

Big Data for Public Policy

3. Machine Learning Essentials - Regressions

Elliott Ash & Malka Guillot

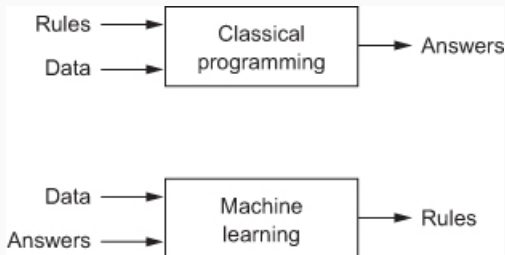
Where we are

- Past weeks:
 - w1: Overview and motivation
 - w2: Finding datasets using webcrawling and API
- This week (w3):
 - Intro to supervised Machine Learning (ML) - regressions
 - Corresponding references: Geron chap 2, chap 3 (pages 111 to 141)
- Next:
 - w4: Text analysis fundamentals
 - w5: Supervised ML - classification
 - w6: Unsupervised ML

Today: supervised ML - regressions

- First hour:
 - What is machine learning?
 - Basic steps and concepts
- Second hour:
 - Application: predicting the prices of houses using given features

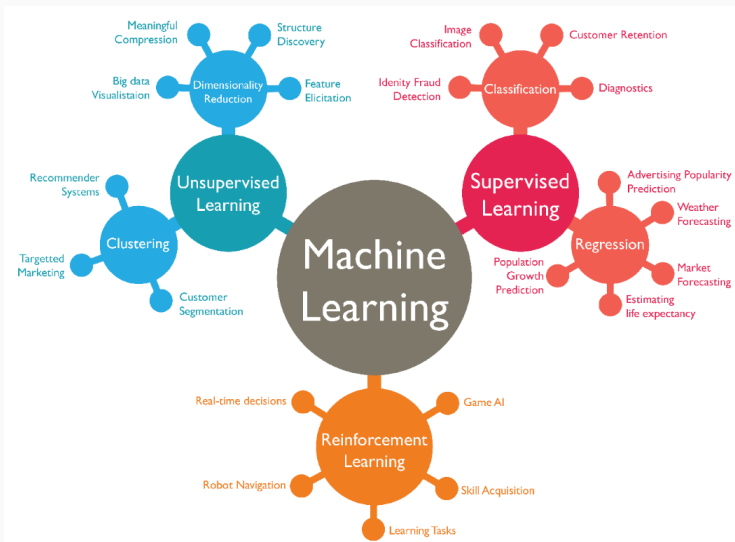
What is machine learning?



- In classical computer programming, humans input the rules and the data, and the computer provides answers.
- In machine learning, humans input the data and the answers, and the computer learns the rules.

⇒ **Machine learning is the science (and art) of programming computers so they can *learn from data*.**

The Machine Learning Landscape



Usually, ML is divided in 2 categories:

- the predictive or supervised learning approach
- the descriptive or unsupervised learning approach

Econometrics vs. Machine Learning

- In **econometrics**, we estimate a low-dimensional **causal parameter** θ .
 - goal of estimation: unbiasedness
 - predicts how outcome y would change if treatment variable x were **exogenously shifted**.
 - useful for policy evaluation.

Econometrics vs. Machine Learning

- In **econometrics**, we estimate a low-dimensional **causal parameter** θ .
 - goal of estimation: unbiasedness
 - predicts how outcome y would change if treatment variable x were **exogenously shifted**.
 - useful for policy evaluation.
- In **machine learning**, we approximate a function $h(X; \theta)$ to predict Y given covariates X .
 - goal of prediction: loss minimization
 - if we collected more data on X , we could predict the associated \hat{Y} .

Econometrics vs. Machine Learning

- In **econometrics**, we estimate a low-dimensional **causal parameter** θ .
 - goal of estimation: unbiasedness
 - predicts how outcome y would change if treatment variable x were **exogenously shifted**.
 - useful for policy evaluation.
- In **machine learning**, we approximate a function $h(X; \theta)$ to predict Y given covariates X .
 - goal of prediction: loss minimization
 - if we collected more data on X , we could predict the associated \hat{Y} .
 - but $h(\cdot)$ does not provide a *counterfactual prediction* – that is, how the outcome would change if X 's were exogenously shifted.

Example of Applications

- Detect fraud: taxes, social benefits
- Forecasts next year's revenue
- Diagnosis of diseases

Machine Learning Overview

Overfitting

Cross-Validation

Regularization

ML in practice

- Suppose we have m observations within a dataset of the form (y_i, x_i) for $i = 1, \dots, m$:
 - y_i : dependent / response / label variable
 - x_i : P -dimensional vector of independent variables, covariates or features. Potentially, $P \gg N$
- **Supervised Learning**: Learn a mapping from x_i to y_i
 - Classification problem: y_i is categorical
 - Regression problem: y_i is continuous
- **Unsupervised Learning**: Learn some structure within the x_i observations

What do ML Algorithms do? Minimize a cost function

- A typical cost function for regression problems is Mean Squared Error (MSE):

$$\text{MSE}(X, h) = \frac{1}{m} \sum_{i=1}^m (h(x_i) - y_i)^2$$

- m , the number of rows/observations
- X , the feature set, with row x_i
- Y , the outcome, with item y_i
- $h(x_i)$ the model prediction (hypothesis)

Linear Regression is Machine Learning

- OLS assumes the functional form

$$y_i = x_i' \theta + \epsilon_i$$

and minimizes the MSE

$$\min_{\hat{\theta}} \frac{1}{m} \sum_{i=1}^m (x_i' \hat{\theta} - y_i)^2$$

Linear Regression is Machine Learning

- OLS assumes the functional form

$$y_i = x_i' \theta + \epsilon_i$$

and minimizes the MSE

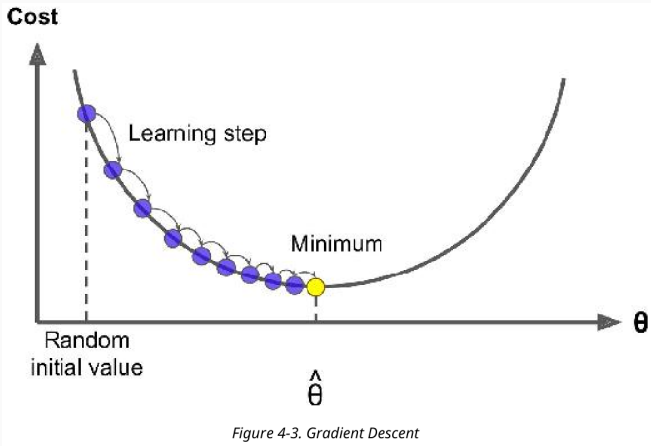
$$\min_{\hat{\theta}} \frac{1}{m} \sum_{i=1}^m (x_i' \hat{\theta} - y_i)^2$$

- This has a closed form solution

$$\hat{\theta} = (\mathbf{X}' \mathbf{X})^{-1} \mathbf{X}' \mathbf{Y}$$

- But
 - this solution does not work well with large m and n
 - most machine learning models do **not** have a closed form solution.

How do ML Algorithms Work? Gradient Descent



- Gradient descent measures the local gradient of the error function, and then steps in that direction.
 - Once the gradient equals zero, you have reached a minimum. 13/36

- Machine learning models can achieve arbitrarily high accuracy in-sample.
 - performance should be evaluated out-of-sample.

- Machine learning models can achieve arbitrarily high accuracy in-sample.
 - performance should be evaluated out-of-sample.
- Standard approach: Train-Set Sample Split
 - randomly sample 80% of the data as a training dataset
 - data transformations, feature selection, and hyperparameter tuning only see this data.

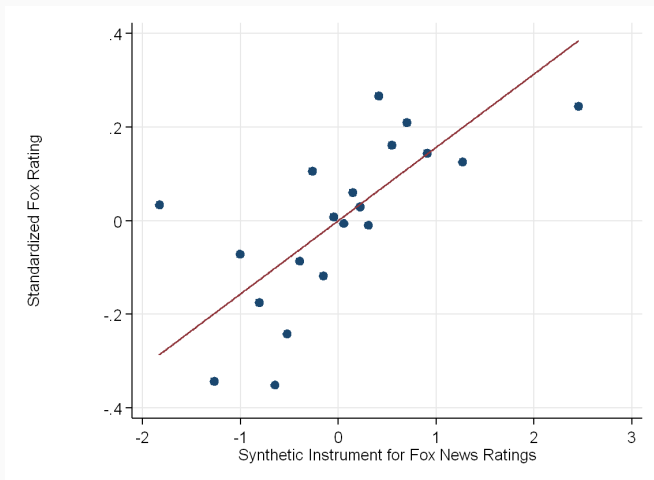
- Machine learning models can achieve arbitrarily high accuracy in-sample.
 - performance should be evaluated out-of-sample.
- Standard approach: Train-Set Sample Split
 - randomly sample 80% of the data as a training dataset
 - data transformations, feature selection, and hyperparameter tuning only see this data.
 - form predictions in 20% test dataset to evaluate performance.

Evaluating Regression Models: R^2

- Mean squared error (MSE) can be used to compare regression models, but the units depend on the outcome variable and therefore are not interpretable.
 - Use R^2 in the test set for a more interpretable evaluation metric.
 - MSE provides same ranking as R^2 and is faster to compute, so it's preferred for model selection.

Evaluating Regression Models: Binscatter Plots

- Binscatters provide visual evidence of regression model performance: Plot Y against \hat{Y} in the test set:



Machine Learning Overview

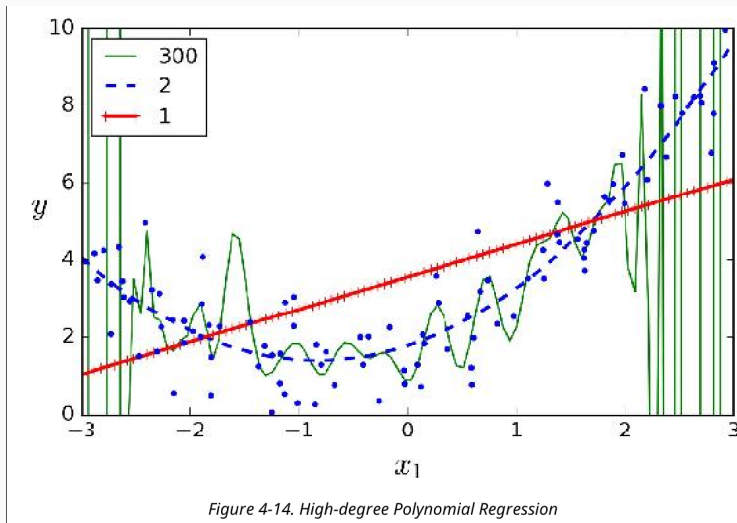
Overfitting

Cross-Validation

Regularization

ML in practice

The Problem of Overfitting



⇒ poor generalization of the model

error of the model = bias + variance + irreducible error

error of the model = bias + variance + irreducible error

- *Bias*
 - Error due to wrong assumptions, such as assuming data is linear when it is quadratic.
 - Will underfit the training data

error of the model = bias + variance + irreducible error

- *Bias*
 - Error due to wrong assumptions, such as assuming data is linear when it is quadratic.
 - Will underfit the training data
- *Variance*
 - Error due to excess sensitivity to small variations in the training data.
 - A model with high variance is likely to overfit the training data.

error of the model = bias + variance + irreducible error

- *Bias*
 - Error due to wrong assumptions, such as assuming data is linear when it is quadratic.
 - Will underfit the training data
- *Variance*
 - Error due to excess sensitivity to small variations in the training data.
 - A model with high variance is likely to overfit the training data.
- *Irreducible error*
 - Error due to noise in the data.

- In general, increasing a model's complexity will increase variance and reduce bias.

- In general, increasing a model's complexity will increase variance and reduce bias.
- If the model is underfitting:
 - adding more training data will not help.
 - use a more complex model

- In general, increasing a model's complexity will increase variance and reduce bias.
- If the model is underfitting:
 - adding more training data will not help.
 - use a more complex model
- If the model is overfitting:
 - adding more training data may help
 - or use regularization
 - cross-validation

Machine Learning Overview

Overfitting

Cross-Validation

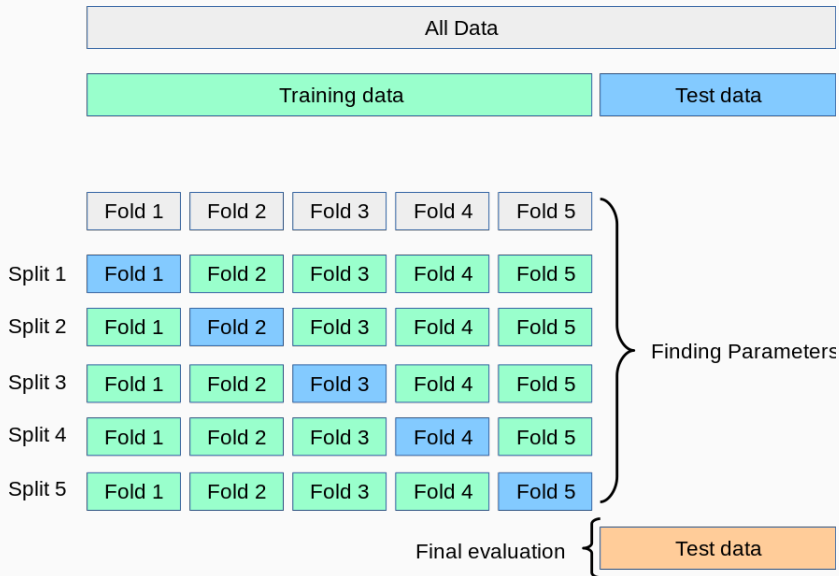
Regularization

ML in practice

- looks at how the distribution of the model's error
- Use `cross_val_score` method to get model performance across subsets of the training set:
 - split data into K folds.
 - for each fold $k \in \{1, 2, \dots, K\}$, train model in rest of data ($-k$) and evaluate MSE in k .
 - Report mean and s.d. of MSE across folds.

- looks at how the distribution of the model's error
- Use `cross_val_score` method to get model performance across subsets of the training set:
 - split data into K folds.
 - for each fold $k \in \{1, 2, \dots, K\}$, train model in rest of data ($-k$) and evaluate MSE in k .
 - Report mean and s.d. of MSE across folds.

Cross-Validation



Machine Learning Overview

Overfitting

Cross-Validation

Regularization

ML in practice

Ridge, Lasso, and Elastic Net

- Ridge and lasso regression are tools for dealing with large feature sets where:
 - models have multicollinearity that causes bias
 - models tend to overfit
 - models are computationally costly to fit
- These algorithms work by constraining estimated parameter sizes.

Ridge Regression

- The Ridge cost function is

$$J(\theta) = \text{MSE}(\theta) + \underbrace{\alpha_2 \frac{1}{2} \sum_{i=1}^n \theta_i^2}_{\text{Regularization term}}$$

- i indexes over n features
- α_2 is a **hyperparameter** setting the strength of the L2 penalty

Ridge Regression

- The Ridge cost function is

$$J(\theta) = \text{MSE}(\theta) + \underbrace{\alpha_2 \frac{1}{2} \sum_{i=1}^n \theta_i^2}_{\text{Regularization term}}$$

- i indexes over n features
 - α_2 is a **hyperparameter** setting the strength of the L2 penalty
- Ridge penalizes large coefficients, which reduces over-fitting to the training set.
 - The estimated coefficients, when taken to other data, will generalize better.

Ridge Regression

- The Ridge cost function is

$$J(\theta) = \text{MSE}(\theta) + \underbrace{\alpha_2 \frac{1}{2} \sum_{i=1}^n \theta_i^2}_{\text{Regularization term}}$$

- i indexes over n features
 - α_2 is a **hyperparameter** setting the strength of the L2 penalty
- Ridge penalizes large coefficients, which reduces over-fitting to the training set.
 - The estimated coefficients, when taken to other data, will generalize better.
- It turns out that the Ridge estimator, like OLS, has a closed-form solution:

$$\hat{\theta}_{\text{Ridge}} = (X'X + \alpha_2 I_n)^{-1} X' y$$

where I_n is the identity matrix.

- But it can also be solved by (stochastic) gradient descent

Lasso Regression

- *Least Absolute Shrinkage and Selection Operator Regression*
- The Lasso cost function is

$$J(\theta) = \text{MSE}(\theta) + \alpha_1 \sum_{i=1}^n |\theta_i|$$

- i indexes over n features
- α_1 is a hyperparameter setting the strength of the L1 penalty

Lasso Regression

- *Least Absolute Shrinkage and Selection Operator Regression*
- The Lasso cost function is

$$J(\theta) = \text{MSE}(\theta) + \alpha_1 \sum_{i=1}^n |\theta_i|$$

- i indexes over n features
 - α_1 is a hyperparameter setting the strength of the L1 penalty
- Lasso automatically performs feature selection and outputs a sparse model.
- It does not have a closed-form solution but can be solved by gradient descent.

- Elastic Net uses both L1 and L2 penalties:

$$J(\theta) = \text{MSE}(\theta) + \alpha_1 \sum_{i=1}^n |\theta_i| + \alpha_2 \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

- in general, elastic net is preferred to lasso, which can behave erratically when the number of features is greater than the number of rows, or when some features are highly collinear.
 - but you have to tune two hyperparameters rather than one

Hyperparameters vs. Parameters

- **Parameters:** *internal* to the model whose values can be estimated from the data and we are often trying to estimate them as best as possible
 - **hyperparameters:** *external* to the model and cannot be directly learned from the regular training process
- ⇒ model-specific properties that are *fixed* before the model is trained

Hyperparameters tuning

- Use `GridSearchCV` or `RandomizedSearchCV` to automate search over parameter space.
 - For example: Elastic net hyperparameters should be selected to optimize out-of-sample fit.
 - “Grid search” scans over the hyperparameter space $(\alpha_1 \geq 0, \alpha_2 \geq 0)$, computes out-of-sample MSE for all pairs (α_1, α_2) , and selects the MSE-minimizing model.

Regularized Models Require Standardized Data

- Regularized models are designed to work with standardized predictors:

$$\tilde{x}_i = \frac{x_i - \bar{x}}{SD[\mathbf{x}]}$$

Regularized Models Require Standardized Data

- Regularized models are designed to work with standardized predictors:

$$\tilde{x}_i = \frac{x_i - \bar{x}}{SD[\mathbf{x}]}$$

- Taking out the mean will convert sparse data to dense data, you can avoid that by just dividing by the standard deviation:

$$\tilde{x}_i = \frac{x_i}{SD[\mathbf{x}]}$$

- in sklearn, set `with_mean=False`.

Machine Learning Overview

Overfitting

Cross-Validation

Regularization

ML in practice

A Machine Learning Project, End-to-End

Aurelien Geron, *Hands-on machine learning with Scikit-Learn, Keras, & TensorFlow*, Chapter 2:

1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.

A Machine Learning Project, End-to-End

Aurelien Geron, *Hands-on machine learning with Scikit-Learn, Keras, & TensorFlow*, Chapter 2:

1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model:
 - 6.1 Testing and validating
 - 6.2 Hyperparameter tuning and model selection.
7. Present your solution.
8. Launch, monitor, and maintain your system.

- Python packages
 - pandas for handling tabular data
 - matplotlib and seaborn for plotting
 - Scikit-Learn (sklearn) for standard (non-deep-learning) ML algorithm
 - scikit-learn is very comprehensive and the online-documentation itself provides a good introduction into ML.
 - TensorFlow, PyTorch and Keras for deep-learning.

- See Geron Chapter 2 for pandas and sklearn syntax:
 - imputing missing values.
 - feature scaling (often helpful/necessary for ML models to work well)
 - encoding categorical variables.
 - see jupyter notebook
- Best practice: **reproducible data pipeline.**

- See Geron Chapter 2 for pandas and sklearn syntax:
 - imputing missing values.
 - feature scaling (often helpful/necessary for ML models to work well)
 - encoding categorical variables.
 - see jupyter notebook
- Best practice: **reproducible data pipeline**.
- **Key point**: all data transformations, feature selection, and hyperparameter tuning must be done **in the training set**.

- **Consistency:**

- *Estimator*: An object that can estimate parameters. Estimation is performed by `fit()` method. Exogenous parameters (provided by the researcher) are called *hyperparameters*.

- **Consistency:**

- *Estimator*: An object that can estimate parameters. Estimation is performed by `fit()` method. Exogenous parameters (provided by the researcher) are called *hyperparameters*.
- *Transformer*: An object that transforms a data set. Transformation is performed by the `transform()` method. The convenience method `fit_transform()` both fits an estimator and returns the transformed input data set.

- **Consistency:**

- *Estimator*: An object that can estimate parameters. Estimation is performed by `fit()` method. Exogenous parameters (provided by the researcher) are called *hyperparameters*.
- *Transformer*: An object that transforms a data set. Transformation is performed by the `transform()` method. The convenience method `fit_transform()` both fits an estimator and returns the transformed input data set.
- *Predictor*: An object that forms a prediction from an input data set. The `predict()` method forms the predictions. It also has a `score()` method that measures the quality of the predictions given a test set.

- **Consistency:**

- *Estimator*: An object that can estimate parameters. Estimation is performed by `fit()` method. Exogenous parameters (provided by the researcher) are called *hyperparameters*.
- *Transformer*: An object that transforms a data set. Transformation is performed by the `transform()` method. The convenience method `fit_transform()` both fits an estimator and returns the transformed input data set.
- *Predictor*: An object that forms a prediction from an input data set. The `predict()` method forms the predictions. It also has a `score()` method that measures the quality of the predictions given a test set.

- **Inspection**: Hyperparameters and parameters are accessible. Learned parameters have an underscore suffix (e.g. `lin_reg.coef_`)

Scikit-Learn Design Principles

- **Consistency:**

- *Estimator*: An object that can estimate parameters. Estimation is performed by `fit()` method. Exogenous parameters (provided by the researcher) are called *hyperparameters*.
- *Transformer*: An object that transforms a data set. Transformation is performed by the `transform()` method. The convenience method `fit_transform()` both fits an estimator and returns the transformed input data set.
- *Predictor*: An object that forms a prediction from an input data set. The `predict()` method forms the predictions. It also has a `score()` method that measures the quality of the predictions given a test set.

- **Inspection**: Hyperparameters and parameters are accessible. Learned parameters have an underscore suffix (e.g. `lin_reg.coef_`)

- **Non-proliferation of classes**: Use native Python data types; ^{36/36}

Scikit-Learn Design Principles

- **Consistency:**

- *Estimator*: An object that can estimate parameters. Estimation is performed by `fit()` method. Exogenous parameters (provided by the researcher) are called *hyperparameters*.
- *Transformer*: An object that transforms a data set. Transformation is performed by the `transform()` method. The convenience method `fit_transform()` both fits an estimator and returns the transformed input data set.
- *Predictor*: An object that forms a prediction from an input data set. The `predict()` method forms the predictions. It also has a `score()` method that measures the quality of the predictions given a test set.

- **Inspection**: Hyperparameters and parameters are accessible. Learned parameters have an underscore suffix (e.g. `lin_reg.coef_`)

- **Non-proliferation of classes**: Use native Python data types; ^{36/36}