Chapter 4: Training models

- Intro: Closed form vs gradient descent
- Linear regression
  - Intro
    - Equation is weighted sum of input features + bias term
    - ŷ = θ * x, theta * x is dot product, Σ θ * x
    - Find value of θ that minimizes error (mean squared error in practice)
  - The Normal Equation
    - $\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \mathbf{y}$
    - Inverse of matrix squared times transpose of matrix
    - Singular Value Decomposition: training set decomposed into 3 matrices that create pseudoinverse
      - More efficient to compute than Normal Equation
      - Pseudo is always invertible, not always case for Normal
  - Computation complexity
    - Matrix inversion complexity $O(n^{2.4})$ to $O(n^3)$
      - Double features, quadruple computation time
    - But once linear regression model is trained, predictions are fast
- Gradient descent
  - Intro
    - Good for large number of features or too many training instances for memory
    - Tweak parameters iteratively to minimize cost function
    - Measure gradient of error wrt to parameter vector (e.g. slope) and move in that direction until reached a minimum
    - Step size determined by learning rate; small: many iterations, long time to converge; high: might diverge
    - Cost functions have different shapes:
      - Leading to local as opposed to global minima
      - MSE: parabola because quadratic
        - Different feature scales → longer time to convergence
  - Batch
    - Compute gradient of cost function wrt to each parameter: partial derivative
      - Can be done all in one go, yielding gradient vector
    - Once have gradient vector, move in opposite direction
    - Parameter – (learning rate * gradient vector)
    - Can use grid search to find good learning rate
    - Set iteration number: large, but interrupt when gradient vector becomes tiny → tolerance
    - Convergence rate: for convex cost function without abrupt slope changes
      - O(1/tolerance) iterations to reach with range of tolerance
  - Stochastic
    - Batch is slow since uses whole training set

        - Stochastic uses random instances
        - Continues to bounce around. Good, not optimal parameter values
        - Works well for irregular cost functions, but my still never settle:
            - Gradually reduce learning rate: learning schedule
        - Training instances need to be independent and identically distributed to ensure get near global optimum
            - Shuffle training set at beginning of each epoch
    - Mini-batch
        - Compute gradients on random mini-batches
        - Get performance boost
        - Less erratic, but continues to bounce around minimum
- Polynomial regression
    - Beware of combinatorial explosion: higher degree polynomials
        - N features, d degrees, (n+d)!/n!d! array
- Learning curves
    - High degree polynomials → severe overfitting
    - Use cross-validation to assess model generalizability
        - Performs well on all data, poorly on CV → overfitting
        - Performs poorly on both: underfitting
    - Learning curves help assess too
        - If error rates plateau, typically underfitting
        - Gap between training & validation → overfitting
    - Bias/Variance Trade-off
        - Generalization error expressed as sum of bias, variance, and irreducible
        - Bias: wrong assumptions, underfitting
        - Variance: high sensitivity to small variations, overfitting
        - Irreducible error: noisiness in data
        - Complexity ↑, Bias ↓, Variance ↑ and vice versa
- Regularized linear models
    - Reduce overfitting, reduce degrees of freedom
    - Ridge
        - L2 norm
        - Background
            - Add regularization term to cost function
            - Fit data, but keep model weights small too
            - Only add regularization term during training
            - Cost function and performance measures often different
        - Alpha hyperparameter controls regularization.
            - Alpha = 0 → linear regression, alpha is large → flat line at mean
    - Lasso
        - L1 norm
        - Eliminates weights of least important features; sets them to zero
        - Automatic feature selection, output is sparse model (
    - Elastic Net

        - Combination of Ridge and Lasso
        - Mix ratio r controls how much of each
        - Decision on which to use
            - Avoid Linear, Ridge good default
            - Believe few useful features → choose Lasso/Elastic Net
            - Elastic Net over Lasso only b/c Lasso might be erratic when # of features > # of labels; or features strongly correlated
    - Early Stopping
        - Stop when validation error reaches minimum
        - SGD and mini-batch curves not smooth, hard to know if reached minimum
            - Stop after validation error been above minimum for some time and then rollback to min
- Logistic regression
    - Background
        - Estimate probabilities: ρ > 0.5, class = 1
    - Estimating probabilities
        - Sigmoid function: $1/(1+\ e^{-t})$
        - Log odds: log(p/(1-p))
    - Training and cost function
        - Estimate high probabilities for positive, low for negative instances
        - -log($\hat{p}$) for y = 1 and -log(1-$\hat{p}$) for y = 0
        - ↑ when $\hat{p}$ gets close to zero, so cost ↑
        - Log-loss for whole training set:
            - −(ylog(p)+(1−y)log(1−p))
        - No closed form solution → Gradient Descent to the rescue!
        - Calculate partial derivatives wrt each model parameter
        - Compute prediction error, multiply by feature value, take average
        - Gradient vector of partial derivatives → use in Batch Gradient
    - Decision Boundaries
        - Where the logistic regression model separates positive vs. negative
        - Boundary is usually at the 50% mark, but can find wrt to independent variable data
            - EG, 1.6 cm when using petal width to predict Iris Virginica
        - Boundary is a line
    - Softmax regression
        - Generalization of logistic regression to handle multiple classes
        - Computes score for each class, then estimates probability of each class
        - Take exponent of each score, and then normalize by sum of exponentiated scores:
            - $\exp\left(s_k(x)\right)/\Sigma\exp\left(s_k(x)\right)$
        - Predicts most likely class based on highest score
        - Predicts only one class at a time; cannot be used for multioutput → photos of many people
        - Cross entropy used to measure how well model matches targets

- Cross entropy gradient vector for class k

$$1/m \sum_{i=0}^{m} (\hat{p}_k^{(i)} - y_k^{(i)})(x)^{(i)}$$