CSC 461: Machine Learning

Fall 2024

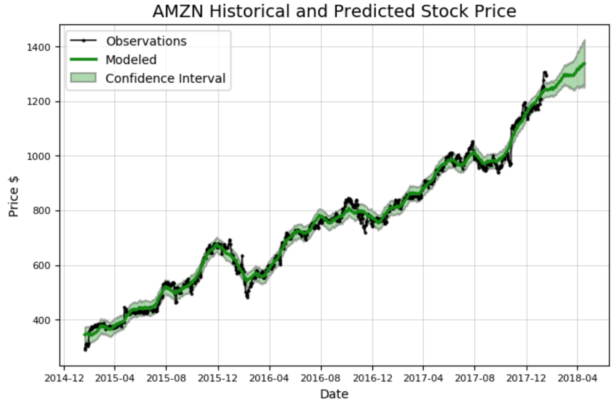
Linear regression

Prof. Marco Alvarez, Computer Science
University of Rhode Island

Preliminaries

Continuous output

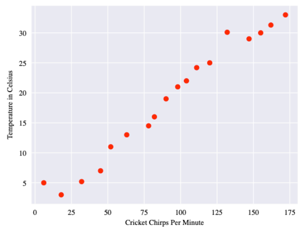
- Certain applications require the prediction of continuous values

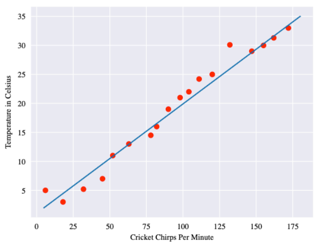


https://towardsdatascience.com/stock-prediction-in-python-b66555171a2

Linear functions

- Assumes the output y is a linear function of the input $\mathbf{x}$
  - can use the function to make predictions, very simple approach, e.g. linear regression

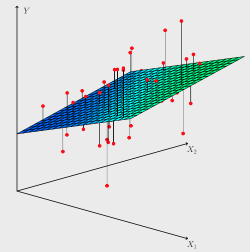




$$\underbrace{y = wx}_{\text{slope}} + \underbrace{b}_{\text{intercept}} \quad w, x, b \in \mathbb{R}$$

Linear functions

- What if we have d features?
  - $\mathbf{w}, \mathbf{x} \in \mathbb{R}^d$
  - $b \in \mathbb{R}$



$y = w_2 x_2 + w_1 x_1 + b$

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

hypothesis weights bias

The weights and bias are the model **parameters** which define the hypothesis and are used to make predictions

Alternative notation

- Augmented vectors
  - incorporate bias into the weight vector
    - $\mathbf{w} = [b, w_1, w_2, \ldots, w_d]^T$
  - augment input vector with 1
    - $\mathbf{x} = [1, x_1, x_2, \ldots, x_d]^T$
  - simplified equation
    - $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

X_1	X_2	Y
0.5	0.1	0.25
0.3	0.9	0.5
0.3	0.875	1.15
0.45	0.15	2.13
…	…	…

X_0	X_1	X_2	Y
1	0.5	0.1	0.25
1	0.3	0.9	0.5
1	0.3	0.875	1.15
1	0.45	0.15	2.13
…	…	…	…

Hypothesis space $\mathscr{H} = \left\{h_w : h_w(\mathbf{x}) = \mathbf{w}^T\mathbf{x},\ \mathbf{w} \in \mathbb{R}^{d+1}\right\}$

Linear regression

Linear regression

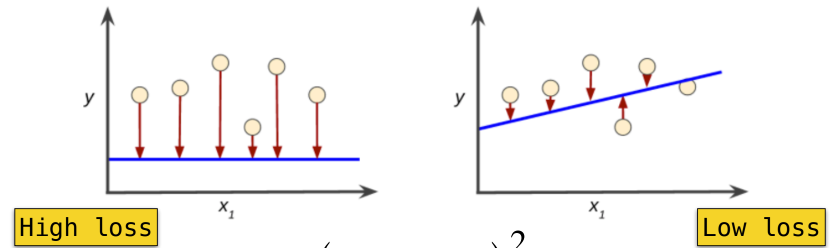
- Fundamental supervised learning algorithm
  - used for predicting continuous target variables
  - assumes a linear relationship between input features and the target
- Applications
  - financial forecasting (e.g., stock prices, economic indicators)
  - environmental science (e.g., climate change predictions)
  - marketing (e.g., sales forecasting)
  - real estate (e.g., house price prediction)

Linear regression

- Data
  - $\mathcal{D} = \left\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\right\}$
  - $\mathbf{x}^{(i)} \in \mathbb{R}^{d+1}, y^{(i)} \in \mathbb{R}$
- Model
  - $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$
- Squared Loss (L2)
  - $l_{sq}(h, \mathbf{x}, y) = \left(h(\mathbf{x}) - y\right)^2$
  - $L(h, \mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} l_{sq}(h, \mathbf{x}^{(i)}, y^{(i)})$

Residuals and MSE loss

- Residual
  - difference between predicted and actual value

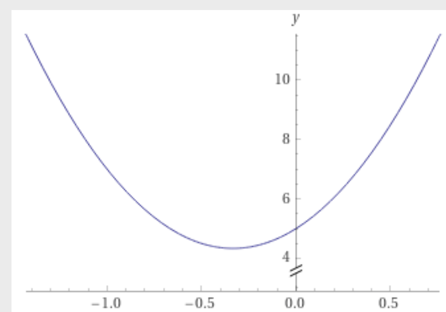


$$\left(\underbrace{h(\mathbf{x})}_{\text{predicted}} - \underbrace{y}_{\text{actual}}\right)^2$$

Want to find a **linear function** with **small residuals**

Closed-form solution

min vs. argmin



$f(x) = 6x^2 + 4x + 5$

$\min\ f(x)?$

$\arg\min_x f(x)?$

Given a function, it represents the input value(s) that produce the function's minimum output value

Normal equations

- Analytical solution to **minimize the loss function**

$$\mathbf{w}^* = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2$$

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Using matrix notation

$$\mathbf{y} = \mathbf{X}\mathbf{w} \qquad \begin{bmatrix} \left(\mathbf{x}^{(1)}\right)^T \\ \vdots \\ \left(\mathbf{x}^{(n)}\right)^T \end{bmatrix} \begin{bmatrix} w_0 \\ \vdots \\ w_d \end{bmatrix} \approx \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Using matrix notation

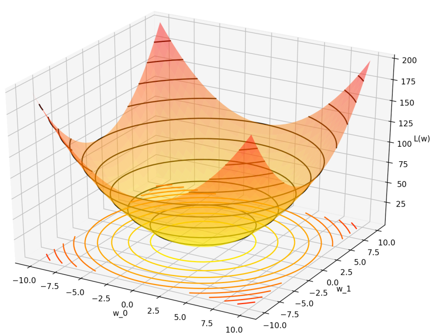
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Minimizing the loss function

- Set the gradient to zero
  - solve for $\mathbf{w}$

continuous, differentiable, **convex**



$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Minimizing the loss function

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2$$

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N}\mathrm{X}^{\intercal}(\mathrm{X}\mathbf{w} - \mathbf{y}) = \mathbf{0}$$

$$\mathrm{X}^{\intercal}\mathrm{X}\mathbf{w} = \mathrm{X}^{\intercal}\mathbf{y}$$

$$\mathbf{w} = \mathrm{X}^{\dagger}\mathbf{y} \quad \text{where} \quad \mathrm{X}^{\dagger} = (\mathrm{X}^{\intercal}\mathrm{X})^{-1}\mathrm{X}^{\intercal}$$

$\mathrm{X}^{\dagger}$ is the 'pseudo-inverse' of X

There are other methods for finding the optimal solution e.g. gradient descent, MLE

http://work.caltech.edu/slides/slides03.pdf

Show me the code

```
# generate random data
Xtr = np.hstack((np.ones((100, 1)), np.random.rand(100, 4)))
Ytr = np.random.rand(100, 1)
Xte = np.hstack((np.ones((10, 1)), np.random.rand(10, 4)))
Yte = np.random.rand(10, 1)

        w = np.linalg.pinv(Xtr) @ Ytr
        pred = Xte @ w
                                          or

        np.linalg.inv(Xtr.T @ Xtr) @ Xtr.T @ Ytr
        pred = Xte @ w
                                          or

        reg = LinearRegression().fit(Xtr, Ytr)
        pred = reg.predict(Xte)


loss = np.mean((pred-Yte) ** 2)
print(loss)
```

Conclusion

- Linear regression: simple yet powerful
  - foundation for many advanced ML techniques
  - serves as a baseline for more complex models
- Computational Complexity
  - becomes inefficient for large datasets or high-dimensional data
  - time complexity: $O(nd^2 + d^3)$
- Alternatives
  - mini-batch gradient descent
  - regularization techniques (Ridge, Lasso)
- Limitations and Considerations
  - assumes linearity between features and target
  - sensitive to outliers