3 Fundamental Algorithms (part 1 of 5)

Here are five popular supervised learning algorithms trained on \( \{(x_i, y_i)\}_{i=1}^N \) (where \( x = [x_1, \ldots, x_D] \)), each useful directly or as a part of a more complex algorithm.

**Linear regression**

- *Linear regression* models \( y \in \mathbb{R} \) as a linear combination of the features in \( x \): \( \hat{y} \leftarrow f_{w,b}(x) = wx + b \). Draw the case where \( D = 1 \), that is, \( x \) is 1D:
  
  - \((x, y)\): \( i^{th} \) example
  - \( \hat{y} = wx + b \): regression line
    * \( w \): slope
    * \( b \): intercept
  - \( \hat{y}_i = wx_i + b \): predicted \( y \) for \( x = x_i \)
  - \( \hat{y}_i - y_i \): vertical difference

Now we consider any \( D \geq 1 \) (simple linear regression or multiple linear regression).

- Find optimal \([w^*, b^*]\) that minimize the *objective function* \( \frac{1}{N} \sum_{i=1}^{N} [f_{w,b}(x_i) - y_i]^2 \) (also called the *mean squared error* (MSE)). Alternatives to MSE include:
  - mean error: lousy, as any line through centroid of two points works equally well
  - mean absolute error, \( \frac{1}{N} \sum_{i=1}^{N} |f_{w,b}(x_i) - y_i| \): useful, but discontinuous derivative

We predict \( y \) from \( x \), so minimize vertical difference in the “least squares” sense.

- \([f_{w,b}(x_i) - y_i]^2 \), a *loss function*, penalizes poor estimation of example \( i \). We minimize a *cost function* given by the average loss (MSE, above) of all penalties from applying the model to the training data.

- Minimize MSE by setting gradient (vector of partial derivatives) to zero. Use matrix notation:

\[
X = \begin{bmatrix}
1 & x_1 \\
\vdots & \vdots \\
1 & x_i \\
\vdots & \vdots \\
1 & x_N
\end{bmatrix}
= \begin{bmatrix}
x_1^{(1)} & \ldots & x_j^{(1)} & \ldots & x_D^{(1)} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
x_1^{(i)} & \ldots & x_j^{(i)} & \ldots & x_D^{(i)} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
x_1^{(N)} & \ldots & x_j^{(N)} & \ldots & x_D^{(N)}
\end{bmatrix} = \begin{bmatrix}
x_{10} & x_{11} & \ldots & x_{1j} & \ldots & x_{1D} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_{i0} & x_{i1} & \ldots & x_{ij} & \ldots & x_{iD} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_{N0} & x_{N1} & \ldots & x_{Nj} & \ldots & x_{ND}
\end{bmatrix}_{N \times (1+D)}
\]
This design matrix $X$ is the training examples (without their labels \{\{y_i\}\}) preceded by a column of ones for convenience: we understand $x_{i0} \equiv 1$ for $i = 1, \ldots, N$.

$$- \mathbf{w} = \begin{bmatrix} w_0 = b \\ w_1 \\ \vdots \\ w_i \\ \vdots \\ w_D \end{bmatrix}_{(1+D) \times 1} \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_N \end{bmatrix}_{N \times 1}, \quad \text{and} \quad \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_i \\ \vdots \\ \hat{y}_N \end{bmatrix}_{N \times 1}$$

The system of equations, $\hat{y}_i = \mathbf{w}_i x_i + b$ for $i = 1, \ldots, N$ is expressed as $\hat{y} = X \mathbf{w}$. We minimize

$$MSE = \frac{1}{N} \sum_{i=1}^{N} [f_{\mathbf{w},b}(\mathbf{x}_i) - y_i]^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} [\hat{y}_i - y_i]^2 \quad \left(\text{which is also equal to} \quad \frac{1}{N} ||\hat{y} - \mathbf{y}||^2 = \frac{1}{N} ||X \mathbf{w} - \mathbf{y}||^2\right)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \left( y_i - \sum_{j=0}^{D} x_{ij} w_j \right)^2$$

by setting to zero the partial derivative with respect to $w_k$ for each $k = 0, \ldots, D$:

$$\frac{\partial}{\partial w_k} (MSE) = \frac{1}{N} \sum_{i=1}^{N} 2 \left( y_i - \sum_{j=0}^{D} w_j x_{ij} \right) (-x_{ik})$$

$$= 0 \quad \Rightarrow \quad \sum_{i=1}^{N} y_i x_{ik} = \sum_{i=1}^{N} \sum_{j=0}^{D} w_j x_{ij} x_{ik}$$

$$\Rightarrow \quad \sum_{i=1}^{N} X^T_{ki} y_i = \sum_{j=0}^{D} \left( \sum_{i=1}^{N} X^T_{ki} x_{ij} \right) w_j$$

$$\Rightarrow \quad [X^T \mathbf{y}]_k = \sum_{j=0}^{D} (X^T X)_{kj} w_j$$

$$= [(X^T X) \mathbf{w}]_k$$

\[ \text{\footnote{This is an abuse of notation: I moved } b \text{ into } \mathbf{w} \text{ as } w_0 \text{ and added an } x_{i0}^{(i)} = 1 \text{ element to each } \mathbf{x} \text{ feature vector.}} \]
This is true for each $k$, so we can write $X^T y = (X^T X) w$. To solve for $w$, multiply both sides on the left by $(X^T X)^{-1}$: 

$$w = (X^T X)^{-1} X^T y.$$

e.g. Use $w = (X^T X)^{-1} X^T y$ to find the regression line for the points (1, 1), (2, 3) (3, 2).

- Overfitting occurs when a model predicts training example labels (real numbers here) well but not unseen example labels.
  
e.g. Consider fitting successively higher-degree polynomials to a small 2D scatterplot.

- We found a closed-form solution to this minimization problem. \footnote{Gradient descent (coming in §4) is a numerical method that can minimize some cost functions where no closed-form solution is known.} Gradient descent (coming in §4) is a numerical method that can minimize some cost functions where no closed-form solution is known.

\footnote{What did one regression coefficient say to the other?}{\footnote{Scikit-learn does not invert the matrix, instead using a faster, more stable algorithm that solves $X^T y = (X^T X) w$.}}
Python

- `from sklearn import linear_model` loads the `linear_model` module
- `model = linear_model.LinearRegression()` gives the model
- `model.fit(X, y)` fits the model to array $X_{N \times D}$ and $y_{N \times 1}$
- `model.coef_` gives $w^*$ and `model.intercept_` gives $b^*$
- `model.predict(X)` gives predictions for examples in $X$
- `model.score(X, y)` gives $R^2$, the coefficient of determination from statistics, that is the proportion (in $[0, 1]$) of variability in $y$ accounted for by $X$ via the linear model.

To learn more: