CSE514 Fall 2022 Data Mining
Regression

Supervised learning is a class of learning methods that matches how we generally think us humans learn: from positive and negative examples.

- The problem formally: Given *n* data points
  $$\{(x_1, y_1), (x_2, y_2), \dots , (x_n, y_n)\}$$
  where x_i is the *i^th^* data point's independent (or predictor) variables, and y_i is the dependent (or response) variable, learn a model that can estimate y using x
- Training data:
  - Data points where the response variable values are known
  - Are used to train the model
- Testing/validation data:
  - Data points where the response variable values are **not** known
  - Are **not** used to train the model. Instead, the trained model is used to predict the response variables of testing data, and the accuracy of these predictions is used to evaluate the model

Regression is one of the most commonly used supervised learning methods.

- The task: Fit a function $y = f(x)$ to a given set of training data, where y has continuous, numerical values
- Why?
  1. To reverse engineer the unknown process that generated the training data
  2. To use the function for predicting unknown y from new x
- This problem has two parts:
  1. Estimate the class of functions $f(\cdot)$ belongs to, e.g. linear, polynomial, exponential. This is hard, and is usually solved by having a human expert guess
  2. Estimate the parameters for the chosen form of $f(\cdot)$, e.g. if $f(x) = mx + b$, we need to estimate parameters m and b to fit the training data

Note: Since regression is usually applied when the response variable values are continuous quantities, the class of *problems* that involve predicting or estimating continuous, rank-able numbers, is known as regression problems. This is in contrast to **classification** problems, which involve predicting or estimating discrete, un-ordered class labels.

Example **function classes**:

Linear	$f(x) = mx + b$
Polynomial	$f(x) = m_n x^n + m_{n-1} x^{n-1} + \cdots + m_2 x^2 + m_1 x^1 + m_0$
Exponential	$f(x) = m(k)^x$
Logarithmic	$f(x) = log_k(x)$
Logistic	$f(x) = \frac{1}{1+e^{-k(x-x_0)}}$

"Fit the training data"

- Minimize the difference between observed values and the function's estimated values. The function that defines this difference is called a loss function or an objective function. For example:

Total Absolute Error:

$$\sum_{i=1}^{n} |y_i - f(x_i)|$$

Total Squared Error:

$$\sum_{i=1}^{n} (\, y_i - f(x_i)\,)^2$$

- **Gradient descent:** a technique used to minimize an objective function

  Ex. For a univariate linear function:

  $$f(x) = mx + b$$

  And a chosen objective function of Mean Squared Error (MSE):

  $$L(m, b) = \frac{1}{n}\sum_{i=1}^{n} (\, y_i - (mx_i + b)\,)^2$$

  The gradient of the objective function with respect to parameter m:

  $$\frac{\partial L}{\partial m} = \frac{1}{n}\sum_{i=1}^{n} -2x_i(\, y_i - (mx_i + b)\,)$$

  Parameters are updated iteratively, using the gradient to direct the change towards minimization of the objective function.

Step size: Uses a hyperparameter called the learning rate, usually symbolized with α, that controls how far along the gradient to update parameter values at each iteration.

A step in our example for updating parameter m:

$$m_{new} = m_{old} - \frac{\alpha}{n}\sum_{i=1}^{n} -2x_i(\, y_i - (m_{old}x_i + b_{old})\,)$$

Stop conditions: when to end the search

1. A maximum number of steps to search
   - A failsafe to guarantee the algorithm will stop searching
2. When further improvement in the objective function is not needed
   - This is some threshold where the objective function is close enough to the theoretical minimum
   - **OR** some threshold when the change in the objective function between steps is considered negligeable

Stochastic gradient descent (SGD): Instead of calculating the partial derivatives for every sample in the dataset for every iteration/step of gradient descent, randomly select a single sample or a subset of samples (batch GD) at each iteration. Larger subsets generally have more robust performance, while smaller subsets will step faster.

Overfitting: This refers to the scenario where a model fits well with training data, but poorly to new data. This means the model cannot generalize beyond the scope of the training data, limiting its utility. This problem can be detected by comparing the model's performance on seen and unseen data, such as through cross validation.

Cross-validation is one method for checking if your model is overfitting.

1. Split the given data into subsets
2. Train the model on all but one subset
3. Test the model on the held-out data
4. Repeat with all subsets

The average performance on the held-out data is used to evaluate if the model can generalize to unseen data.
***k*-fold CV** refers to the practice of splitting the data into *k* subsets for cross validation

Regularization is the practice of adding a penalty term to an objective function, typically chosen to penalize model complexity under the assumption that less complex models overfit less and generalize better. A lambda hyperparameter can also be trained to scale how much influence this penalty should have. Some example penalty terms:

- L1 regularization used for LASSO regression
    - The penalty added is the L1-norm of the vector of parameters.
- L2 regularization used for Ridge regression
    - The penalty added is the L2-norm of the vector of parameters.