Regression and Miscellaneous Topics

- In **regression**, we want to map inputs to continuous outputs (as opposed to classification, where we had categorical outputs).

- In k**-nearest-neighbor** regression, like k-nearest-neighbor classification, we find the k nearest training points to an input, but instead of taking a majority vote as our prediction, we take the average the neighbors' output values.

- **Locally-weighted averaging** is similar to k-nearest-neighbors, but instead of using the k nearest points, we use all points within some distance λ, and compute their average weighted by (some function of) their distances. The output we predict for an input $\mathbf{x}$ will be

  $$\frac{\sum_{i=1}^{k} K(\mathbf{x}, \mathbf{x}_i) y_i}{\sum_{i=1}^{k} K(\mathbf{x}, \mathbf{x}_i)},$$

  where K is a kernel function. One such function is the **Epanechnikov kernel**:

  $K(\mathbf{x}, \mathbf{x}_i) = \max\left(0, \frac{3}{4} \cdot \left(1 - \frac{D(\mathbf{x}, \mathbf{x}_i)^2}{\lambda^2}\right)\right)$, where D is Euclidean distance.

- A **regression tree** is like a decision tree, but the training points in the leaves do not generally all have the same output value, so we use the average of their outputs as our prediction. And instead of minimizing entropy, we can minimize the variance. The variance σ^2 of a set of values $y_1, ..., y_m$ is

  $$\sigma^2 = \frac{1}{m-1} \sum_{k=1}^{m} (y_k - \bar{y})^2, \text{ where } \bar{y} = \frac{1}{m} \sum_{k=1}^{m} y_k.$$

- The most common type of regression is **linear regression**. Like linear classification, we want to learn a weight vector $\mathbf{w}$, but our predicted output will just be $\mathbf{w} \cdot \mathbf{x}$ rather than $\text{sign}(\mathbf{w} \cdot \mathbf{x})$.

- Usually we define the best $\mathbf{w}$ as the one that minimizes the mean squared error. Recall that the mean squared error is

  $$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2.$$

  The $\mathbf{w}$ that minimizes this error is $\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$, where $\mathbf{X}$ is a matrix with the training points $\mathbf{x}_i$ as rows, and $\mathbf{y}$ is a vector of the corresponding outputs.

- Alternatively, we could minimize a different error function. In **ridge regression**, we use

  $J_n(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}\cdot\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|^2.$

  The added term is called a "regularization" term. It penalizes large weights, and it also guarantees that the matrix in the expression for $\mathbf{w}^*$ will be invertible:
  $\mathbf{w}^* = (\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

  Lasso regression is similar to ridge regression except it uses the L1 norm
  $\|\mathbf{w}\|_{L1} = \sum_{i=0}^{d}|w_i|$ instead of $\|\mathbf{w}\|^2$.

Miscellaneous Topics

- A **neural network** has layers of perceptron units, where the outputs of each layer are the inputs to the next layer. In order to efficiently use gradient descent to find the weights, we can use a sigmoid function instead of using a sharp threshold function to get the outputs.

- Sometimes a dataset has lots of dimensions, and we want to use fewer dimensions to avoid overfitting. **Feature selection** is the process of selecting features to train with, either a subset of the existing features or a set of features that are functions of the existing ones.

- **Clustering** is the process of splitting data points into clusters; it is generally unsupervised. The **k-means** algorithm keeps a set of k means (points in feature space), puts each data point into the cluster of the mean it's closest to, recomputes each mean based on which points are now in its cluster, and repeats until convergence.

- When evaluating a classifier, it is good to consider both the false-positive rate and the false-negative rate. A **receiver operating characteristic (ROC)** curve plots the false-positive rate versus the true-positive rate.

Exercises

1. You are given a training set with n points in the x, y plane and are interested in training a univariate linear regression model for prediction of future data points.

   (a) Write down the function h_w that you would use to predict a value for a new data point at x_i.

   (b) Write down the loss function that you're trying to minimize in order to fit a line to the data.

   (c) Express the coefficients in your function h_w in a solvable system of equations. Can you obtain a closed-form solution for these coefficients?

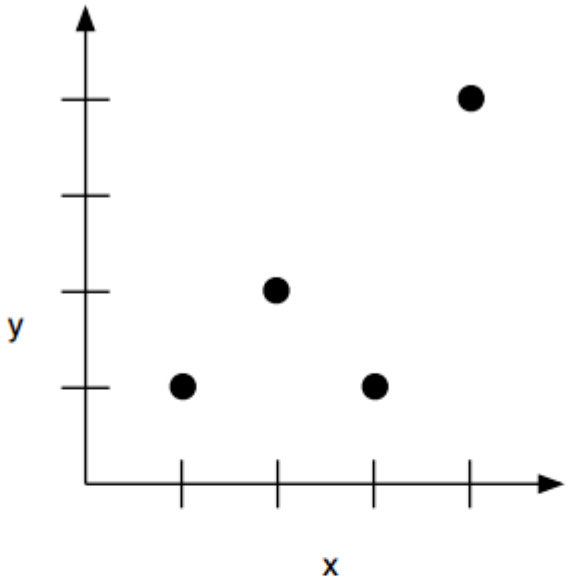
   (d) Suppose that instead of scalar values x and y your data is now multi-dimensional (that is, $\mathbf{x}$ is now an n-dimensional vector, but y is still a scalar value.) Can you obtain a closed-form solution for the coefficients in h_w?

   (e) Suppose now that you're interested in predicting output values for data with the following model: $y = w_0 + w_1 x + w_2 x^2$. Can you still use linear regression?

2. Draw a graph showing what function optimization might look like in a two-dimensional space, where the two axes correspond to values of w_1 and w_2. Write down the definition of L_1 and L_2 norms for vectors. Show how L_1 and L_2 regularization constrain the optimal values obtained by optimization in this space. Use your graph to explain why L_1 regularization leads to sparser models than L_2 regularization.

3. Consider a one-dimensional regression problem (predict y as a function of x). For each of the algorithms below, draw the approximate shape of the output of the algorithm, given the data points shown in the graph.

   1. 2-nearest-neighbor (equally weighted averaging)



   2. regression trees (with leaf size 1)