Supervised Learning Fundamentals

AI FUNDAMENTALS

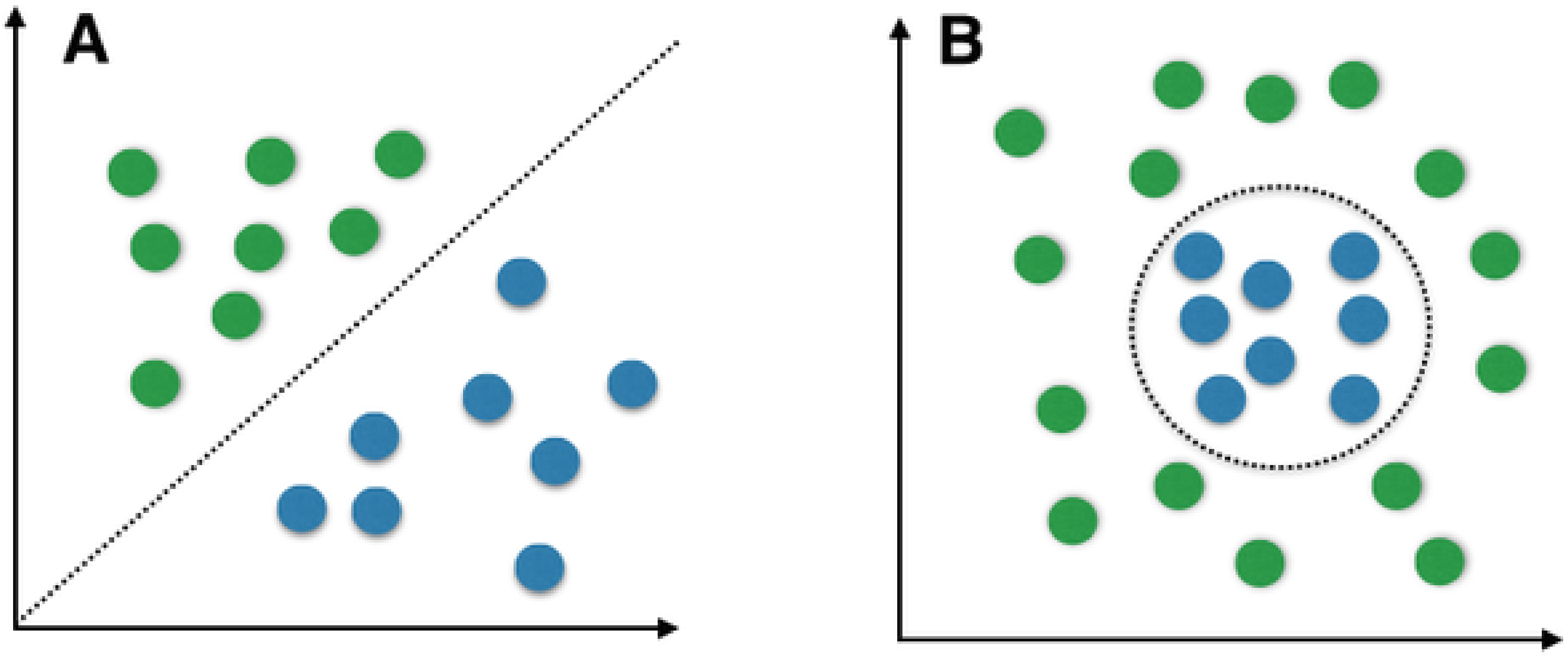
Nemanja Radojkovic
Senior Data Scientist

Classification: Definition and intuition

- **Intuition:** Classification == *"Putting things in boxes"*
- **Targets:** only categorical
- **Inputs:** numerical, categorical

Creating Classification Models

Intuition: Creating the "category boxes" based on training data.



Common Classification models

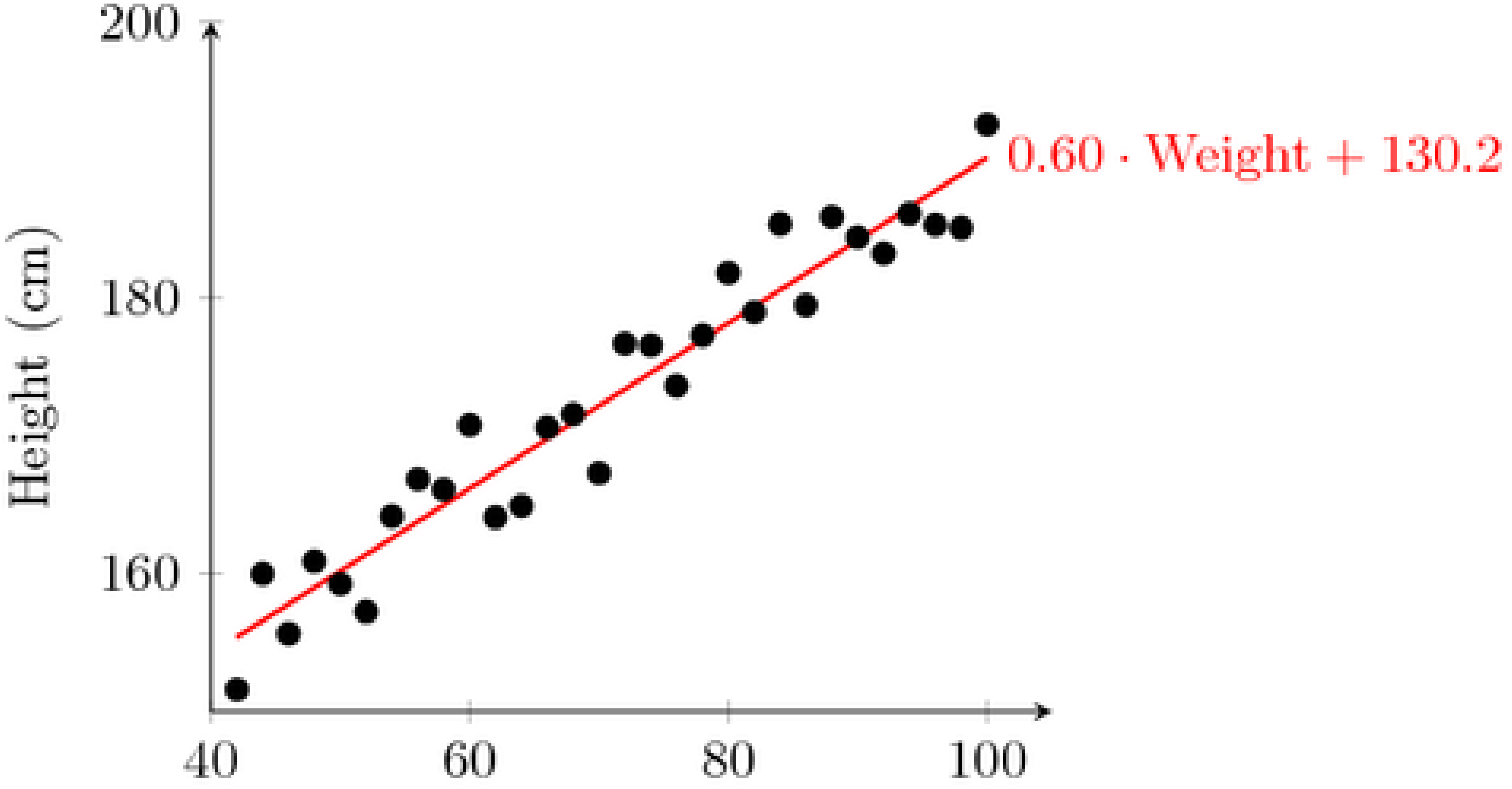
```
# Decision trees
from sklearn.tree import DecisionTreeClassifier

# Logistic regression
from sklearn.linear_model import LogisticRegression

# Support Vector Machine
from sklearn.svm import SVC

# Random Forest
from sklearn.ensemble import RandomForestClassifier
```

Regression: Definition and intuition

Weather...

... or sports

Creating Regression models

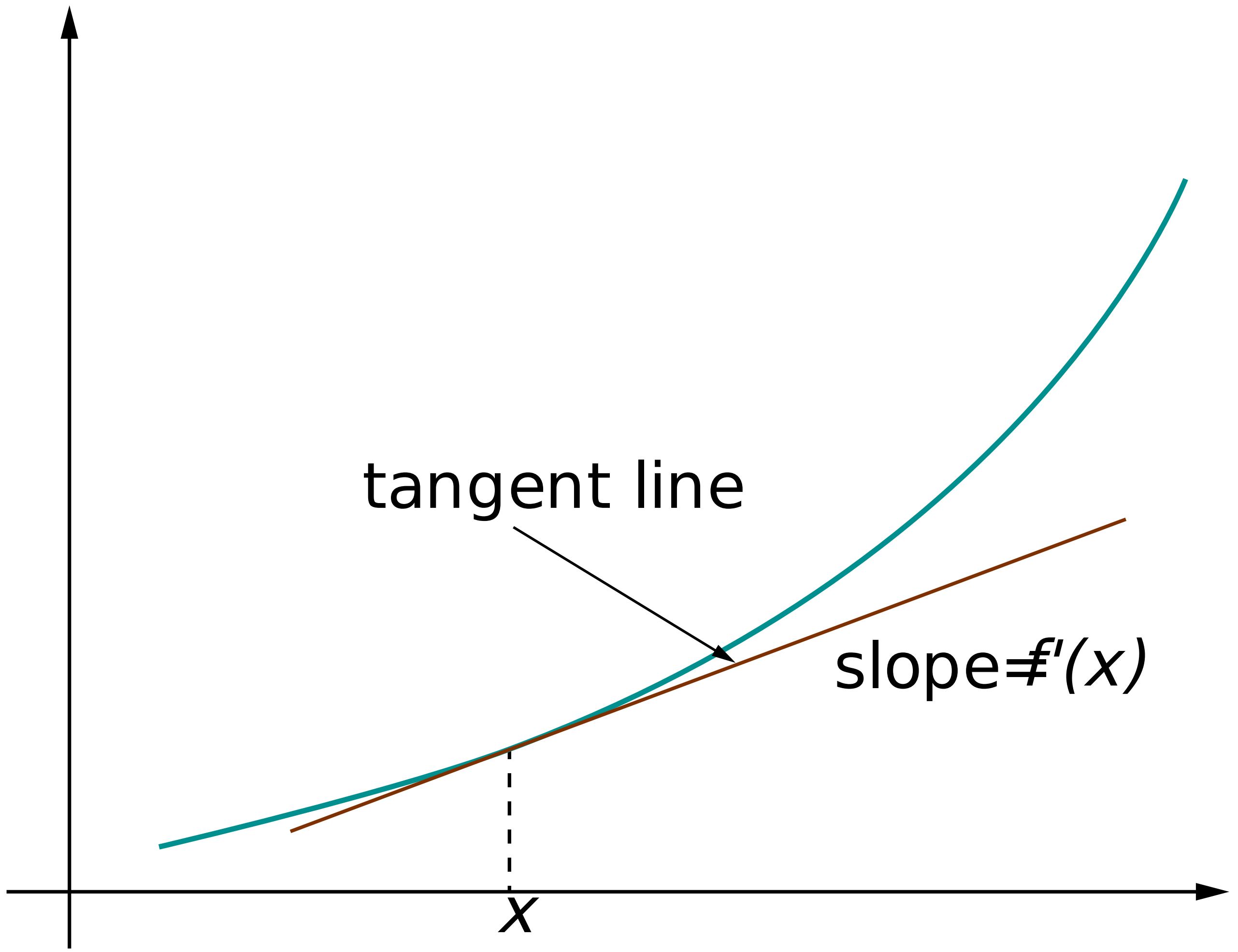


Common Regression Models

```python
# Ordinary Least Squares Regression
from sklearn.linear_model \
    import LinearRegression
```

```python
# Lasso Regression
from sklearn.linear_model \
    import Lasso
```

```python
# Ridge Regression
from sklearn.linear_model \
    import Ridge
```



Classification and Regression

AI FUNDAMENTALS

Training and evaluating classification models

AI FUNDAMENTALS

Nemanja Radojkovic
Senior Data Scientist

Train/test splitting

Test data ? training data

Simplest approach (Hold-out method)

- 60% of all data used for training
- remaining 40% of data used for testing

Code example:

```python
from sklearn.model_selection \
    import train_test_split

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.4)
```

Full labeled dataset

X (inputs)			y (target)	
x1	x2	x3	y	
8,0	0,70	13,2	C	X_train, y_train (random 60%)
3,0	0,23	6,1	D	
8,0	0,85	10,1	D	
8,0	0,43	4,2	A	
9,0	0,93	16,3	D	
8,0	0,40	16,0	D	
4,0	0,18	5,3	C	X_test, y_test (remaining 40%)
6,0	0,75	7,7	D	
10,0	0,79	1,7	B	
7,0	0,03	3,1	B	

Model training

Use the default model configuration/hyper-parameters:

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
```

Use a custom model configuration/hyper-parameters:

```
model = RandomForestClassifier(n_estimators=500, # Number of trees
                               max_depth=20)     # Tree depth
```

Start the training procedure:

```
model.fit(X_train, y_train)
```

Model testing

Generic syntax

```
model.predict(X=X_test)
```

Example: News title classifier

```
model.predict(X=['Denver Nuggets win against GSW and clinch playoff spot!'])
```

```
Out: ['Sport']
```

Inspecting model outputs

```
y_predicted = model.predict(X_test_all)
```

Is y_predicted == y_true ?

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true, y_predicted)
```

Inspecting model outputs

```
y_predicted = model.predict(X_test_all)
```

Is y_predicted == y_true ?

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true, y_predicted)
```

The confusion matrix:

	REALITY: YES	REALITY: NO
PREDICTION: YES	560	80
PREDICTION: NO	50	210

Confusion matrix: True positives

	Diabetes present	No diabetes
Diabetes predicted	TRUE POSITIVES	
No diabetes predicted		

TRUE POSITIVE = the model predicts diabetes and the patient is actually suffering from it.

Confusion matrix: True negatives

	Diabetes present	No diabetes
Diabetes predicted	true positives	
No diabetes predicted		**TRUE NEGATIVES**

TRUE POSITIVE = the model predicts diabetes and the patient is actually suffering from it.

TRUE NEGATIVE = model predicts no diabetes and the patient is actually healthy.

Confusion matrix: False positives

	Diabetes present	No diabetes
Diabetes predicted	true positives	**FALSE POSITIVES**
No diabetes predicted		true negatives

TRUE POSITIVE = the model predicts diabetes and the patient is actually suffering from it.

TRUE NEGATIVE = model predicts no diabetes and the patient is actually healthy.

FALSE POSITIVE = model predicts diabetes but the patient is actually healthy (**Type I error**).

Confusion matrix: False negatives

	Diabetes present	No diabetes
Diabetes predicted	true positives	false positives
No diabetes predicted	**FALSE NEGATIVES**	true negatives

TRUE POSITIVE = the model predicts diabetes and the patient is really suffering from it.

TRUE NEGATIVE = model predicts no diabetes and the patient is really healthy.

FALSE POSITIVE = model predicts diabetes but the patient is actually healthy (**Type I error**).

FALSE NEGATIVE = diabetes present but not detected by the model (**Type II error**).

Accuracy, precision, recall

Metrics:

- **Accuracy:** "How often did I make the correct diagnosis?"
- **Precision:** "How often was I correct when I said a person has diabetes?" (= 1 - T1 error)
- **Recall:** "What percentage of actual diabetes cases did my model detect?" (= 1 - T2 error)

Code example using Python + Scikit-learn

Using **Python** and **scikit-learn**:

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score

accuracy_score(y_true, y_predicted) # Same arguments for precision and recall
```

```
Result: 0.88
```

Knowledge check!

AI FUNDAMENTALS

Training and evaluating regression models

AI FUNDAMENTALS

Nemanja Radojkovic
Senior Data Scientist

Different but the same

Difference compared to classification:

- **Target variable:** Numerical (quantities)
- **Model structure:** a line or surface fitted closely to the data, not separating it into regions.
- **Key metrics:** Mean Absolute Error (MAE), Root Mean Squared Error (RMSE).

Same:

- train/test splitting
- fit/predict functions and arguments
- the impact of data quality.

Going non-linear

Input features: (a, b)

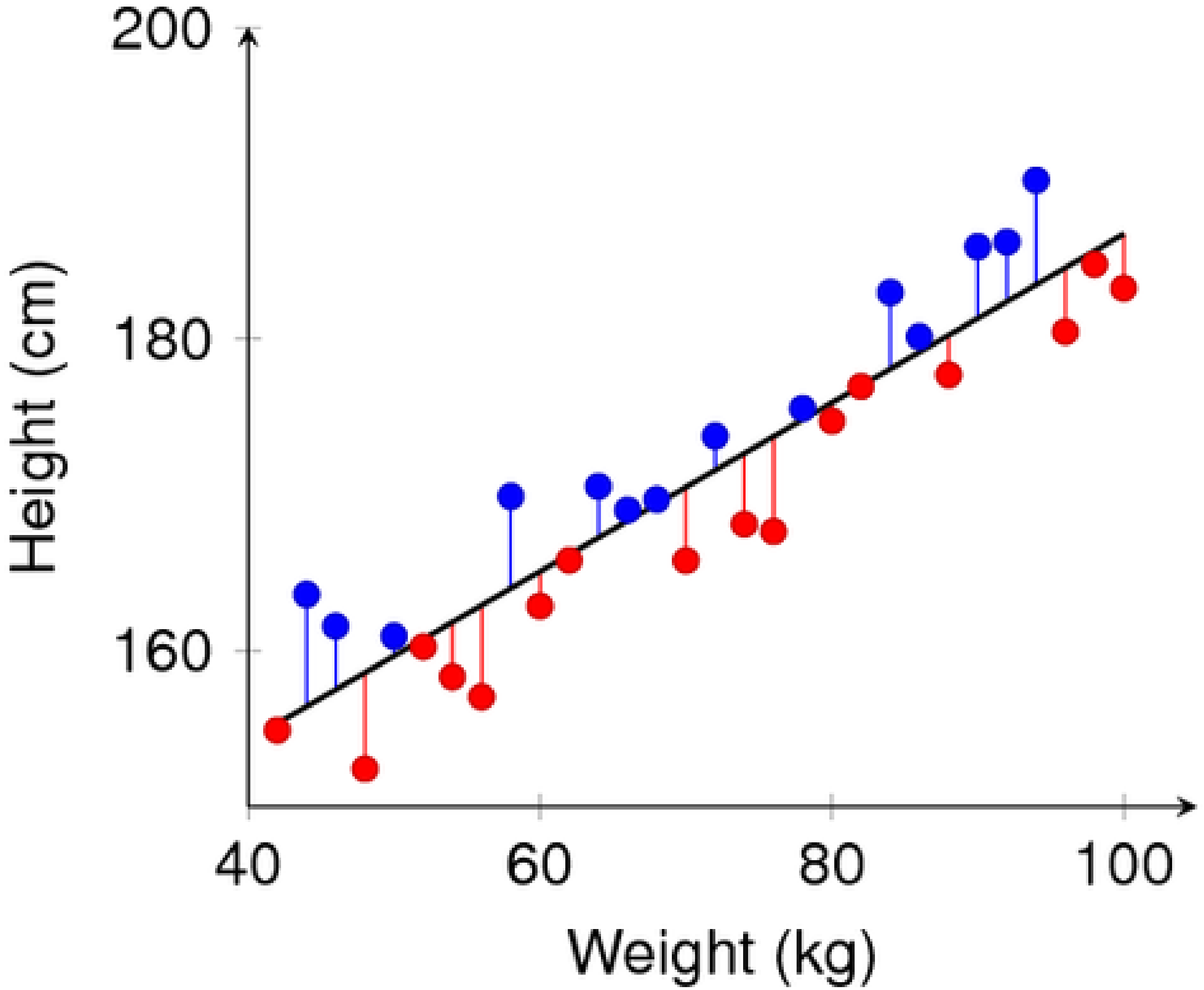
Output features: (1, a, b, a^2, a*b, b^2)

```
from sklearn.preprocessing import PolynomialFeatures

# Setup the preprocessor
poly = PolynomialFeatures(degree=2)
```

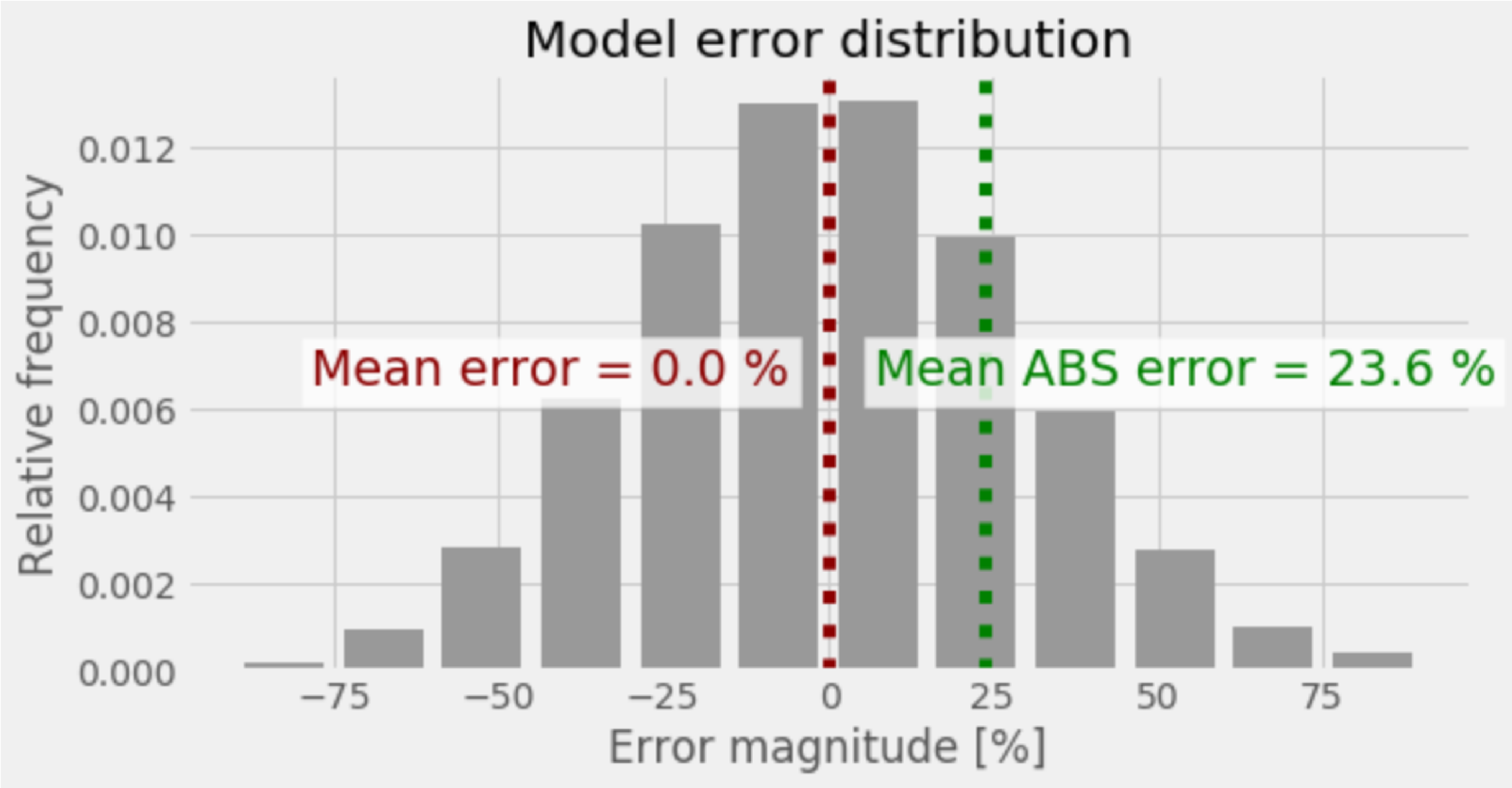
```
# Apply the transformation
polynomial_features_array = poly.fit_transform(linear_features_array)
```

```
model.fit(polynomial_features_array, y_train)
```

Regression error

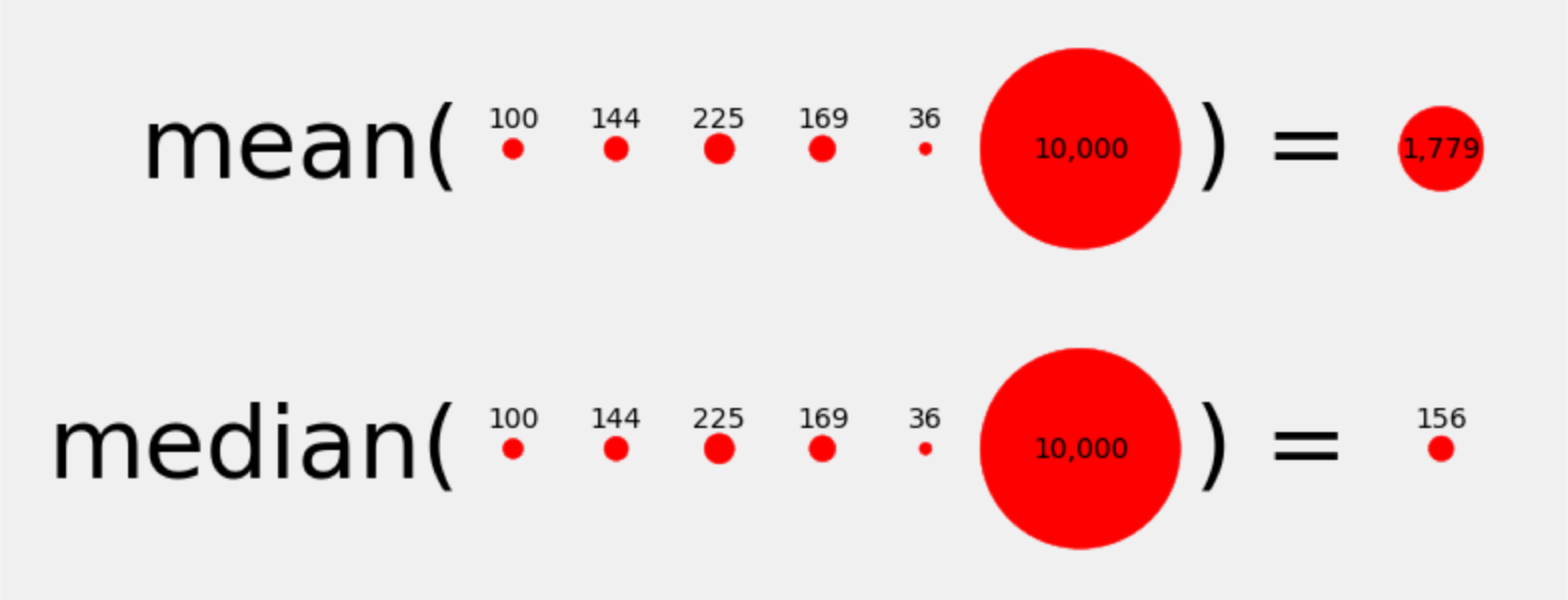


Regression metrics: "Raw" vs. Absolute



Unit-less alternative: **R^2 score**

Regression metrics: Mean vs. Median



Regression metrics: Code examples

```
# Mean absolute error; range: [-Inf..+Inf]
from sklearn.metrics import mean_absolute_error

# Median absolute error; range: [-Inf..+Inf]
from sklearn.metrics import median_absolute_error

# R^2 (coefficient of determination); range: [0..1]
from sklearn.metrics import r2_score
```

Example:

```
r2_score(y_true, y_predicted)
```

```
Out: 0.72
```

Practice time!

AI FUNDAMENTALS