quantco

Machine Learning with Python

Hands-on, from zero to hero

quantco

Introduction

Bozhidar Vasilev

Software @ Quantco
(Public Health Insurance)

Ivan Dimitrov

Software @ Quantco
(Public Health Insurance)

Simeon Stoykov

Software @ Quantco
(Car Insurance Pricing)

Why Bother with ML

State of the art for many relevant problems:

- Cool AI
- Automate/optimize business decisions:
  - Pricing
  - Fraud detection
  - Demand prediction
  - Medical forecasting
- Trading

Even if you are not a Data Scientists, you need to understand the workflows in order to support them.

Ecosystem

Infrastructure:

- Python
- conda
- Jupyter Notebook

Machine Learning:

- "Traditional":
  - Feature Engineering
  - Models
- Deep Learning (out-of-scope)
- Unsupervised (out-of-scope)
- Reinforcement (out-of-scope)

Libraries:

- `numpy` - arrays
- `pandas` - tables
- `scipy` - stats
- `scikit-learn` - models
- `matplotlib.pyplot` - visualization
- `pytorch` - neural networks (out-of-scope)
- `opencv` - computer vision (out-of-scope)

Infrastructure

Set up your tools for success.

quantco

How to Install Things

Install:

- fetch code
- put it at the right place

Things:

- software
- libraries

With package managers! They know:

- where to download things from;
- how to organize them;
- how to reuse them and save memory.

Conda

Some theory:

- The installable things are called *packages*
- Packages hosted remotely in *channels*
- You can install packages in local *environments*
- Conda is both a specification and a program (alternatives: `pixi`, `micromamba`)

Hands-on:

- Follow the instructions from `https://github.com/SimeonStoykovQC/workshop-boilerplate`

Jupyter Notebooks

Great for prototyping and experimenting:

- *Cells* contain runnable code
- *Notebooks* are collections of cells
- *Jupyter Lab*: a web-based IDE for managing notebooks

Hands-on:

- Create a new Notebook
- Navigate around & run simple code
- Shortcuts

Python

Why people use Python:
- Rich ecosystem of libraries
- Extensible through bindings for other languages
- Easy syntax and rapid prototyping
- Object-oriented support

Hands-on:
- Varaibles and lists
- Loops and comprehensions
- Functions
- Objects

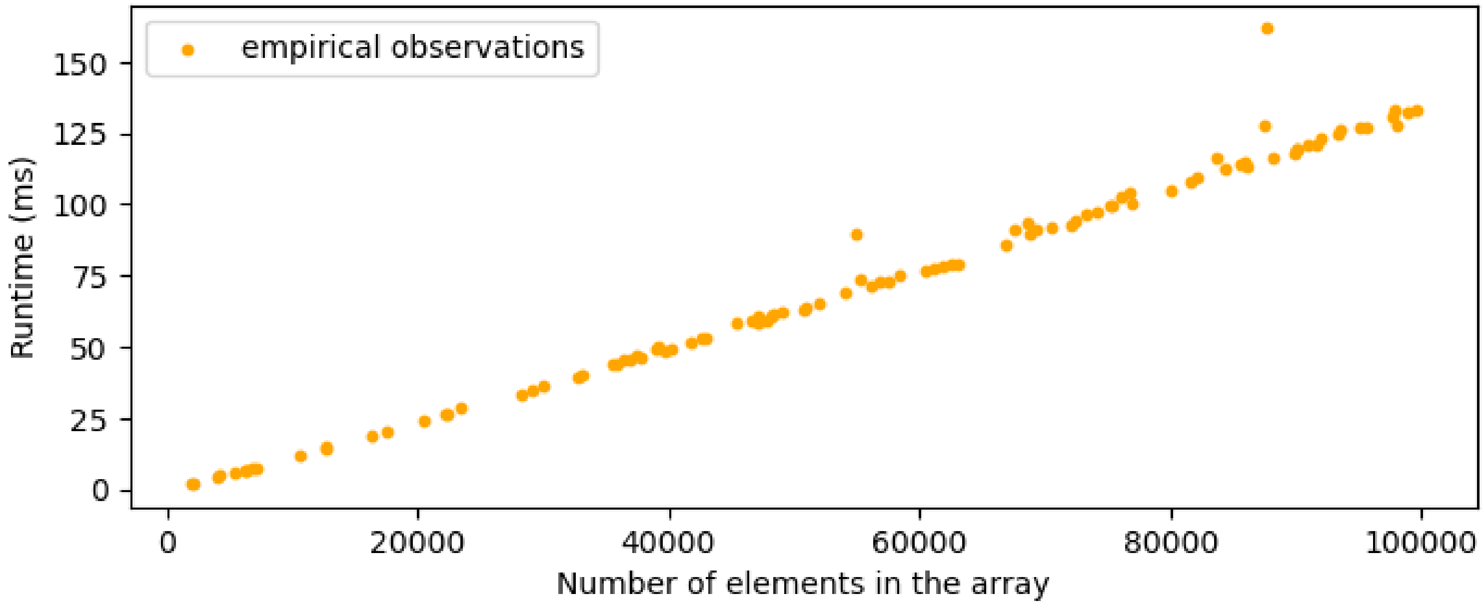
Machine Learning

Just statistics. On steroids.

quantco

A simple example

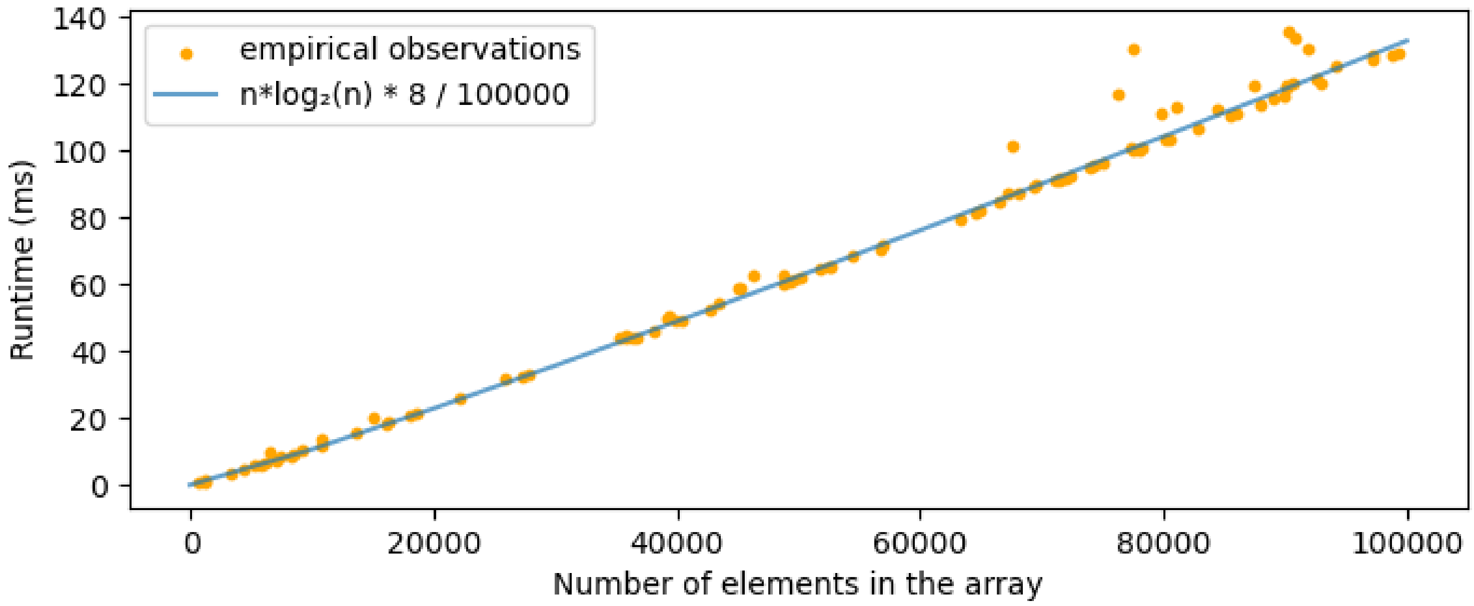
From past data, predict the runtime of my Merge Sort implementation for future runs.



	array_length	runtime_ms
0	89766	130.441833
1	82457	109.976458
2	95942	130.564833
3	23185	26.879167
4	81557	106.686167
...	...	...
95	78528	103.033958
96	94578	125.000959
97	89313	118.009458
98	29673	35.491125
99	73691	96.230542

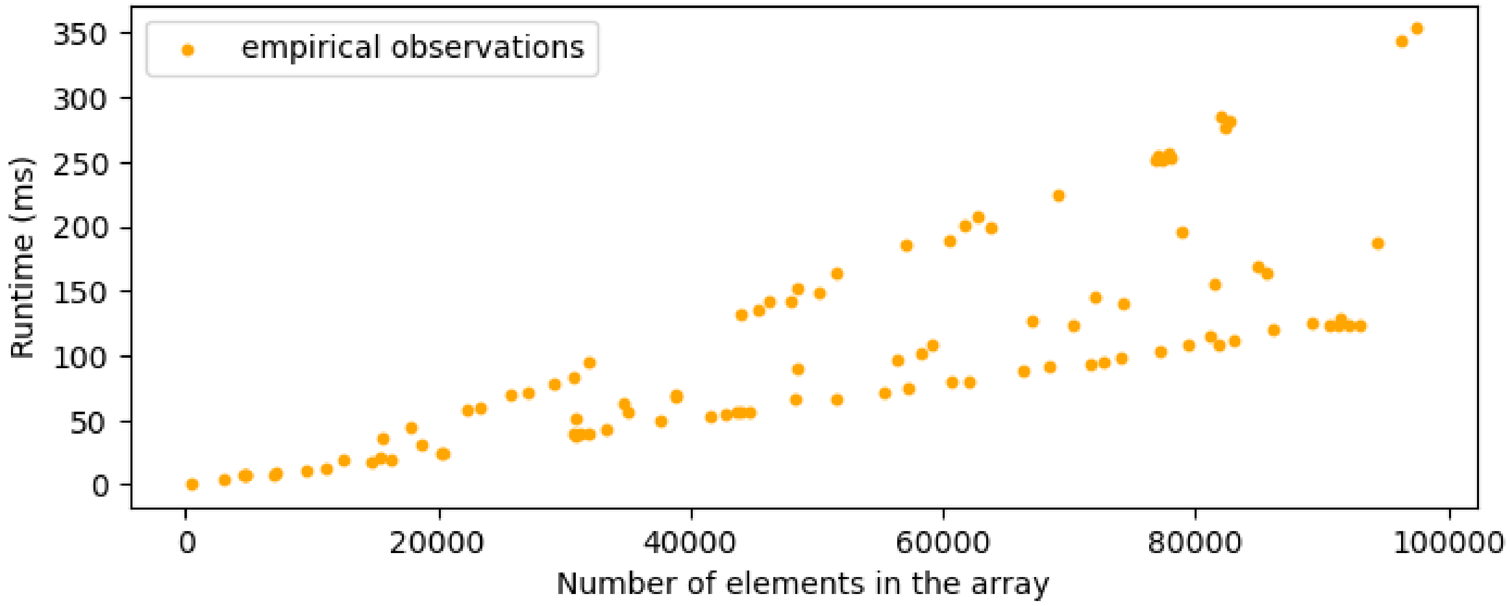
A simple example

We expect that the runtime scales proportionally to the complexity of Merge Sort: $n \cdot \log_2(n)$



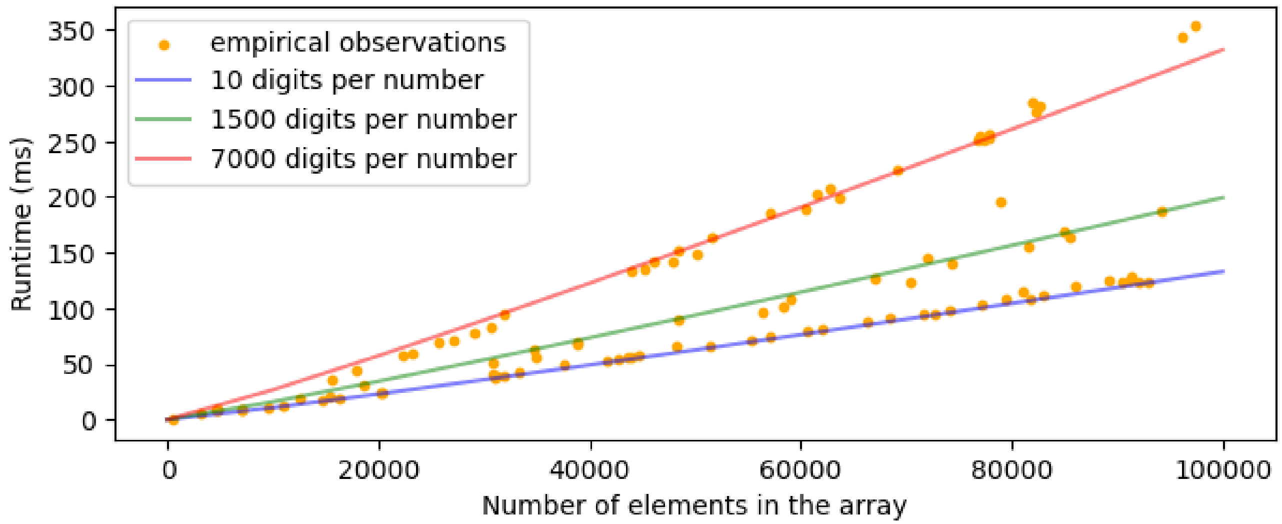
A slightly more complicated example

What could have changed?



A slightly more complicated example

Adding more “features” to your data can help explain and “learn” it better.



	array_length	num_digits	runtime_ms
0	72712	10	95.245709
1	31922	10	38.890250
2	42831	10	53.799792
3	83067	10	111.578125
4	499	10	0.401417
...	...	...	...
95	22264	7000	58.389583
96	4579	7000	8.255084
97	51584	7000	164.453542
98	97407	7000	353.574208
99	61637	7000	201.725875

Machine Learning* in a nutshell

- You have past observations, that you need to clean and transform to be useful.
- You decide how to “model” that data.
- You find the best parameters for your model.

*There are other types of Machine Learning. This is the most classic set up.

Feature Engineering

- Clean the data:
  - Fill in any missing values
  - Delete bad rows
  - Delete bad columns
- Make the data more “learnable”:
  - Encode categorical (non-numeric) variables
  - Normalize values
  - Combine features to create new ones
  - Use domain knowledge

Models

- Linear regression
- Decision trees
- Ensemble models
- Gradient boosting

Hands-on ML

quantco

Kaggle

`kaggle.com` is a platform for AI competitions:

- A bit like “marathon” tasks, but you won’t get far with algorithms and heuristics.
- You get training data with outputs, and test data for which you submit the predictions.

Examples:

- Predict real estate price.
- Score essays.
- Generate art.

Hands-on:

- Create an account on `kaggle.com`
- Enroll for the Titanic competition

Today's goal - an MVP submission

MVP - "Minimum Viable Product":

- The slimmest possible version of something,
- that still satisfies the minimum requirements.

In this case:

- The simplest possible data transformations;
- that allow to successfully train a model;
- and produce a working submission.

Dataframes

- Dataframe = table data (2d arrays with column headers).
- Allow for transformations: add or drop columns and rows, transform values…
- Read from/write to various formats, for example CSV and parquet.

Libraries:

- `pandas`: what everyone has been using in the past years
- `polars`: newer, faster alternative

Hands-on:

- Download the data
- Load the dataframes with `pandas`
- Split the training data into two: the features and the target

Modelling

`scikit-learn`:

- A massive collection of ML models and utilities around them.
- Mostly everything follows the same fit-predict interface:
    - `model = SomeModelClass(...)`
    - `model.fit(X, y)`
    - `y_pred = model.predict(X_new)`

Hands-on:

- Create a decision tree model with the `DecisionTreeClassifier`.
- Train it on the training data.
- Make predictions on the test data.
- Submit it on Kaggle.

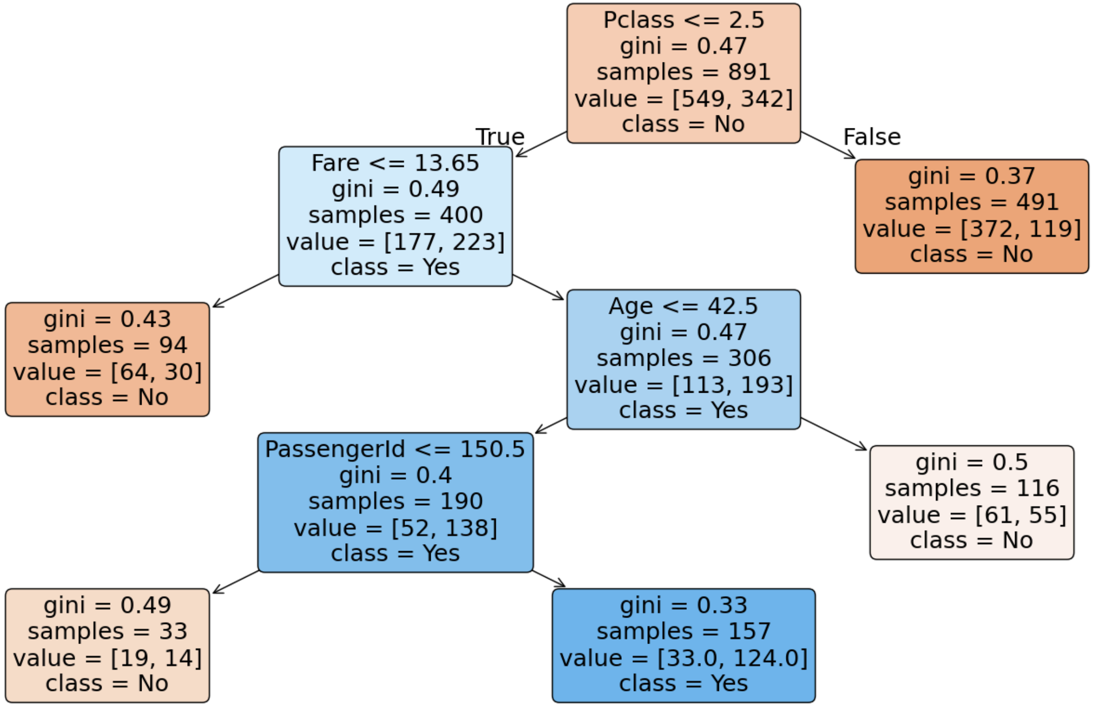
A small trained decision tree

[57]:
```python
X = df_train.drop(
    columns=["Survived"],
    inplace=False,
)
X.drop(
    columns=["Name", "Sex", "Ticket", "Cabin", "Embarked"],
    inplace=True,
)

y = df_train["Survived"]
```

[58]:
```python
from sklearn.tree import DecisionTreeClassifier
dtc_vis = DecisionTreeClassifier(max_leaf_nodes=5)
dtc_vis.fit(X, y)
```

[58]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_leaf_nodes=5)
```

[59]:
```python
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 10))  # Set the figure size
plot_tree(
    dtc_vis,
    feature_names=X.columns,
    class_names=["No", "Yes"],
    filled=True,
    rounded=True,
    fontsize=18,
    precision=2,
)
plt.show()
```



Overfitting

- When your model only performs well on the training data.
- Should only learn “generalisable trends”.
- Should NOT learn training data specifics that do not extend to the test data.
- Overfitting on the data we have = underperforming on new data.

Train-test split

Kaggle competitions (and real-world problems) have a hidden dataset for grading:

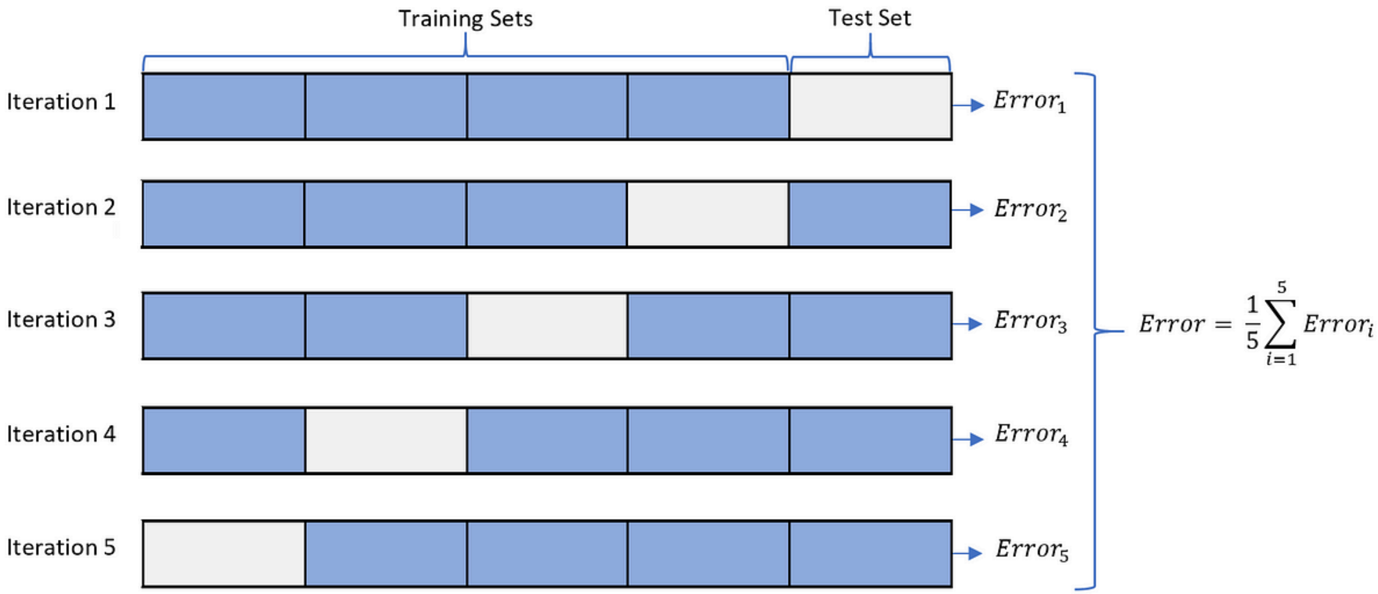
- Feature engineering and parameter tuning are part of the training;
- so even if you see an improvement, it could be misleading;
- so don't do it on all of the data!

Hands-on:

- Use the `train_test_split` function from scikit-learn to split your dataset;
- Evaluate your model locally with scikit-learn's `accuracy_score`, without submitting to Kaggle.

Cross-validation

Basically, a train-test split, done a few times:



Cross-validation

Why:

- It makes it more difficult to overfit in the process of:
  - feature engineering
  - model hyperparameter tuning
- A more accurate score of what you'll get on new data.

Hands-on:

- Use sklearn's `cross_val_score`

Improving our score

Improving the feature engineering:

- Remove features that add no information.
- Add the categorical features that we dropped.
- Create new features.

Improving the model we use:

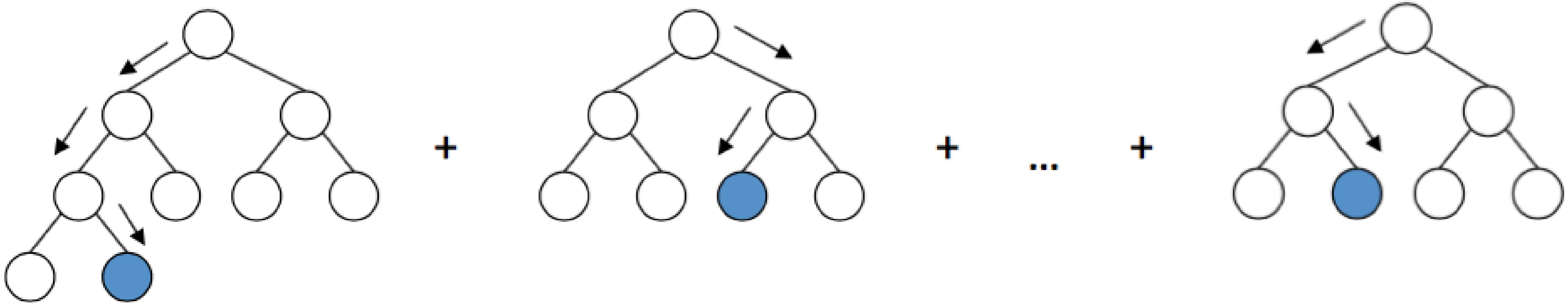
- Use Random Forests
- Use Gradient-Boosted Forests

ML theory - on the whiteboard

How models work under the hood.

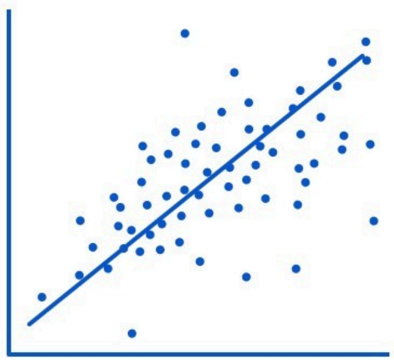
quantco

Decision Trees -> Forests -> Gradient-boosting



Linear Regression

Simple Linear Regression



Multiple Linear Regression