

scikit-learn and nemos

Billy Broderick

Jan 2025

https://scikit-learn.org

The screenshot shows the scikit-learn website homepage. At the top, there is a navigation bar with links for 'Install', 'User Guide', 'API', 'Examples', 'Community', and 'More'. The main header features the 'scikit-learn' logo and the tagline 'Machine Learning in Python'. Below the header, there are buttons for 'Getting Started' and 'Release Highlights for 1.6'. A list of key features is provided: 'Simple and efficient tools for predictive data analysis', 'Accessible to everybody, and reusable in various contexts', 'Built on NumPy, SciPy, and matplotlib', and 'Open source, commercially usable - BSD license'. The main content area is a grid of six topic cards: 'Classification', 'Regression', 'Clustering', 'Dimensionality reduction', 'Model selection', and 'Preprocessing'. Each card includes a brief description, applications, algorithms, and a grid of example images. The 'Classification' card shows handwritten digit images. The 'Regression' card shows a line plot of energy transfer. The 'Clustering' card shows a scatter plot of digits with centroids. The 'Dimensionality reduction' card shows a scatter plot of data points. The 'Model selection' card shows a line plot of model performance. The 'Preprocessing' card shows a grid of images with different preprocessing steps.

scikit-learn
Machine Learning in Python

Getting Started Release Highlights for 1.6

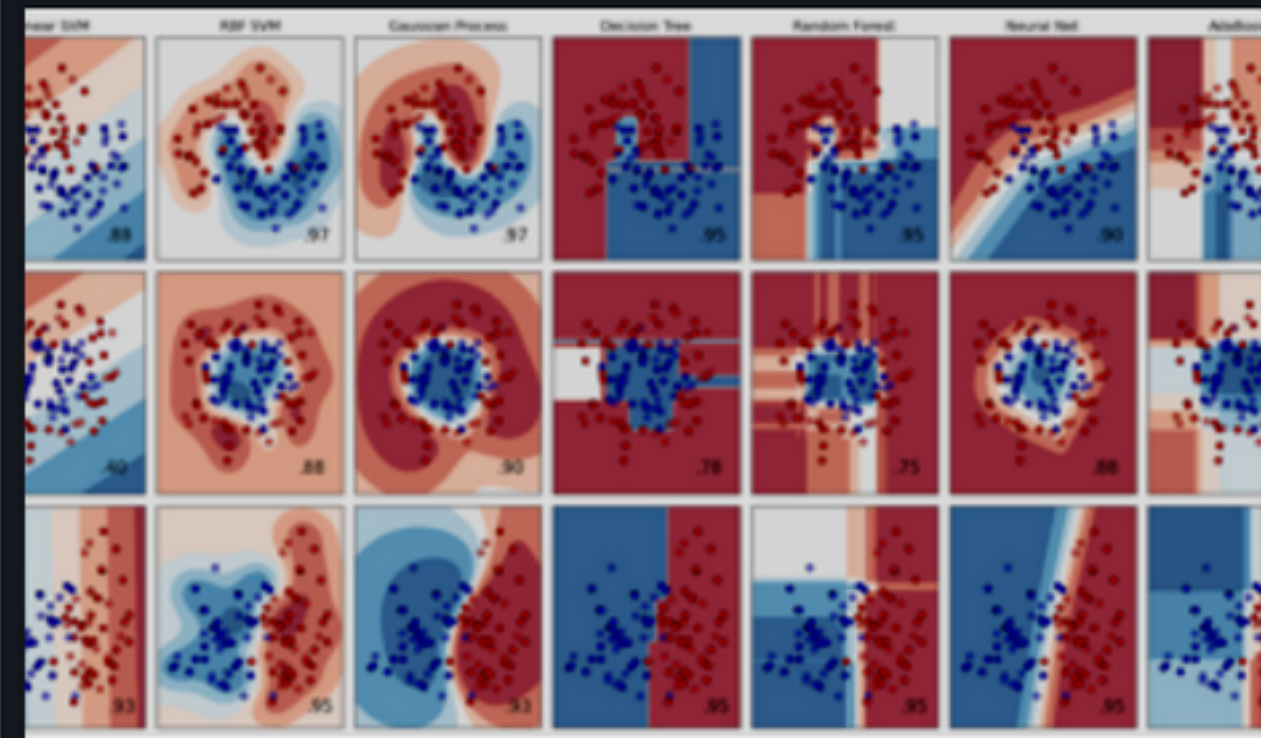
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: [Gradient boosting](#), [nearest neighbors](#), [random forest](#), [logistic regression](#), and [more...](#)



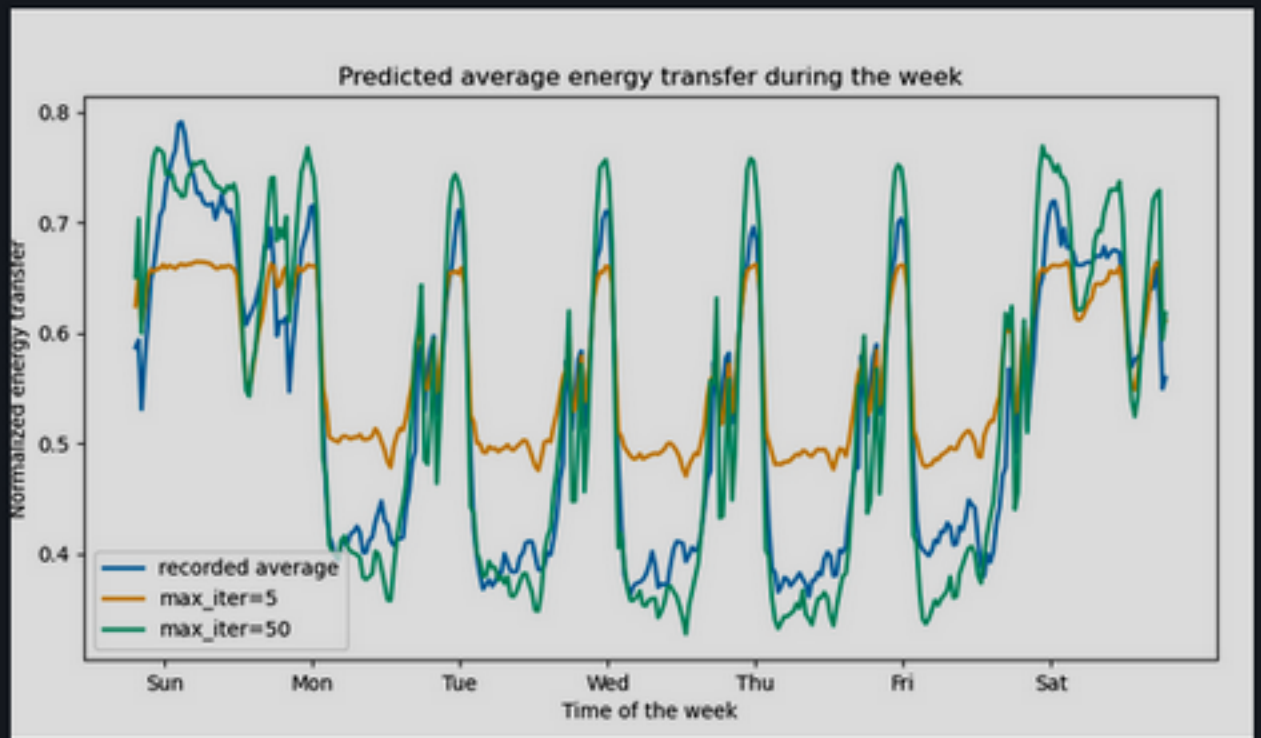
Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, stock prices.

Algorithms: [Gradient boosting](#), [nearest neighbors](#), [random forest](#), [ridge](#), and [more...](#)



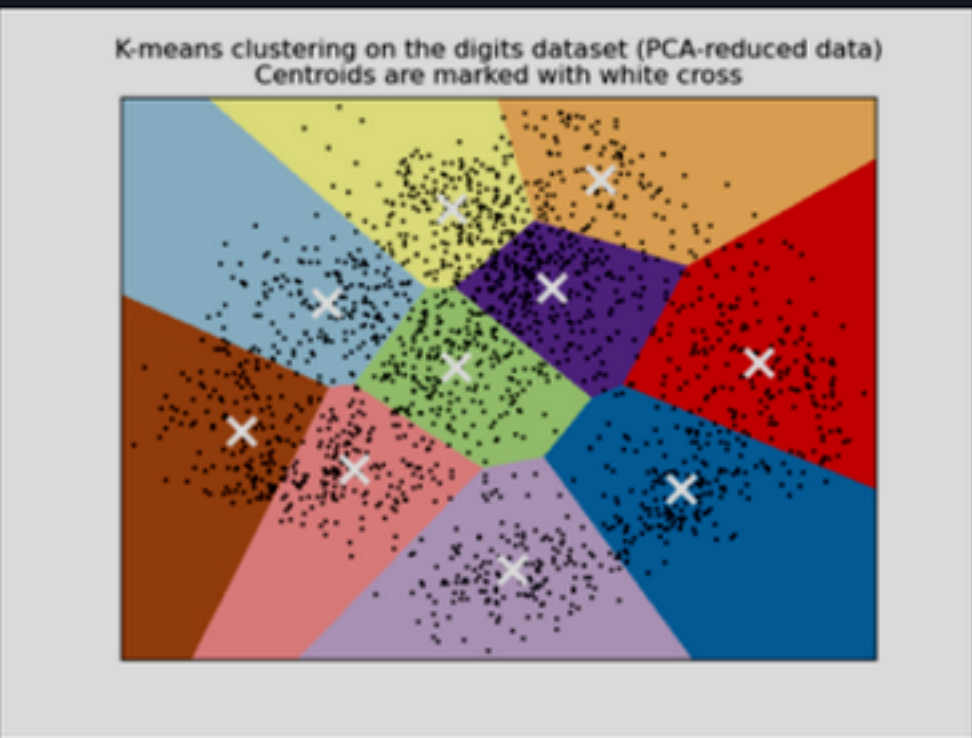
Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, grouping experiment outcomes.

Algorithms: [k-Means](#), [HDBSCAN](#), [hierarchical clustering](#), and [more...](#)



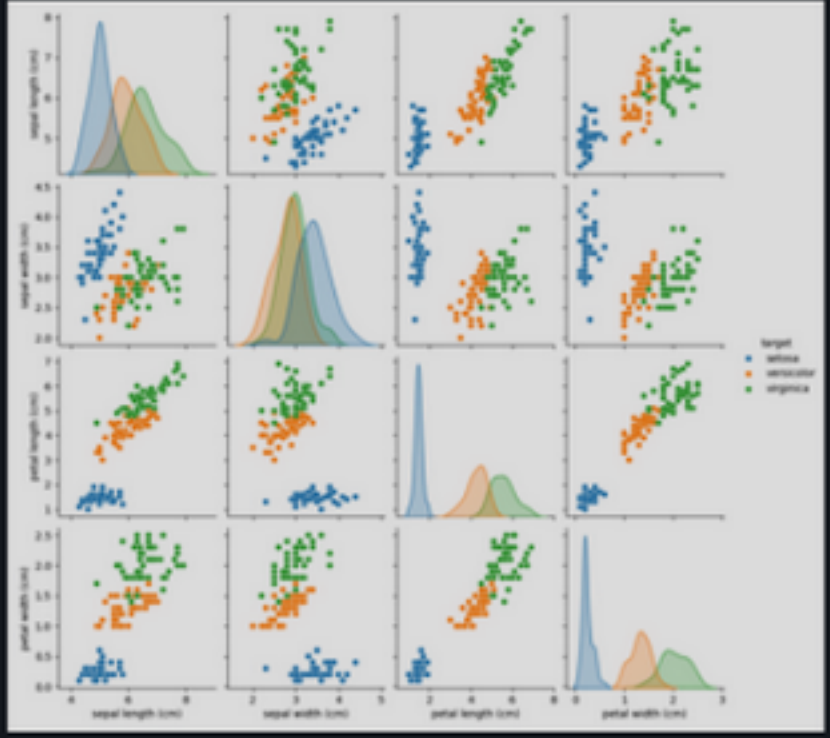
Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, increased efficiency.

Algorithms: [PCA](#), [feature selection](#), [non-negative matrix factorization](#), and [more...](#)



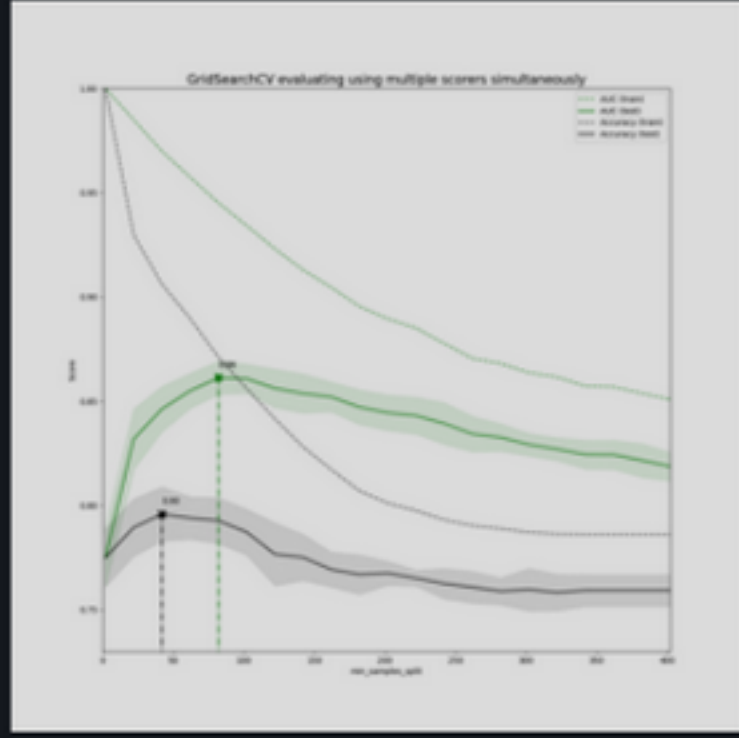
Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning.

Algorithms: [Grid search](#), [cross validation](#), [metrics](#), and [more...](#)



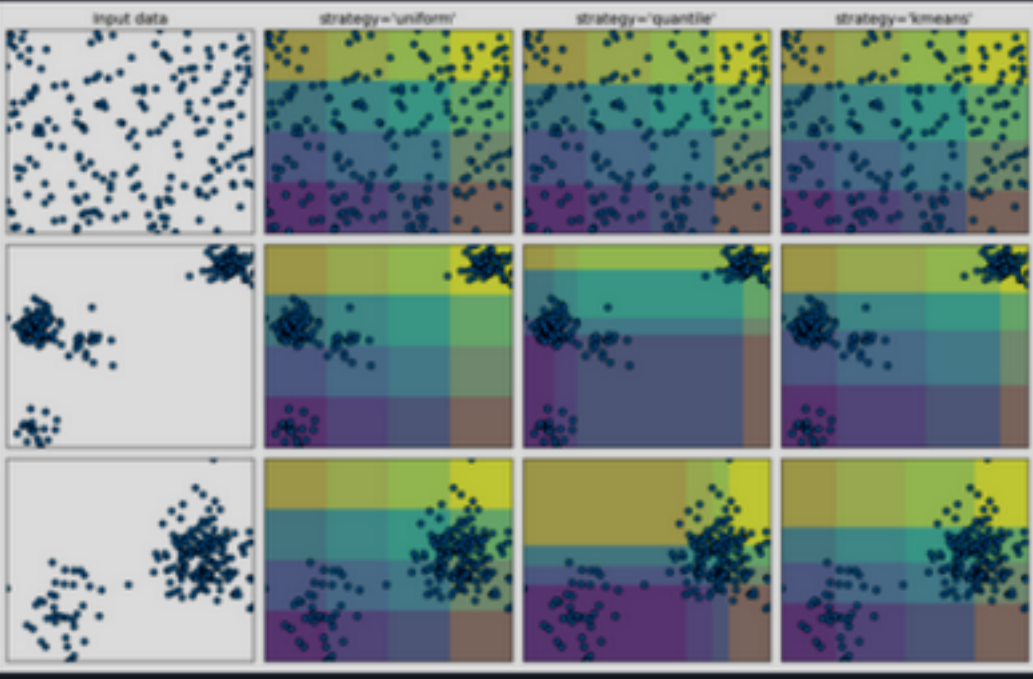
Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.

Algorithms: [Preprocessing](#), [feature extraction](#), and [more...](#)



Examples

https://scikit-learn.org

scikit-learn
Machine Learning in Python

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification
Identifying which category an object belongs to.
Applications: Spam detection, image recognition.
Algorithms: [Gradient boosting](#), [nearest neighbors](#), [random forest](#), [logistic regression](#), and [more...](#)

Regression
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, stock prices.
Algorithms: [Gradient boosting](#), [nearest neighbors](#), [random forest](#), [ridge](#), and [more...](#)

Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, grouping experiment outcomes.
Algorithms: [k-Means](#), [HDBSCAN](#), [hierarchical clustering](#), and [more...](#)

Dimensionality reduction
Reducing the number of random variables to consider.
Applications: Visualization, increased efficiency.
Algorithms: [PCA](#), [feature selection](#), [non-negative matrix factorization](#), and [more...](#)

Model selection
Comparing, validating and choosing parameters and models.
Applications: Improved accuracy via parameter tuning.
Algorithms: [Grid search](#), [cross validation](#), [metrics](#), and [more...](#)

Preprocessing
Feature extraction and normalization.
Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: [Preprocessing](#), [feature extraction](#), and [more...](#)

scikit-learn objects

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

or:

```
estimator = estimator.fit(data)
```

Predictor: For supervised learning, or some unsupervised problems, implements:

```
prediction = predictor.predict(data)
```

Classification algorithms usually also offer a way to quantify certainty of a prediction, either using `decision_function` or `predict_proba`:

```
probability = predictor.predict_proba(data)
```

Transformer: For modifying the data in a supervised or unsupervised way (e.g. by adding, changing, or removing columns, but not by adding or removing rows). Implements:

```
new_data = transformer.transform(data)
```

When fitting and transforming can be performed much more efficiently together than separately, implements:

```
new_data = transformer.fit_transform(data)
```

Model: A model that can give a [goodness of fit](#) measure or a likelihood of unseen data, implements (higher is better):

```
score = model.score(data)
```

scikit-learn objects

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

or:

```
estimator = estimator.fit(data)
```

Predictor: For supervised learning, or some unsupervised problems, implements:

```
prediction = predictor.predict(data)
```

Classification algorithms usually also offer a way to quantify certainty of a prediction, either using `decision_function` or `predict_proba`:

```
probability = predictor.predict_proba(data)
```

Transformer: For modifying the data in a supervised or unsupervised way (e.g. by adding, changing, or removing columns, but not by adding or removing rows). Implements:

```
new_data = transformer.transform(data)
```

When fitting and transforming can be performed much more efficiently together than separately, implements:

```
new_data = transformer.fit_transform(data)
```

Model: A model that can give a [goodness of fit](#) measure or a likelihood of unseen data, implements (higher is better):

```
score = model.score(data)
```

GLM
PopulationGLM

scikit-learn objects

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

or:

```
estimator = estimator.fit(data)
```

Predictor: For supervised learning, or some unsupervised problems, implements:

```
prediction = predictor.predict(data)
```

Classification algorithms usually also offer a way to quantify certainty of a prediction, either using `decision_function` or `predict_proba`:

```
probability = predictor.predict_proba(data)
```

Transformer: For modifying the data in a supervised or unsupervised way (e.g. by adding, changing, or removing columns, but not by adding or removing rows). Implements:

```
new_data = transformer.transform(data)
```

When fitting and transforming can be performed much more efficiently together than separately, implements:

```
new_data = transformer.fit_transform(data)
```

Model: A model that can give a [goodness of fit](#) measure or a likelihood of unseen data, implements (higher is better):

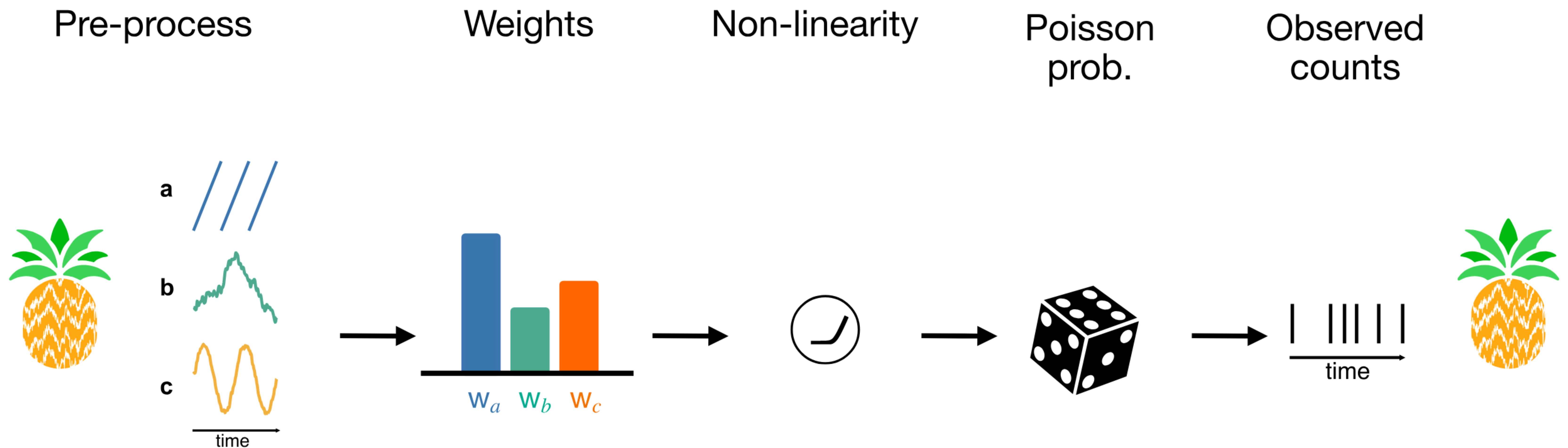
```
score = model.score(data)
```

GLM
PopulationGLM

GLM
PopulationGLM

GLM
PopulationGLM

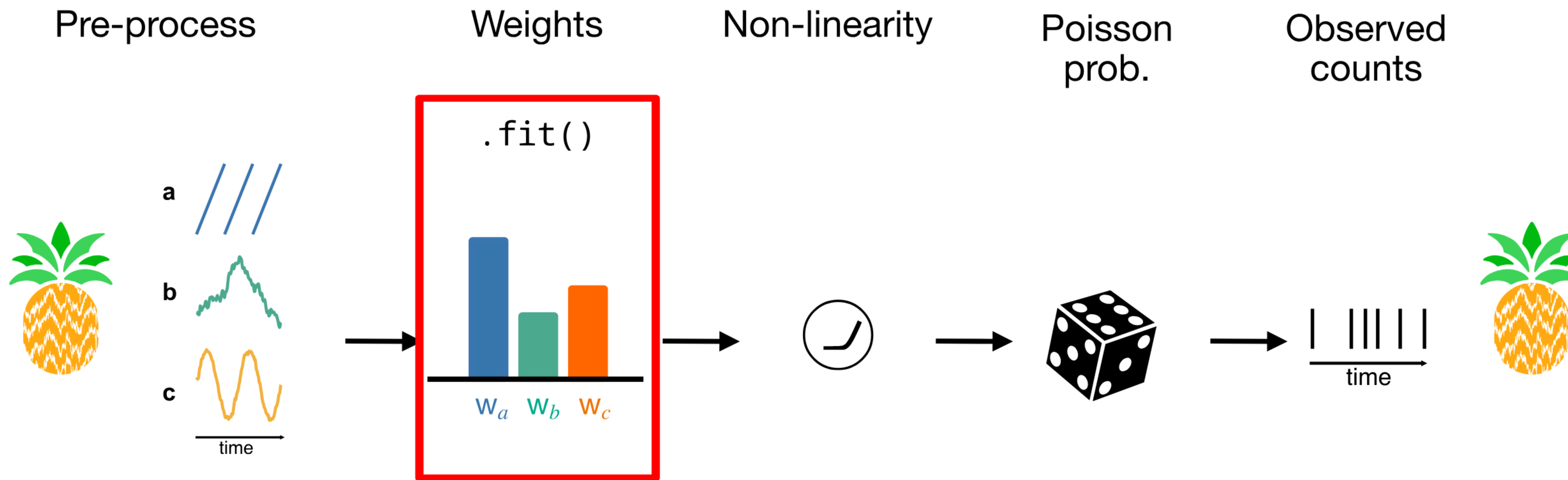
What are GLMs?



$$\text{firing rate} = \exp(\mathbf{a} \cdot \mathbf{w}_a + \mathbf{b} \cdot \mathbf{w}_b + \mathbf{c} \cdot \mathbf{w}_c)$$

$$\sum_k \text{Poisson}(k | \text{firing rate})$$

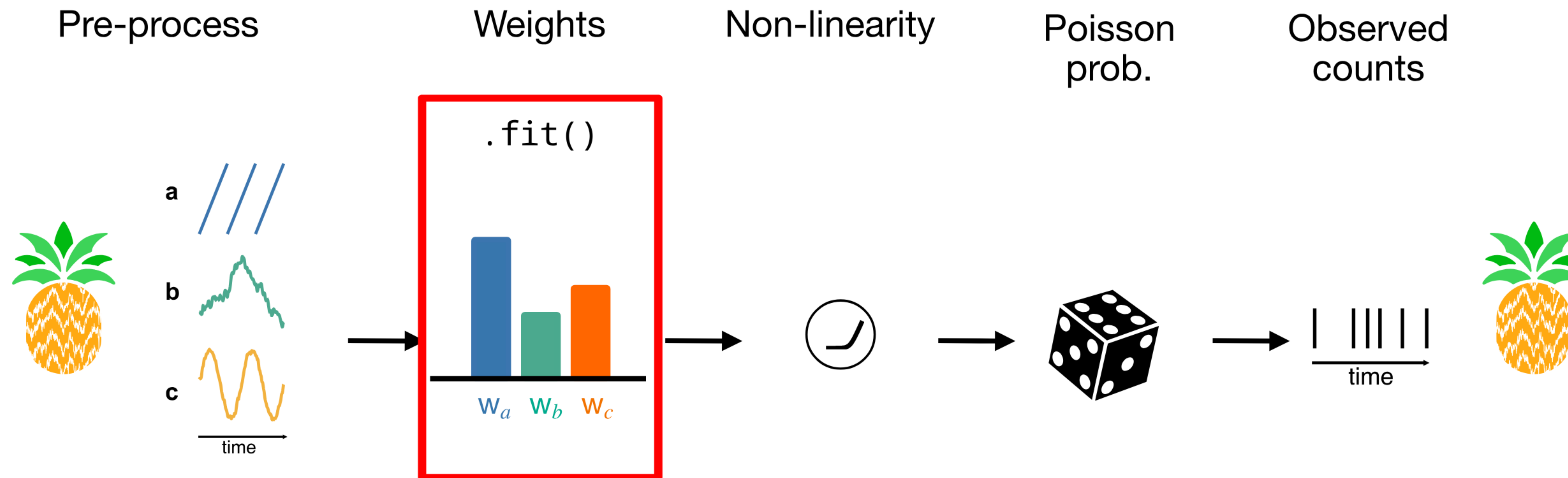
What are GLMs?



$$\text{firing rate} = \exp(\mathbf{a} \cdot \mathbf{w}_a + \mathbf{b} \cdot \mathbf{w}_b + \mathbf{c} \cdot \mathbf{w}_c)$$

$$\sum_k \text{Poisson}(k \mid \text{firing rate})$$

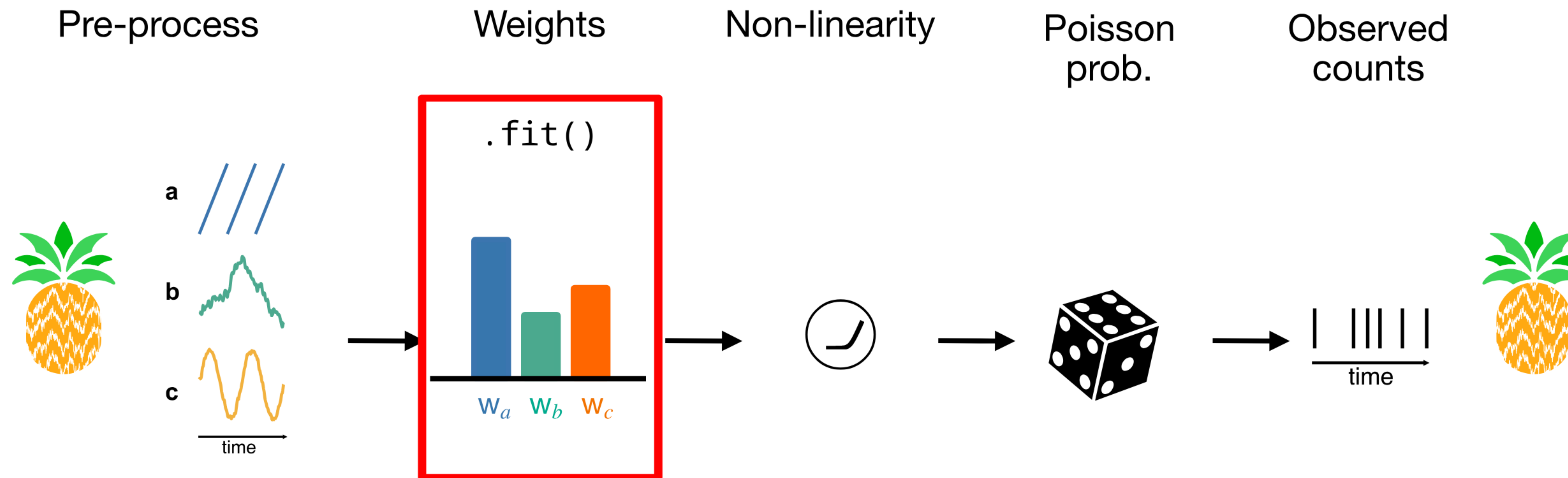
What are GLMs?



`.predict()` firing rate = $\exp(\mathbf{a} \cdot \mathbf{w}_a + \mathbf{b} \cdot \mathbf{w}_b + \mathbf{c} \cdot \mathbf{w}_c)$

$$\sum_k \text{Poisson}(k \mid \text{firing rate})$$

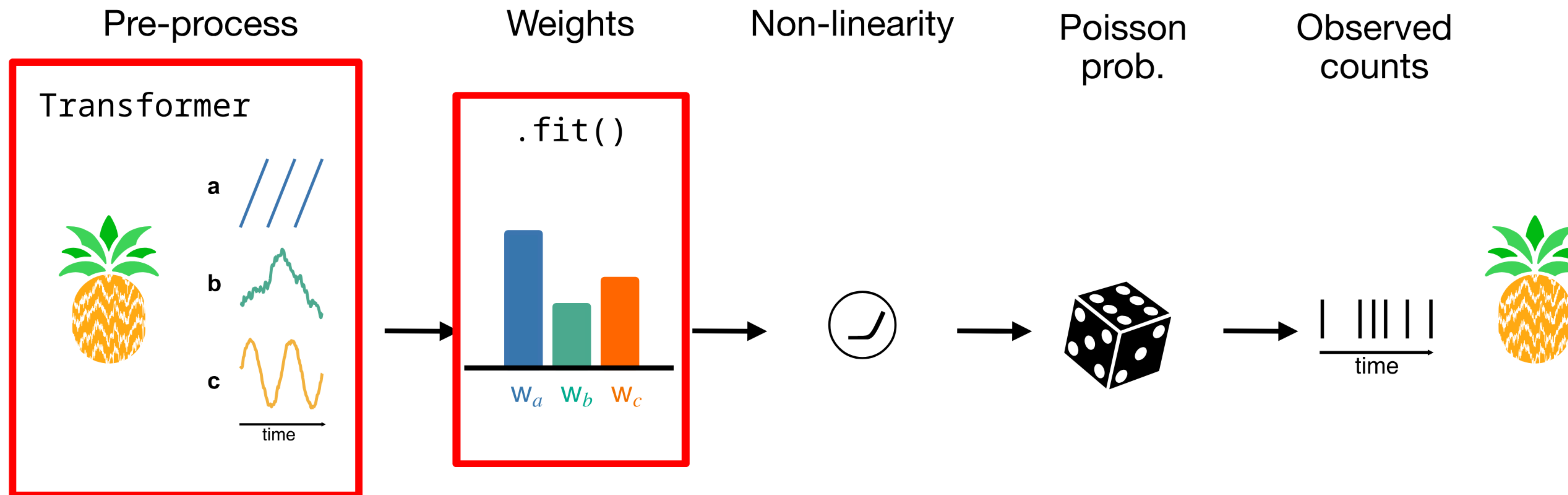
What are GLMs?



`.predict()` firing rate = $\exp(\mathbf{a} \cdot \mathbf{w}_a + \mathbf{b} \cdot \mathbf{w}_b + \mathbf{c} \cdot \mathbf{w}_c)$

`.score()` $\sum_k \text{Poisson}(k \mid \text{firing rate})$

What are GLMs?



`.predict()` firing rate = $\exp(\mathbf{a} \cdot \mathbf{w}_a + \mathbf{b} \cdot \mathbf{w}_b + \mathbf{c} \cdot \mathbf{w}_c)$

`.score()` $\sum_k \text{Poisson}(k \mid \text{firing rate})$

scikit-learn objects

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

or:

```
estimator = estimator.fit(data)
```

Predictor: For supervised learning, or some unsupervised problems, implements:

```
prediction = predictor.predict(data)
```

Classification algorithms usually also offer a way to quantify certainty of a prediction, either using `decision_function` or `predict_proba`:

```
probability = predictor.predict_proba(data)
```

Transformer: For modifying the data in a supervised or unsupervised way (e.g. by adding, changing, or removing columns, but not by adding or removing rows). Implements:

```
new_data = transformer.transform(data)
```

When fitting and transforming can be performed much more efficiently together than separately, implements:

```
new_data = transformer.fit_transform(data)
```

Model: A model that can give a [goodness of fit](#) measure or a likelihood of unseen data, implements (higher is better):

```
score = model.score(data)
```

GLM
PopulationGLM

GLM
PopulationGLM

GLM
PopulationGLM

scikit-learn objects

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

or:

```
estimator = estimator.fit(data)
```

Predictor: For supervised learning, or some unsupervised problems, implements:

```
prediction = predictor.predict(data)
```

Classification algorithms usually also offer a way to quantify certainty of a prediction, either using `decision_function` or `predict_proba`:

```
probability = predictor.predict_proba(data)
```

Transformer: For modifying the data in a supervised or unsupervised way (e.g. by adding, changing, or removing columns, but not by adding or removing rows). Implements:

```
new_data = transformer.transform(data)
```

When fitting and transforming can be performed much more efficiently together than separately, implements:

```
new_data = transformer.fit_transform(data)
```

Model: A model that can give a [goodness of fit](#) measure or a likelihood of unseen data, implements (higher is better):

```
score = model.score(data)
```

GLM
PopulationGLM

GLM
PopulationGLM

`basis.to_transformer()`
`TransformerBasis(basis)`

GLM
PopulationGLM

scikit-learn objects

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

```
# nemos: accepts any number of n-dim arrays  
basis.compute_features(position, speed, head_dir)
```

Pre

Tra

implements:

```
new_data = transformer.fit_transform(data)
```

Model: A model that can give a [goodness of fit](#) measure or a likelihood of unseen data, implements (higher is better):

```
score = model.score(data)
```

GLM
PopulationGLM

scikit-learn objects

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

Pre

```
# nemos: accepts any number of n-dim arrays  
basis.compute_features(position, speed, head_dir)
```

```
# sklearn: accepts one 2d array  
transformer.transform(data)
```

Tra

implements:

```
new_data = transformer.fit_transform(data)
```

Model: A model that can give a [goodness of fit](#) measure or a likelihood of unseen data, implements (higher is better):

```
score = model.score(data)
```

GLM
PopulationGLM

scikit-learn objects

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

Pre

```
# nemos: accepts any number of n-dim arrays  
basis.compute_features(position, speed, head_dir)
```

```
# sklearn: accepts one 2d array  
transformer.transform(data)
```

Tra

```
# nemos with sklearn: go from one to the other  
t_basis = basis.to_transformer().set_input_shape(3)  
t_basis.transform(data)
```

implements:

```
new_data = transformer.fit_transform(data)
```

Model: A model that can give a [goodness of fit](#) measure or a likelihood of unseen data, implements (higher is better):

```
score = model.score(data)
```

GLM
PopulationGLM

scikit-learn objects

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

or:

```
estimator = estimator.fit(data)
```

Predictor: For supervised learning, or some unsupervised problems, implements:

```
prediction = predictor.predict(data)
```

Classification algorithms usually also offer a way to quantify certainty of a prediction, either using `decision_function` or `predict_proba`:

```
probability = predictor.predict_proba(data)
```

Transformer: For modifying the data in a supervised or unsupervised way (e.g. by adding, changing, or removing columns, but not by adding or removing rows). Implements:

```
new_data = transformer.transform(data)
```

When fitting and transforming can be performed much more efficiently together than separately, implements:

```
new_data = transformer.fit_transform(data)
```

Model: A model that can give a [goodness of fit](#) measure or a likelihood of unseen data, implements (higher is better):

```
score = model.score(data)
```

GLM
PopulationGLM

GLM
PopulationGLM

basis.to_transformer()
TransformerBasis(basis)

GLM
PopulationGLM

scikit-learn pipelines

Pipeline

