CS 181 Spring 2022 Section 1 Notes: Nonparametric Regression, Linear Regression, MLE

1 Types of Learning

1.1 Takeaways

1. Supervised - we are given the labels $\mathbf{y}$ during training. There are two main types.
   (a) Regression - labels $\mathbf{y}$ are continuous real numbers (usually, e.g. predict future stock price in dollars).
   (b) Classification - labels $\mathbf{y}$ are discrete and categorical (e.g. is this a picture of a pig, panda, or parakeet?).
2. Unsupervised - we are *not* given the labels $\mathbf{y}$ during training, and are only provided with the input data $\mathbf{x}$. Why would we even do unsupervised learning at all then? Well, generally, we use unsupervised learning to find out more about the construction of our data, relationships between our data, and "important" features of our data. Some examples of unsupervised learning include:
   (a) Clustering - finding natural groupings of our data.
   (b) PCA - projecting high dimensional data into lower dimensions (we'll discuss why this is important in the future).
3. Reinforcement Learning (RL) - we'll save this one for later! In summary, RL can be broken down into $(states, actions, rewards)$. Many strategy game bots (any StarCraft folks?) are designed using RL.

2 KNN and Kernelized Regression

2.1 K-Nearest Neighbors (KNN)

1. KNN is considered a form of non-parametric regression. Intuitively, for a fixed value of K=k, there are no parameters that we have to tune (i.e., no weights that we have to learn). More rigorously, non-parametric means that we do not make any assumptions about the parameters / characteristics of our data (in context, this means that we do not assume an underlying probability distribution).
2. Rundown of the KNN Algorithm:
   (a) Let x^* be the point that we would like to make a prediction about. Let's find the k nearest points $\{\mathbf{x}_1, \dots \mathbf{x}_k\}$ to $\mathbf{x}^*$, based on some predetermined distance function.
   (b) Denote the true y values of these k points as $\{y_1, \dots y_k\}$.

(c) Output our prediction $\hat{y}^*$ for our point of interest $\mathbf{x}^*$:

$$\hat{y}^* = \frac{\sum_{i=1}^{k} y_k}{k}$$

Remark: the little "hat" in $\hat{y}^*$ is just notation telling us that this is our *prediction*, rather than the true y value.

2.2 Kernelized Regression

Note: You might come across this concept under the name kernel-weighted average regression.

1. Kernelized Regression is considered to be a smoother, more general extension of KNN.

2. Let $k(\mathbf{x}^*, \mathbf{x}_n)$ be our "kernel function." In Kernelized Regression, we want to take a *weighted average* of all the points in our training data when outputting our prediction for an unknown point. Intuitively, we want to weigh points that are "closer" to our unknown point of interest *more heavily* than points that are "farther" away. As such, our kernel function $k(\mathbf{x}^*, \mathbf{x_n})$ should be *larger* for a point $\mathbf{x_n}$ closer to our point of interest $\mathbf{x}^*$ than a point $\mathbf{x_n}$ farther away.

3. Importantly, the following should always hold:

$$\arg\max_{\mathbf{x}} k(\mathbf{x}^*, \mathbf{x}) = \mathbf{x}^*$$

   Remark: What the above statement means is that "the value of $\mathbf{x}$ that results in the largest value of $k(\mathbf{x}^*, \mathbf{x})$ should be $\mathbf{x}^*$ itself".

4. Rundown of the Kernelized Algorithm:

   (a) Let $\{\mathbf{x}_1, \ldots \mathbf{x}_N\}$ (and their corresponding y values) be *all* of the N points comprising our training data set.

   (b) Let $\mathbf{x}^*$ be our point of interest that we want to make a prediction for. We make our prediction as follows:

$$\hat{y}^* = \frac{\sum_{i=1}^{N} k(\mathbf{x}^*, \mathbf{x}_i) \cdot y_i}{\sum_{j=1}^{N} k(\mathbf{x}^*, \mathbf{x}_j)}$$

   Remark: notice how we include *all* points of the training data in our calculation?

   Remark: we have to include the denominator term in order to normalize the sum of our weights to equal 1.

3 Least Squares (Linear) Regression

3.1 Takeaways

3.1.1 Linear Regression

The simplest model for regression involves a linear combination of the input variables:

$$h(\mathbf{x};\mathbf{w}) = w_1 x_1 + w_2 x_2 + \ldots + w_D x_D = \sum_{d=1}^{D} w_d x_d = \mathbf{w}^\top \mathbf{x} \tag{1}$$

where $x_j \in \mathbb{R}$ for $j \in \{1, \ldots, D\}$ are the features, $\mathbf{w} \in \mathbb{R}^D$ is the weight parameter, with $w_1 \in \mathbb{R}$ being the bias parameter. Recall the trick of letting $x_1 = 1$ to merge bias.

3.1.2 Least Squares Loss Function

The least squares loss function assuming a basic linear model is given as follows:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \mathbf{w}^\top \mathbf{x}_n\right)^2 \tag{2}$$

For regularized regression, such as LASSO (L1) or Ridge (L2) regression, we use a different loss function with an added penalty term. The L1 penalty is $\lambda\|\mathbf{w}\|_1$, and the L2 penalty is $\lambda\|\mathbf{w}\|_2^2$.

A few remarks on notation:

1. Let $\mathbf{w} = \{w_1, w_2, \ldots, w_D\}$
2. LASSO (L1) Regularization (written out):

$$\lambda|\mathbf{w}| = \lambda\left(\sum_{i=1}^{D}|w_i|\right)$$

   Note: $|\mathbf{w}|$ means the same thing as $\|\mathbf{w}\|_1$, depending on textbook/region.

3. Ridge (L2) Regularization (written out):

$$\lambda\|\mathbf{w}\|_2^2 = \lambda \cdot \left(\sqrt{w_1^2 + \cdots + w_D^2}\right)^2 = \lambda\,(w_1^2 + \cdots + w_D^2)$$

Remark: $\|\mathbf{w}\|$ means the "norm" of $\mathbf{w}$. Usually, this means the L2 norm. In most contexts, $\|\mathbf{w}\|$ means the same thing as $\|\mathbf{w}\|_2$.

Remark: occasionally, you may see a $\frac{\lambda}{2}$ instead of λ. Conceptually, these two conventions are very similar. The dividing by 2 just helps us make things a little bit nicer when we take the derivative.

3.1.3 Optimizing Weights to Minimize Loss Function

In class, we briefly discussed how to find the weights that minimize the least squares loss function. Let us derive the end result from scratch again, in more detail. This will be a good exercise for future problem sets and topics.

Some important notes on notation and dimensions:

1. In this class, if $y \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^n$, then

$$\frac{dy}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix}$$

   Note that this derivative is a *column* vector!

2. If $\mathbf{y} \in \mathbb{R}^m$ and $\mathbf{x} \in \mathbb{R}^n$, then

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

   Note that this derivative is basically like stacking the $\frac{dy_i}{d\mathbf{x}}$ column vectors together like "books on a bookshelf!"

3. By convention, we will treat our vector of y-values, $\mathbf{y}$, as a *column* vector. We will also treat our weight vector, $\mathbf{w}$, as a *column* vector. Finally, we will treat each *individual* data point $\mathbf{x}_i$ as a column vector.

4. *However*, we will treat our data matrix $\mathbf{X}$ (of all the individual data points $\mathbf{x_i}$ combined together), which is also called a "design matrix," as follows. This is not a typo:

$$\mathbf{X} = \begin{bmatrix} \leftarrow \mathbf{x_1}^T \rightarrow \\ \leftarrow \mathbf{x_2}^T \rightarrow \\ \vdots \\ \leftarrow \mathbf{x_n}^T \rightarrow \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1D} \\ x_{21} & \cdots & x_{2D} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nD} \end{bmatrix}.$$

   D is the number of dimensions in our data (i.e., the dimensions of each data point $\mathbf{x_i}$).

Now, let us find the weights that minimize the least squares loss function.

1. We begin with our loss function:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \mathbf{w}^\top \mathbf{x}_n\right)^2$$

2. A few remarks on this loss function:

   (a) $\hat{y_n} = \mathbf{w}^\top \mathbf{x}_n$ is our predicted y value for each $\mathbf{x}_n$, under linear regression. Thus, our loss function could also be written as the following for intuition:

   $$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (y_n - \hat{y_n})^2$$

   (b) The presence of the $\frac{1}{2}$ is just to make things neater when we take the derivative.

   (c) The loss function itself outputs a *scalar* value! The total loss is *always* a scalar real value, and *not* vector!

3. To find $\mathbf{w}$ that minimizes our loss function, we take the derivative of the loss function (again, a scalar) with respect to $\mathbf{w}$ and set the derivative equal to 0. By the power, sum (across the summation), and chain rules, we have the following:

   $$\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_{n=1}^{N} \left((y_n - \mathbf{w}^\top \mathbf{x}_n) \cdot \frac{d(y_n - \mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}} \right) = 0$$

4. Let's look at that inner derivative a bit more closely. This is a good example of how to think through a matrix derivative. By rules from single variable calculus, we have:

   $$\frac{d(y_n - \mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}} = \frac{d(-\mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}} = -\frac{d(\mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}}$$

   Note that $\mathbf{w}^\top \mathbf{x}_n$ is a scalar function, while $\mathbf{w}$ is a vector. Thus, our resultant derivative should be a column vector. We rewrite the following for intuition:

   $$\mathbf{w}^\top \mathbf{x}_n = w_1 x_{n1} + w_2 x_{n2} + \cdots + w_D x_{nD}$$

   For any arbitrary element of $\mathbf{w}$, which we'll call w_i, we have, via single variable calculus:

   $$\frac{d(\mathbf{w}^\top \mathbf{x}_n)}{dw_i} = x_{ni}$$

   From our notational conventions previously combined with our observation directly above,

   $$\frac{d(\mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}} = \begin{bmatrix} \frac{\partial(\mathbf{w}^\top \mathbf{x}_n)}{\partial w_1} \\ \frac{\partial(\mathbf{w}^\top \mathbf{x}_n)}{\partial w_2} \\ \vdots \\ \frac{\partial(\mathbf{w}^\top \mathbf{x}_n)}{\partial w_D} \end{bmatrix} = \begin{bmatrix} x_{n1} \\ x_{n2} \\ \vdots \\ x_{nD} \end{bmatrix} = \mathbf{x}_n$$

5. Now, going back to our derivative of the loss function with respect to $\mathbf{w}$, we have:

   $$\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_{n=1}^{N} \left((y_n - \mathbf{w}^\top \mathbf{x}_n) \cdot \frac{d(y_n - \mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}} \right) = \sum_{n=1}^{N} \left((y_n - \mathbf{w}^\top \mathbf{x}_n) \cdot -\frac{d(\mathbf{w}^\top \mathbf{x}_n)}{d\mathbf{w}} \right) = 0$$

$$\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_{n=1}^{N} \Big((y_n - \mathbf{w}^\top \mathbf{x}_n) \cdot -\mathbf{x}_n \Big) = \sum_{n=1}^{N} \Big(- y_n \mathbf{x}_n + (\mathbf{w}^\top \mathbf{x}_n)\mathbf{x}_n \Big) = 0$$

Equivalent, we can move the negative terms to the right hand side (RHS):

$$\sum_{n=1}^{N} (\mathbf{w}^\top \mathbf{x}_n)\mathbf{x}_n = \sum_{n=1}^{N} y_n \mathbf{x}_n$$

6. Note that $\mathbf{w}^\top \mathbf{x}_n$ is a *scalar*, and that $\mathbf{w}^\top \mathbf{x}_n = \mathbf{x}_n^T \mathbf{w}$. This is important because we can left-multiply or right-multiply by scalars however we want:

$$\sum_{n=1}^{N} (\mathbf{w}^\top \mathbf{x}_n)\mathbf{x}_n = \sum_{n=1}^{N} (\mathbf{x}_n^T \mathbf{w})\mathbf{x}_n = \sum_{n=1}^{N} \mathbf{x}_n (\mathbf{x}_n^T \mathbf{w}) = \sum_{n=1}^{N} y_n \mathbf{x}_n$$

7. One key observation is that $\mathbf{w}$ does *not* depend on the value of our index variable n in the summation! Thus, we can take it out of our summation:

$$\sum_{n=1}^{N} \mathbf{x}_n (\mathbf{x}_n^T \mathbf{w}) = \Big(\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T \Big) \mathbf{w}$$

8. Remember our "design matrix" $\mathbf{X}$ from earlier? Also, recall that $\mathbf{y}$ is a *column* vector! If we draw out $\mathbf{X}$, $\mathbf{X}^T$, and $\mathbf{y}$, we will soon notice that

$$\mathbf{X}^T \mathbf{y} = \sum_{n=1}^{N} y_n \mathbf{x}_n = \sum_{n=1}^{N} \mathbf{x}_n y_n$$

9. If we draw out $\mathbf{X}$ and $\mathbf{X}^T$, we will realize that

$$\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T = \mathbf{X}^T \mathbf{X}$$

And of course,

$$\Big(\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T \Big) \mathbf{w} = \Big(\mathbf{X}^T \mathbf{X} \Big) \mathbf{w}$$

10. Putting our LHS and RHS together, we have:

$$\Big(\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T \Big) \mathbf{w} = \sum_{n=1}^{N} \mathbf{x}_n y_n \Leftrightarrow \Big(\mathbf{X}^T \mathbf{X} \Big) \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

11. Taking the liberty to assume that $\mathbf{X}^T \mathbf{X}$ is invertible, we can isolate $\mathbf{w}$:

$$\mathbf{w}^* = \Big(\mathbf{X}^T \mathbf{X} \Big)^{-1} \mathbf{X}^T \mathbf{y}$$

The little $*$ in $\mathbf{w}^*$ just means that these are the *optimal* weights, as opposed to any generic set of weights.

In short, if we minimize our least squares loss function with respect to the weights, we get the following solution:

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y} = \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \tag{3}$$

where $\mathbf{X} \in \mathbb{R}^{N \times D}$. Each row represents one data point and each column represents values of one feature across all the data points. In practice, gradient descent is often used to compute w^*. Today, we just got super lucky that there was a clean-cut closed-form analytical solution. [1]

3.2 Concept Question

How is a model (such as linear regression) related to a loss function (such as least squares)?

- The model (of the data) and the loss functions are both important pieces to the ML pipeline, but they are distinct. The model describes how you believe the data is related and/or generated. Very commonly, you will be optimizing over a family of models. The loss function measures how well a specific model (i.e. with specific parameters) fits the data, and it is used in the previously mentioned optimization.
- Least squares and linear regression are often used together, especially since there are theoretical justifications (i.e. MLE connection) to why least squares is a good loss function for linear regression. However, you do **not** have to use them together. Another loss function that could be used with linear regression is an absolute difference (L1) loss.

[1]Note: $(\mathbf{X}^T\mathbf{X})^{-1}$ is invertible iff X is full column rank (i.e. rank D, which implies $N \geq D$). **What if $(\mathbf{X}^T\mathbf{X})^{-1}$ is not invertible?** Then, there is not a unique solution for w^*. If $d > N$, computing the pseudoinverse of $\mathbf{X}^T\mathbf{X}$ will find one solution. Alternatively, in general applying ridge regression can fix the invertibility issue.

3.3 Exercise: Practice Minimizing Least Squares

Let $\mathbf{X} \in \mathbb{R}^{N \times D-1}$ be our design matrix, $\mathbf{y}$ our vector of N target values, $\mathbf{w}$ our vector of $D-1$ parameters, and w_0 our bias parameter. The least squares error function of $\mathbf{w}$ and w_0 can be written as follows

$$\mathcal{L}(\mathbf{w}, w_0) = \frac{1}{2} \sum_{n=1}^{N} \left(y_n - w_0 - \sum_{d=1}^{D-1} w_d X_{nd} \right)^2 .$$

Find the value of w_0 that minimizes $\mathcal{L}$. Can you write it in both vector notation and summation notation? Does the result make sense intuitively?

Solution: We minimize by finding gradient w.r.t w_0, setting to 0, and solving.

$$\begin{aligned}
\frac{\partial L}{\partial w_0} &= -\sum_{n=1}^{N} (y_n - w_0 - \sum_{d=1}^{D-1} w_d X_{nd}) = 0 \\
N w_0^* &= \sum_{n=1}^{N} \left(y_n - \sum_{d=1}^{D-1} w_d X_{nd} \right) \\
w_0^* &= \frac{1}{N} \left[\left(\sum_{n=1}^{N} y_n \right) - \sum_{n=1}^{N} \sum_{d=1}^{D-1} w_d X_{nd} \right] \\
&= \frac{1}{N} \left(\mathbf{y}^\top \mathbf{1} - \sum_{n=1}^{N} \mathbf{w}^\top \mathbf{x}_n \right)
\end{aligned}$$

The result makes sense intuitively as it is the average deviation of the outputs from the predictions.

4 Linear Basis Function Regression

4.1 Takeaways

We allow $h(\mathbf{x};\mathbf{w})$ to be a non-linear function of the input vector $\mathbf{x} \in \mathbb{R}^D$, while remaining linear in $\mathbf{w} \in \mathbb{R}^M$ by using a basis function $\phi : \mathbb{R}^D \to \mathbb{R}^M$. The resulting basis regression model is below:

$$h(\mathbf{x};\mathbf{w}) = \sum_{m=1}^{M} w_m \phi_m(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) \tag{4}$$

To merge the bias term, we can define $\phi_1(\mathbf{x}) = 1$. Some examples of basis functions include polynomial $\phi_m(x) = x^m$, Fourier $\phi_m(x) = \cos(m\pi x)$, and Gaussian $\phi_m(x) = \exp\{-\frac{(x-\mu_m)^2}{2s^2}\}$.

4.2 Concept Questions

- What are some advantages and disadvantages to using linear basis function regression to basic linear regression?
- How do we choose the bases?

Basis functions allow us to capture nonlinear relations that may exist in the data, which linear functions can not. There is, however, a greater risk of overfitting with the more flexible linear basis function regression - more on this in the upcoming weeks in lecture with bias-variance tradeoff.

We can choose the bases with expert domain knowledge, or they could even be learned themselves... (foreshadowing neural nets).

We don't talk about feature engineering and incorporating expert domain knowledge too much in this class; however, these are vital in real world situations for good performance. The practical assignment and pset problems may touch on this.

5 Maximum Likelihood Estimation (Prep for Future + Pset)

5.1 Takeaways

- Given a model and observed data, the **maximum likelihood estimate** (of the parameters) is the estimate that maximizes the probability DENSITY of seeing the observed data under the model.
- It is obtained by maximizing the **likelihood function**, which is the same as the joint pdf of the data, but viewed as a function of the parameters rather than the data.
- Since log is monotonic function, we will often maximize the **log likelihood** rather than the likelihood as it is easier (turns products from independent data into sums) and results in the same solution.

5.2 Exercise: MLE for Gaussian Data

We are given a data set $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ where each observation is drawn independently from a multivariate Gaussian distribution:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{|(2\pi)\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{5}$$

where $\boldsymbol{\mu}$ is a m-dimensional mean vector, $\boldsymbol{\Sigma}$ is a m by m covariance matrix, and $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$. Find the maximum likelihood value of the mean, $\boldsymbol{\mu}_{MLE}$.

Solution: We find the MLE by maximizing the log likelihood:

$$\begin{aligned} l(\boldsymbol{\mu}, \boldsymbol{\Sigma}; \mathbf{x}) &= \log\left(\prod_{i=1}^{n} \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma})\right) = \sum_{i=1}^{n} \log(\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma})) \\ &= -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(|\boldsymbol{\Sigma}|) - \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}) \end{aligned}$$

Taking the derivative (matrix cookbook ch 2.4 eqn 78) with respect to $\boldsymbol{\mu}$ and setting it equal to 0, we get

$$0 = \frac{\partial l}{\partial \boldsymbol{\mu}} = \sum_{i=1}^{n} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu})$$

and solving gives us that

$$\boldsymbol{\mu}_{MLE} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i.$$