hotel type will an Expedia customer book?

16 days
1549 teams
1846 scripts
$25,000

Avito Duplicate Ads Detection
Can you detect duplicitous duplicate ads?

47 days
231 teams
155 scripts
$20,000

Facebook V: Predicting Check Ins

42 days