If the glove fits
A gentle introduction to ML
Recap: Simulation

- Use code to generate data
- Evaluate mean, variance, etc of complex quantities over simulation scenarios.
- Basic statistics is a starting point to help you write the code
Recap: Linear Models

\[ y_i = A \cdot x_i + \epsilon \]

def simulate(x):
    A = #constant
    return A*x + np.random.randn(1)

Accurately models “small variation” systems
Each $y_i$ (simulation output) is generated by a function of $x_i$ plus noise.

**Theta:** Parameters that govern how $f$ behaves.
More Interesting Example

\[ y_i = ec\_count\left( \begin{bmatrix} 0.54 \\ ... \\ 0.48 \end{bmatrix} + \text{sampling\_error} \right) \]
Recap: Simulation Models

**Linear**: continuous input, continuous output

**Classification**: continuous input, discrete output

**Mixture**: discrete input, continuous output

Code to generate outputs!
Simulation v.s. Model Fitting

\[ y_i = f_\theta(x_i, \epsilon_i) \]

**Simulation**: Observe \( x_i, e_i, \) and theta, **generate** \( y_i \)

**Fitting**: Observe \( x_i, y_i, \) **infer** theta

Why not observe epsilon?
“Generic” term that captures process/observation/modeling error
Example With Linear Model

\[ y_i = A \cdot x_i + \epsilon \]
Example With Linear Model

I get an idealized predictive model: \( y_i = A \cdot x_i \)
Example With Linear Model

\[ y_i = A \cdot x_i + \epsilon \]

```python
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression

>>> X = # temperatures
>>> y = #sales
>>> reg = LinearRegression().fit(X, y)

>>> reg.coef_, reg.intercept_
(array([26.00]), -90.0000...)
>>> reg.predict(np.array([21])) #sales = 26*temp - 90
array([456])
```
How Does It Work?

Least squares principle

\[
\min_{a,c} \sum_{i=1}^{N} (y_i - (ax_i + c))^2
\]

Find parameters that minimize the squared error between prediction and actual.
How Does It Work?

Where does epsilon fit in?

\[
\min_{a,c} \sum_{i=1}^{N} (y_i - (ax_i + c))^2 \quad \epsilon_i = (y_i - (ax_i + c))
\]

\[
\sum_{i=1}^{N} \epsilon_i^2 \approx \text{Var} [\epsilon] + \mathbb{E} [\epsilon]^2
\]

Variation In Error

“Bias” of the model
High Bias

$$\min_{a,c} \text{Var}[\epsilon] + \text{Bias}[\epsilon]^2$$
\[
\min_{a,c} \text{Var}[\epsilon] + \text{Bias}[\epsilon]^2
\]
High Variance

$$\min_{a,c} \text{Var}[\epsilon] + \text{Bias}[\epsilon]^2$$
Low Variance

\[
\min_{a,c} \text{Var}[\epsilon] + \text{Bias}[\epsilon]^2
\]
**Bias-Variance Tradeoff**

**Bias:** How closely does your selected model align with the data.

**Variance:** How different predictions are (in terms of error).
Bias: How closely does your selected model align with the data

Measures “systematic” misprediction
Bias-Variance Tradeoff

**Variance:** How different predictions are (in terms of error).

Measures predictability of error
How Does It Work?

Least squares principle

\[
\min_{a,c} \sum_{i=1}^{N} (y_i - (ax_i + c))^2
\]

Minimize the balance of bias and variance

\[
\min_{a,c} \text{Var}[\epsilon] + \text{Bias}[\epsilon]^2
\]
Problem?

Minimize the balance of bias and variance

$$\min_{a,c} \text{Var}[\epsilon] + \text{Bias}[\epsilon]^2$$

Not robust!!
Problem: Work with finite data

\[
\min_{a,c} \text{Var}[\epsilon] + \text{Bias}[\epsilon]^2
\]
Problem: Work with finite data

\[
\min_{a,c} \text{Var}[\epsilon] + \text{Bias}[\epsilon]^2
\]

Accept some bias if that means much lower variance.
Regularization

Bias that is baked into the optimization problem to avoid being fooled by outliers

\[ \min_{\theta} \sum_{i=1}^{N} (y_i - (ax_i + c))^2 + \lambda \cdot \text{Reg}(a, c) \]

Penalize certain solutions

Lambda is the amount of bias
Types of Models

**sklearn.linear_model.LinearRegression**

\[
\min_{\theta} \sum_{i=1}^{N} (y_i - Ax_i + c)^2
\]

**sklearn.linear_model.Ridge**

\[
\min_{\theta} \sum_{i=1}^{N} (y_i - A \cdot x_i + c)^2 + \lambda \left( \sum_{j=1}^{d} A_{d}^2 + c^2 \right)
\]

**sklearn.linear_model.Lasso**

\[
\min_{\theta} \sum_{i=1}^{N} (y_i - A \cdot x_i + c)^2 + \lambda \left( \sum_{j=1}^{d} |A_{d}| + |c| \right)
\]
A General Formula For Fitting

\[ \min_{\theta} \sum_{i=1}^{N} L(y_i, f_\theta(x_i)) + \lambda g(\theta) \]

\[ \min_{\theta} \sum_{i=1}^{N} (y_i - (ax_i + c))^2 + \lambda \cdot \text{Reg}(a, c) \]
“Loss Functions”

\[
\min_\theta \sum_{i=1}^{N} L(y_i, f_\theta(x_i)) + \lambda g(\theta)
\]

Measure of Success

\[
L(y_i, f_\theta(x_i)) = (y_i - f_\theta(x_i))^2 \quad \text{Sq. error}
\]
\[
L(y_i, f_\theta(x_i)) = |y_i - f_\theta(x_i)| \quad \text{Abs error}
\]
\[
L(y_i, f_\theta(x_i)) = 1(y_i! = f_\theta(x_i)) \quad \text{Exact match only}
\]
“Loss Functions”

\[
\min_{\theta} \sum_{i=1}^{N} L(y_i, f_\theta(x_i)) + \lambda g(\theta)
\]

Measure of Success

Minimizing “average” error (for some definition of error)
“Loss Functions”

\[
\min_\theta \sum_{i=1}^{N} L(y_i, f_\theta(x_i)) + \lambda g(\theta)
\]

Measure of Success

Also relevant to classification problems!

More on this next time...
Quadratic Fit

\[
\min_{a,b,c} \sum_{i=1}^{N} (y_i - (ax_i^2 + bx_i + c))^2
\]
Quadratic Fit

$$\min_{a,b,c} \sum_{i=1}^{N} (y_i - (ax_i^2 + bx_i + c))^2$$

The “class” of models that are considered.
Why Not More?
Bias-Variance Tradeoff

Large classes (more parameters) have **lower bias** but **higher variance**.
Bias-Variance Tradeoff

- Very sensitive to small amount of noise
- Misprediction due to data variation.
Bias-Variance Tradeoff

- Misprediction due to model mismatch
Bias-Variance Tradeoff

- **Error** vs **Model Complexity**
- **Total Error**
  - **Variance**
  - **Bias^2**
- **Optimum Model Complexity**
High Variance = Sensitivity to Unseen Data

Idea: Evaluate models on unseen data to test for overfitting.

Works well on data you have, won’t work for prediction.

Idea: Evaluate models on unseen data to test for overfitting.
Training and Testing

- Original Data
- Randomly Selected Rows
- Training Data
- Testing Data

Use this data for fitting
Use this data for evaluation
Training and Testing

```python
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split

>>> X_train, X_test, y_train, y_test = \
train_test_split(X, y, test_size=0.2)
```

Evaluate memorizing the data in the training set.
Training error: average loss over the training set

Testing error: average loss over the testing set
Training and Testing

![Graph showing the relationship between error and model complexity, showing underfitting, best fit, and overfitting.](image)

- **Error**
- **Model "complexity"**
- **Underfitting**
- **Overfitting**
- **Best Fit**
- **Test Error**
- **Training Error**
Summary

\[
\min_{\theta} \sum_{i=1}^{N} L(y_i, f_\theta(x_i)) + \lambda g(\theta)
\]

Always possible to accurately fit data you already have! (if your model is complex enough…)

Use withheld data (called testing data) to evaluate this effect.