1 Getting Started

You may typeset your homework in latex or submit neatly handwritten and scanned solutions. Please make sure to start each question on a new page, as grading (with Gradescope) is much easier that way! Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, "HW[n] Write-Up"
2. Submit all code needed to reproduce your results, "HW[n] Code".
3. Submit your test set evaluation results, "HW[n] Test Set".

After you've submitted your homework, be sure to watch out for the self-grade form.

(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

Jinhong Du
jaydu@berkeley.edu

(b) Please copy the following statement and sign next to it:

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up. Jinhong Du

2 Geometry of Ridge Regression

One way to interpret "ridge regression" is as the Lagrangian form of a constrained problem.

(a) Given a matrix $X \in \mathbb{R}^{n \times d}$ and a vector $\vec{y} \in \mathbb{R}^n$, define the optimization problem

$$\begin{aligned} &\text{minimize } \|\vec{y} - X\vec{w}\|_2^2. \\ &\text{subject to } \|\vec{w}\|_2^2 \leq \beta^2. \end{aligned} \tag{1}$$

Using the spirit of the method of Lagrange multipliers (wherein we replace a constraint with a penalty on the thing that we are constraining, and then adjust the level of the penalty until the solution ends up satisfying the constraint. These penalties are sometimes referred to as "shadow prices," especially in the economics literature.), rewrite the constrained optimization problem an an unconstrained optimization with the constraint incorporated within the objective function.

Recall that ridge regression is given by the unconstrained optimization problem

$$w = \arg\min_w \|\vec{y} - X\vec{w}\|_2^2 + \lambda \|\vec{w}\|_2^2. \tag{2}$$

Hence, by performing ridge regression with penalty λ, we are essentially solving a constrained least squares problem with our parameter having bounded Euclidean norm β. **Qualitatively, how would increasing β be reflected in the desired penalty λ of ridge regression?**

Let

$$L(w, \lambda) = \|y - Xw\|_2^2 + \lambda(\|w\|_2^2 - \beta^2)$$

where $\lambda > 0$, which statisfies Karush-Kuhn-Tucker conditions.

$\therefore$

$$\max_\lambda L(w, \lambda) \approx \|y - Xw\|_2^2$$

$\therefore$ with constrained condition $\|w\|_2^2 \leqslant \beta^2$,

$$\min_w \|y - Xw\|_2^2 = \min_w \max_\lambda L(w, \lambda)$$

Because $\|w\|_2^2 - \beta^2 \leqslant 0$, to first maximize $L(w, \lambda)$, we actually minimize λ or maximize $\|w\|_2^2 - \beta^2$. So when increasing β, $\|w\|_2^2 - \beta^2$ is decreasing. So λ will increase.

(b) One reason why we might want to have small weights $\vec{w}$ has to do with the sensitivity of the predictor to its input. Let $\vec{x}$ be a d-dimensional list of features corresponding to a new test point. Our predictor is $\vec{w}^\top \vec{x}$. **By how much can our prediction change if nature added an arbitrary disturbance vector of length ε to the test point's features $\vec{x}$?**

$$
\begin{aligned}
w^T(x+\varepsilon) &= (w_1,\cdots,w_d)(x_1+\varepsilon_2,\cdots,x_d+\varepsilon_d)^T \\
&= \sum_{i=1}^{d} w_i(x_i+\varepsilon_i) \\
&= \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} w_i \varepsilon_i \\
&= w^T x + w^T \varepsilon \\
|w^T \varepsilon| &\leqslant \sum_{i}^{d} |w_i||\varepsilon_i| \\
&\leqslant \max\{|w_1|,|w_2|,\cdots,|w_d|\} \cdot \max\{|\varepsilon_1|,|\varepsilon_2|,\cdots,|\varepsilon_d|\}
\end{aligned}
$$

$\therefore$ when one component of the w is to large, then the error may be large.

(c) **Derive that the solution to ridge regression** (2) **is given by** $\hat{w}_r = (X^\top X + \lambda I)^{-1} X^\top y$**. What happens when** $\lambda \to \infty$**?** It is for this reason that sometimes regularization is referred to as "shrinkage."

Let

$$
\begin{aligned}
\frac{\partial}{\partial w}(\|y-Xw\|_2^2 + \lambda\|w\|_2^2) &= \frac{\partial}{\partial w}[(y-Xw)^T(y-Xw) + \lambda w^T w] \\
&= \frac{\partial}{\partial w}[y^T y - w^T X^T y - yXw + w^T X^T X w + \lambda w^T w] \\
&= 0 - X^T y - X^T y + 2X^T X w + 2\lambda I w \\
&= 0
\end{aligned}
$$

We get

$$
\begin{aligned}
(X^T X + \lambda I)w &= X^T y \\
w &= (X^T X + \lambda I)^{-1} X^T y
\end{aligned}
$$

$\therefore$ $\hat{w}_r = (X^T X + \lambda I)^{-1} X^T y$

When $\lambda \to \infty$, the absolute values of eigenvalues of $X^T X + \lambda I$ will get much bigger. Then the absolute values of eigenvalues $(X^T X + \lambda I)^{-1}$ will be close to 0. And then $|w|$ will get much smaller.

(d) Note that in computing $\hat{w}_r$, we are trying to invert the matrix $X^\top X + \lambda I$ instead of the matrix $X^\top X$. **If** $X^\top X$ **has eigenvalues** $\lambda_1,\ldots,\lambda_d$**, what are the eigenvalues of** $X^\top X + \lambda I$**? Comment on why adding the regularizer term** λI **can improve the inversion operation numerically.**

$\because$ $\lambda_1,\cdots,\lambda_d$ are eigenvalues of $X^T X$

$$
|X^T X - \lambda_i I| = 0
$$

$\therefore$

$$|(X^TX+\lambda I)-(\lambda_i+\lambda)I|=0$$

$\therefore$ the eigenvalues of $X^TX+\lambda I$ are $\lambda_1+\lambda,\cdots,\lambda_d+\lambda$.

When $\max_i\{\lambda_i\}$ is too small, the computing of the inversion operation will be terrible because $\frac{1}{\lambda_i}$ can be very big with round-off error in λ_i(i.e. λ_i is close to 0). So by adding λI, we can improve the inversion operation numerically.

(e) Let $d=3$, $n=5$, and let the eigenvalues of $X^\top X$ be given by 1000, 1 and 0.001. We must now choose between two regularization parameters $\lambda_1=100$ and $\lambda_2=0.5$. **Which do you think is a better choice for this problem and why?**

I think λ_2 is the best choice.

$\lambda(X^TX)=1000,1,0.001,\qquad \lambda((X^TX)^{-1})=0.0001,1,1000$

$\because\quad \lambda((X^TX+\lambda_1)^{-1})=\frac{1}{1100},\frac{1}{101},\frac{1}{100.001},\lambda((X^TX+\lambda_2)^{-1})=\frac{1}{1000.5},\frac{1}{1.5},\frac{1}{0.51}$

$\therefore$ the second and third eigenvalues of X^TX change a lot, i.e. by adding λ_2 the most features of X^TX remains.

So λ_2 is better.

(f) Another advantage of ridge regression can be seen for under-determined systems. Say we have the data drawn from a 5 parameter model, but only have 4 samples of it, i.e. $X\in\mathbb{R}^{4\times5}$. Now this is clearly an underdetermined system, since $n<d$. **Show that ridge regression with $\lambda>0$ results in a unique solution, whereas ordinary least squares has an infinite number of solutions.**

Hint: To make this point, it may be helpful to expand any vector w as $w=w_0+X^\top a$ for $w_0\in\mathsf{nullspace}(X)$ and some a.

$\because\quad X^TX\in\mathbb{R}^{5\times5}$ $rank(X^TX)\leqslant\{rank(X),rank(X^T)\}\leqslant4$

$\therefore\quad \exists$ one eigenvalue λ_j of X^TX, $\lambda_j=0$ i.e. $X^TXw=X^Ty$ has infinite solutions for y.

$\because\quad \forall\, w\in\mathbb{R}^{5\times1}$, $w^T(X^TX)w=\|X^TX\|\|w\|\geqslant0$

$\therefore\quad$ the eigenvalues of X^TX are nonnegative

By adding $\lambda I(\lambda>0)$, the eigenvalues of $X^TX+\lambda I$ are positive. Then $(X^TX+\lambda I)w=X^Ty$ has unique solution.

(g) (**BONUS**) For the previous part, **what will the answer be if you take the limit $\lambda\to0$ for ridge regression?**

When $\lambda \to 0$, $X^TX + \lambda I$ in (f) is an almost singular matrix.

The numberical result of eigenvalues of $(X^TX + \lambda I)^{-1}$ can be very large and very different from the true value.

So the estimate of $\hat{w} = (X^TX + \lambda I)^{-1}X^Ty$ can be numerically terrible.

(h) Tikhonov regularization is a general term for ridge regression, where the constraint set takes the form of an ellipsoid instead of a ball. In other words, we solve the optimization problem

$$w = \arg\min_w \|y - Xw\|_2^2 + \lambda \|\Gamma w\|_2^2$$

for some full rank matrix $\Gamma \in \mathbb{R}^{d \times d}$. **Derive a closed form solution to this problem.**

$$\begin{aligned}\|y - Xw\|_2^2 + \lambda \|\Gamma w\|_2^2 &= (y - Xw)^T(y - Xw) + \lambda(\Gamma w)^T(\Gamma w) \\ &= y^Ty + w^TX^TXw - w^TX^Ty - y^TXw + \lambda w^T\Gamma^T\Gamma w\end{aligned}$$

Set

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}w}(\|y - Xw\|_2^2 + \lambda \|\Gamma w\|_2^2) &= 2X^TXw - 2X^Ty + 2\lambda\Gamma^T\Gamma w \\ &= 0\end{aligned}$$

We get

$$w = (X^TX + \lambda\Gamma^T\Gamma)^{-1}X^Ty$$

Because Γ has full rank, every eigenvalue ofΓ is non-zero. So the eigenvalues of $\Gamma^T\Gamma$ are positive. So $X^TX + \lambda\Gamma^T\Gamma$ is invertible.

3 Polynomials and invertibility

Consider using a D-degree polynomial to fit a function $y = f(x)$ with n training samples, where both x and y are scalar. We know that this is equivalent to performing linear regression with a feature matrix F constructed from the n training data sampling positions $x_1, \ldots, x_n$. Assume all the training sampling positions are non-zero, and let this mapping be given by $F = [\vec{p}_D(x_1), \ldots, \vec{p}_D(x_n)]^T$ where $\vec{p}_D(x) = [x^0, x^1, \ldots, x^D]^T$.

(a) For $n = 2$ and $D = 1$, **show that the matrix F has full rank iff $x_1 \neq x_2$.**

When $n = 2$ and $D = 1$,

$$F = \begin{bmatrix} x_1^0 & x_1^1 \\ x_2^0 & x_2^1 \end{bmatrix}$$

F has full rank iff

$$\begin{aligned}|F| &= \begin{vmatrix} x_1^0 & x_1^1 \\ x_2^0 & x_2^1 \end{vmatrix} \\ &= x_1^0 x_2^1 - x_1^1 x_2^0 \\ &= x_2 - x_1 \\ &\neq 0\end{aligned}$$

i.e.

$$x_1 = x_2$$

(b) More generally, let us now show that the columns of F are linearly independent provided the sampling data points are distinct and $n \geq D+1$. It suffices to consider the case $n = D+1$ and so assume that from this point forward for all the questions about univariate polynomials.

We do this by performing the following operations in sequence. From the matrix F, subtract the first row from rows 2 through n to obtain matrix F'.

Is it true that $\det(F) = \det(F')$**?**

Hint: Think about representing the row subtraction operation using a matrix multiplication, and argue why this additional matrix must have determinant 1. (What are the eigenvalues of a triangular matrix?)

It is true that $det(F) = det(F')$.

$$\begin{aligned}F' &= \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ -1 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -1 & \cdots & \cdots & 0 & 1 \end{bmatrix} F \\ &= PF\end{aligned}$$

$\because$ the eigenvalues of triangular matrix is elements on the diagonal, determinant of P is the product of its eigenvalues, so

$$|P| = 1$$

$\therefore$

$$|F| = |F'|$$

(c) Perform the following sequence of operations to F', and obtain the matrix F''.

i) Subtract $x_1 * \mathsf{column}_{n-1}$ from column_n.

ii) Subtract $x_1 * \mathsf{column}_{n-2}$ from column_{n-1}.

$\vdots$

n-1) Subtract $x_1 * \mathsf{column}_1$ from column_2.

Write out the matrix F'' and argue why $\det(F') = \det(F'')$.

It is true that $det(F) = det(F')$.

$$
\begin{aligned}
F'' &= F \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -1 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \\
&\quad \cdot \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & -1 & 0 \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \\
&\quad \cdots \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \\
&= F \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -1 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \\
&= FQ
\end{aligned}
$$

$\because$

$$|Q| = 1$$

$\therefore$

$$|F| = |F''|$$

(d) For any square matrix $A \in \mathbb{R}^{d \times d}$ and a matrix

$$B = \begin{bmatrix} 1 & \vec{0}^\top \\ \vec{0} & A \end{bmatrix},$$

argue that the $d+1$ eigenvalues of B are given by $\{1, \lambda_1(A), \lambda_2(A), \ldots, \lambda_d(A)\}$, and conclude that $\det(B) = \det(A)$. Here, $\vec{0}$ represents a column vector of zeros in $\mathbb{R}^d$.

Let

$$\begin{aligned} |B - \lambda I| &= \begin{vmatrix} 1-\lambda I & 0 \\ 0 & A - \lambda I \end{vmatrix} \\ &= (1-\lambda)|A - \lambda I| \\ &= 0 \end{aligned}$$

We have $\lambda = 1$ or $|A - \lambda I| = 0$, i.e. $\lambda \in \{1, \lambda_1(A), \cdots, \lambda_d(A)\}$.

$\therefore$

$$det(B) = 1 \cdot \prod_{i=1}^{d} \lambda_i(A) = \det(A)$$

(e) **Use the above parts to show by induction that we have $\det(F) = \prod_{1 \le i < j \le n}(x_j - x_i)$.** Consequently, the matrix X is full rank unless two data points are equal.

Hint: First show that

$$\det(F) = \prod_{i=2}^{n}(x_i - x_1)\det([\vec{p}_{D-1}(x_2), \vec{p}_{D-1}(x_3), \ldots, \vec{p}_{D-1}(x_n)]^T).$$

Hint Hint: You can use the fact that multiplying a row of a matrix by a constant scales the determinant by this constant. (A fact that is clear from the oriented volume interpretation of determinants.)

$$\begin{aligned} det(F) &= \begin{vmatrix} 1 & x_1 & \cdots & x_1^D \\ 1 & x_2 & \cdots & x_2^D \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & \cdots & x_n^D \end{vmatrix} \\ &= \begin{vmatrix} 1 & 0 & \cdots & 0 \\ 1 & x_2 - x_1 & \cdots & x_2^D - x_1 x_2^{D-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n - x_1 & \cdots & x_n^D - x_1 x_n^{D-1} \end{vmatrix} \\ &= \begin{vmatrix} x_2 - x_1 & x_2(x_2 - x_1) & \cdots & x_2^{D-1}(x_2 - x_1) \\ \vdots & \vdots & \vdots & \vdots \\ x_n - x_1 & x_n(x_n - x_1) & \cdots & x_n^{D-1}(x_n - x_1) \end{vmatrix} \\ &= \prod_{i=2}^{n}(x_i - x_1) \begin{vmatrix} 1 & x_2 & \cdots & x_2^{D-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & \cdots & x_n^{D-1} \end{vmatrix} \\ &= \prod_{i=2}^{n}(x_i - x_1)\det([\vec{p}_{D-1}(x_2), \vec{p}_{D-1}(x_3), \ldots, \vec{p}_{D-1}(x_n)]^T) \end{aligned}$$

$\therefore$

$$\begin{aligned}
det(F) &= \prod_{i=2}^{n}(x_i - x_1)\det([\vec{p}_{D-1}(x_2), \vec{p}_{D-1}(x_3), \ldots, \vec{p}_{D-1}(x_n)]^T) \\
&= \prod_{i=2}^{n}(x_i - x_1)\prod_{i=3}^{n}(x_i - x_2)\det([\vec{p}_{D-1}(x_3), \ldots, \vec{p}_{D-1}(x_n)]^T) \\
&= \cdots \\
&= \prod_{i=2}^{n}(x_i - x_1)\prod_{i=3}^{n}(x_i - x_2)\cdots\prod_{i=n}^{n}(x_i - x_{n-1}) \\
&= \prod_{1\leqslant i<j\leqslant n}(x_j - x_i)
\end{aligned}$$

(f) Let us now extend this argument to features from a multidimensional space of dimension ℓ, using a multivariate polynomial of degree D.

Using a stars and bars (link is here if you did not take CS70) argument, **show that now, we have $\binom{D+\ell}{\ell}$ features for each sampling point.**

Every features can be described as $x_1^{a_1}x_2^{a_2}\cdots x_l^{a_l}$ where $a_1+\cdots+a_l = d(a_i \in \mathbb{N})$, and d is the degree of this feature.

The number of 0^{th}-degree features is $1 = \binom{0+l-1}{l-1} = \binom{0+l}{l}$.

The number of 1^{th}-degree features is $\binom{l}{1} = \binom{1+l-1}{l-1}$.

$\vdots$

The number of D^{th}-degree features is $\binom{D+l-1}{l-1}$.(D stars, $l-1$ bars, $D+l-1$ positions from which to choose $l-1$ bar positions)

Therefore,

$$\begin{aligned}
\sum_{i=0}^{D}\binom{i+l-1}{l-1} &= \binom{1+l}{l} + \sum_{i=2}^{D}\binom{i+l-1}{l-1} \\
&= \binom{2+l}{l} + \sum_{i=3}^{D}\binom{i+l-1}{l-1} \\
&= \cdots \\
&= \binom{D+l}{l}
\end{aligned}$$

Here we use $\binom{m}{n} = \binom{m-1}{n-1} + \binom{m-1}{n}$.

(g) Choose n sample points $\{\vec{x}_i\}_{i=1}^n$ with $\vec{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,\ell})^T$, and stack up all their features like in part (a) to form the feature matrix F_ℓ.

First, we will show that for some choice of distinct sampling points, this may not be full rank. Let us now form a particular instance of the matrix F_ℓ by choosing $x_{i,1} = x_{i,2} = \cdots = x_{i,\ell} = \alpha_i$ for distinct α_i

as $i \in \{1,2,3\ldots,n\}$. **Show that this leads to an F_ℓ with linearly dependent columns no matter how many samples n you take.**

For $D = 1$,

$$F_l = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \end{bmatrix}$$
$$= [a_1, \cdots, a_n]$$

where $a_i \in \mathbb{R}^{l \times 1}$.

$\exists\, a_j \neq 0$. If not, $\alpha_1 = \cdots = \alpha_n = 0$.

$\because$ $a_j \neq 0, \forall\, i \neq j,\ i \in \mathbb{N}, a_i = \dfrac{\alpha_i}{\alpha_j} a_j$

$\therefore$ F_l has linearly dependent columns.

(h) To show that the matrix can be full rank, **pick a set of sampling points $\vec{x}_i$ to show that there is a way to choose the samples to get the matrix F_ℓ to be full column rank.** You are allowed to pick as many sample points as you want. Although we are only asking you to show that there exists a way to sample to achieve full rank with enough samples taken, it turns out to be true that the F_ℓ matrix will be full rank as long as $n = \binom{D+\ell}{\ell}$ "generic" points are chosen.

Hint: Leverage earlier parts of this problem if you can.

We can choose $\vec{x}_1 = (x_{11}, x_{12}, \cdots, x_{1l})^T$ s.t. $x_{11} \leqslant x_{12} \leqslant \cdots \leqslant x_{1l}$

And $\vec{x}_2 = (x_{21}, x_{22}, \cdots, x_{2l})^T$ s.t. $x_{1l} < x_{21} \leqslant x_{22} \leqslant \cdots \leqslant x_{2l}$

$\cdots$

And $\vec{x}_n = (x_{n1}, x_{n2}, \cdots, x_{nl})^T$ s.t. $x_{n-1,l} < x_{n1} \leqslant x_{n2} \leqslant \cdots \leqslant x_{nl}$

We get

$$F_l = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1l} & \cdots & x_{11}^D & \cdots & x_{11}x_{12}\cdots x_{1l} \\ 1 & x_{21} & \cdots & x_{2l} & \cdots & x_{21}^D & \cdots & x_{21}x_{22}\cdots x_{2l} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & \cdots & x_{nl} & \cdots & x_{n1}^D & \cdots & x_{n1}x_{n2}\cdots x_{nl} \end{bmatrix}_{n \times n}$$

We can set

$$\vec{x_1} = (x_{11}, 0, \cdots, 0)^T$$
$$\vec{x_2} = (0, x_{22}, 0, \cdots, 0)^T$$
$$\vdots$$
$$\vec{x_l} = (0, \cdots, 0, x_{n2})^T$$
$$\vec{x_{l+1}} = (x_{l+1,1}, x_{l+1,2}, 0, \cdots, 0)^T$$

$$\vdots$$

$$\vec{x_{2l}} = (x_{l+1,1}, 0, \cdots, 0, x_{l+1,2})^T$$

$$\vdots$$

$$\vec{x_n} = (x_{n1}, x_{n2}, \cdots, x_{nl})^T$$

where $x_{ij} \neq 0$

We can rearrange F_l, s.t.

$$F_l = \begin{bmatrix} 1 & x_{11} & \cdots & x_{11}^D & \cdots & x_{1l}^D & \cdots & x_{11}x_{12}\cdots x_{1l} \\ 1 & x_{21} & \cdots & x_{21}^D & \cdots & x_{2l}^D & \cdots & x_{21}x_{22}\cdots x_{2l} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & \cdots & x_{n1}^D & \cdots & x_{nl}^D & \cdots & x_{n1}x_{n2}\cdots x_{nl} \end{bmatrix}_{n\times n}$$

$$= \begin{bmatrix} 1 & x_{11} & \cdots & x_{11}^D & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & 0 & \cdots & 0 & x_{21} & \cdots & x_{2l}^D & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & \cdots & x_{n1}^D & \cdots & \cdots & x_{n2}^D & \cdots x_{nl}^D & \cdots & x_{n1}x_{n2}\cdots x_{nl} \end{bmatrix}_{n\times n}$$

$$= \begin{bmatrix} 1 & x_{11} & \cdots & x_{11}^D & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & x_{21} & \cdots & x_{21}^D & 0 & \cdots & 0 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} + \sum_{i=1}^{l+1} x_{i1} & \cdots & x_{n1}^D + \sum_{i=1}^{l+1} x_{i1}^D & x_{n2} - x_{22} & \cdots & x_{n2}^D - x_{22}^D & \cdots x_{nl}^D - x_{ll}^D & \cdots & x_{n1}x_{n2}\cdots x_{nl} \end{bmatrix}_{n\times n}$$

$$= \begin{bmatrix} F_{(l+1)\times(l+1)} & \mathbf{0} \\ A & B \end{bmatrix}_{n\times n}$$

$$|F_l| = |F_{l\times l}||B|$$

$$B = \begin{bmatrix} x_{l+2,1}x_{l+2,2} & 0 & \cdots & 0 & x_{l+2,1}x_{l+2,2}^2 & \cdots & 0 \\ 0 & x_{l+3,1}x_{l+3,3} & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n,1}x_{n,2} & \cdots & \cdots & \cdots & \cdots & \cdots & x_{n1}x_{n2}\cdots x_{nl} \end{bmatrix}$$

$$|B| = x_{l+2,1}x_{l+2,2}\cdots x_{n,n-1}x_{nn} \begin{vmatrix} 1 & 0 & \cdots & 0 & x_{l+2,2} & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n,1}x_{n,2} & \cdots & \cdots & \cdots & \cdots & \cdots & x_{n1}x_{n2}\cdots x_{nl} \end{vmatrix}$$

$\because$ $|F_{(l+1)\times(l+1)}| \neq 0$ iff $x_{11}, x_{21}, \cdots, x_{l+1,1}$ are different from each other. $|B| \neq 0$ iff $x_{ij} \neq x_{pq}$

$\therefore$ when $x_{ij} \neq 0$ and $x_{ij} \neq x_{pq}$, F_l is full of column rank

4 Polynomials and approximation

For a p-times differentiable function $f : \mathbb{R} \to \mathbb{R}$, the Taylor series expansion of order $m \leq p-1$ about the point x_0 is given by

$$f(x) = \sum_{i=0}^{m} \frac{1}{i!} f^{(i)}(x_0)(x-x_0)^i + \frac{1}{(m+1)!} f^{(m)}(a(x))(x-x_0)^{m+1}. \tag{3}$$

Here, $f^{(m)}$ denotes the mth derivative of the function f, and $a(x)$ is some value between x_0 and x.

The last term of this expansion is tyically referred to as the approximation-error term when approximating $f(x)$ by an m-th degree polynomial. For functions f whose derivatives are bounded in the neighborhood of interest, if we have $|f^{(m)}(x)| \leq T$ for $x \in (x_0 - s, x_0 + s)$, then we know that for $x \in (x_0 - s, x_0 + s)$ that $|f(x) - \sum_{i=0}^{m} \frac{1}{i!} f^{(i)}(x_0)(x-x_0)^i| \leq \frac{T(x-x_0)^{m+1}}{(m+1)!}$.

(a) **Compute the 2nd, 3rd, and 4th order Taylor approximation of the following functions about the point $x_0 = 0$. Also bound the error of approximation at $x = 3$.**

i) e^x

$$\begin{aligned}
f(x) &= e^x \\
f^{(i)}(x) &= e^x \qquad \forall i \in \mathbb{N} \\
f(x) &= \sum_{i=0}^{2} \frac{f^{(i)}(0)}{i!} x^i + \frac{f^{(3)}(\xi_2)}{3!} x^3 \\
&= 1 + x + \frac{1}{2}x^2 + \frac{f^{(3)}(\xi_2)}{6} x^3 = \frac{9e^3}{2} \\
\left| \frac{f^{(3)}(\xi_2)}{3!} 3^3 \right| &\leqslant \frac{e^3 3^3}{6}
\end{aligned}$$

$$\begin{aligned}
f(x) &= \sum_{i=0}^{3} \frac{f^{(i)}(0)}{i!} x^i + \frac{f^{(4)}(\xi_3)}{4!} x^4 \\
&= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{f^{(4)}(\xi_3)}{24} x^4 \\
\left| \frac{f^{(4)}(\xi_3)}{24} 3^4 \right| &\leqslant \frac{e^3 3^4}{24} = \frac{27e^3}{8} \\
f(x) &= \sum_{i=0}^{4} \frac{f^{(i)}(0)}{i!} x^i + \frac{f^{(5)}(\xi_4)}{5!} x^4 \\
&= 1 + x + \frac{1}{2}(x-x_0)^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{f^{(5)}(\xi_4)}{120} x^5 \\
\left| \frac{f^{(5)}(\xi_4)}{120} 3^3 \right| &\leqslant \frac{e^3 3^5}{120} = \frac{81e^3}{40}
\end{aligned}$$

where ξ_2, ξ_3, ξ_4 are between 0 and x.

ii) $\sin x$

$$f(x) = \sin x$$

$$f^{(i)}(x) = \begin{cases} \cos x, & i = 4k+1 \\ -\sin x, & i = 4k+2 \\ -\cos x, & i = 4k+3 \\ \sin x, & i = 4k \end{cases} \qquad k \in \mathbb{N}$$

$$f^{(i)}(0) = \begin{cases} 1, & i = 4k+1 \\ 0, & i = 4k+2, 4k \\ -1, & i = 4k+3 \end{cases} \qquad k \in \mathbb{N}$$

$$f(x) = \sum_{i=0}^{2} \frac{f^{(i)}(0)}{i!} x^i + \frac{f^{(3)}(\xi_2)}{3!} x^3$$

$$= x + \frac{f^{(3)}(\xi_2)}{6} x^3$$

$$\left| \frac{f^{(3)}(\xi_2)}{3!} 3^3 \right| \leqslant \frac{3^3}{6} = \frac{9}{2}$$

$$f(x) = \sum_{i=0}^{3} \frac{f^{(i)}(0)}{i!} x^i + \frac{f^{(5)}(\xi_3)}{5!} x^5$$

$$= x - \frac{1}{6} x^3 + \frac{f^{(5)}(\xi_3)}{120} x^5$$

$$\left| \frac{f^{(5)}(\xi_3)}{120} 3^5 \right| \leqslant \frac{3^5}{120} = \frac{81}{40}$$

$$f(x) = \sum_{i=0}^{4} \frac{f^{(i)}(0)}{i!} x^i + \frac{f^{(5)}(\xi_4)}{5!} x^4$$

$$= x - \frac{1}{6} x^3 + \frac{f^{(5)}(\xi_4)}{120} x^5$$

$$\left| \frac{f^{(5)}(\xi_4)}{120} 3^3 \right| \leqslant \frac{3^5}{120} = \frac{81}{40}$$

where ξ_2, ξ_3, ξ_4 are between 0 and x.

(b) Let us say we would like an accurate polynomial approximation of the functions in part (a) for all $x \in [-3,3]$. In other words, we want a polynomial of degree D (call it ϕ_D) such that $|f(x) - \phi_D(x)| \leq \varepsilon$ for all $x \in [0,3]$. **How large does D need to be as a function of ε for such a guarantee for the two choices of $f(x)$ in part (a)?**

We pick

$$\phi_D(x) = \sum_{i=0}^{D} \frac{f^{(i)}(0)}{i!} x^i$$

Then (a)

$$\begin{aligned}
|f(x) - \phi_D(x)| &= \left| \frac{f^{(D)}(\xi_D)}{(D+1)!} x^{D+1} \right| \\
&= \frac{1}{(D+1)!} e^{\xi_D} x^{D+1} \\
&\leqslant \frac{e^3 3^{D+1}}{(D+1)!} \\
&\approx \frac{1}{\sqrt{2\pi(D+1)}} \left(\frac{3e}{D+1} \right)^{D+1} \\
&\leqslant \frac{1}{\sqrt{4\pi}} \left(\frac{3e}{D+1} \right)^{D+1} \\
&\leqslant \varepsilon
\end{aligned}$$

i.e.

$$\begin{aligned}
(D+1)[\log_e(3e) - \log_e(D+1)] &\leqslant \log_e(2\sqrt{\pi}\varepsilon) \\
(D+1)[\log_e(3e) - \log_e(D+1)] &\leqslant (D+1)[\log_e(3e) - 1] \\
D &\geqslant \frac{\log_e(2\sqrt{\pi}\varepsilon)}{\log_e(3e) - 1} - 1
\end{aligned}$$

(b)

$$\begin{aligned}
|f(x) - \phi_D(x)| &= \left| \frac{f^{(D)}(\xi_D)}{(D+1)!} x^{D+1} \right| \\
&= \begin{cases} \frac{1}{(D+1)!} |\sin \xi_D| x^{D+1} \\ \frac{1}{(D+1)!} |\cos \xi_D| x^{D+1} \end{cases} \\
&\leqslant \frac{3^{D+1}}{(D+1)!}
\end{aligned}$$

Similarly,

$$D \geqslant \frac{\log_e(2\sqrt{\pi}\varepsilon)}{\log_e(3) - 1} - 1$$

(c) **What is** $\lim_{m\to\infty} \frac{(x-x_0)^{m+1}}{(m+1)!}$**?**

Conclude that a univariate polynomial of high enough degree can approximate any function that is sufficiently smooth. This is the power of using polynomial features, even when we don't know the underlying function f that is generating our data! This universal approximation property gives us some justification for using polynomial features. (Later, we will see that neural networks are also universal function approximators.)

Because $x_0 - x$ is constant, $\exists\, N \in \mathbb{N}$, s.t. $N = 2[x - x_0] + 1$.

Therefore when $m > N$, the asymptotic error is

$$\begin{aligned}\frac{(x-x_0)^{m+1}}{(m+1)!} &\leqslant \frac{(x-x_0)^N}{N!} \cdot \frac{1}{2^{m+1-N}} \\ &\to 0 \qquad (m \to \infty)\end{aligned}$$

It means that when $m \to \infty$, the error of the approximation will close to zero, i.e., we can use a univariate polynomial of high enough degree to approximate any function that is sufficiently smooth.

(d) Now let's extend this idea of approximating functions with polynomials to multivariable functions. The Taylor series expansion for a function $f(x,y)$ about the point (x_0, y_0) is given by

$$\begin{aligned}f(x,y) = f(x_0,y_0) + f_x(x_0,y_0)(x-x_0) + f_y(x_0,y_0)(y-y_0) + \\ \frac{1}{2!}[f_{xx}(x_0,y_0)(x-x_0)^2 + f_{xy}(x_0,y_0)(x-x_0)(y-y_0) + \\ f_{yx}(x_0,y_0)(x-x_0)(y-y_0) + f_{yy}(x_0,y_0)(y-y_0)^2] + \ldots\end{aligned} \tag{4}$$

where $f_x = \frac{\partial f}{\partial x}$, $f_y = \frac{\partial f}{\partial y}$, $f_{xx} = \frac{\partial^2 f}{\partial x^2}$, $f_{yy} = \frac{\partial^2 f}{\partial y^2}$, and $f_{xy} = \frac{\partial^2 f}{\partial x \partial y}$

As you can see, the Taylor series for multivariate functions quickly becomes unwieldy after the second order. Let's try to make the series a little bit more manageable. **Using matrix notation, write the expansion for a function of two variables in a more compact form up to the second order terms where $f(\vec{x}) = f(x,y)$ with $\vec{x} = [x,y]^T$ and $\vec{x}_0 = [x_0, y_0]$. Clearly define any additional vectors and matrices that you use.**

$$\begin{aligned}f(\vec{x}) &= f(x_0,y_0) + f_x(x_0,y_0)(x-x_0) + f_y(x_0,y_0)(y-y_0) + \\ &\quad \frac{1}{2!}[f_{xx}(x_0,y_0)(x-x_0)^2 + f_{xy}(x_0,y_0)(x-x_0)(y-y_0) + \\ &\quad f_{yx}(x_0,y_0)(x-x_0)(y-y_0) + f_{yy}(x_0,y_0)(y-y_0)^2] + \cdots \\ &= f(\vec{x}_0) + [\nabla f(\vec{x}_0)]^T(\vec{x}-\vec{x}_0) + \frac{1}{2!}[\vec{x}-\vec{x}_0]^T \nabla^2 f(\vec{x}_0)[\vec{x}-\vec{x}_0] + \cdots \\ &= \sum_{i=0}^{\infty} \frac{[(\vec{x}-\vec{x}_0)^T\nabla]^i}{i!} f(\vec{x}) \\ &= \sum_{i=0}^{m} \frac{[(\vec{x}-\vec{x}_0)^T\nabla]^i}{i!} f(\vec{x}) + \frac{[(\vec{x}-\vec{x}_0)^T\nabla]^{m+1}}{(m+1)!} f(\vec{\xi})\end{aligned}$$

where $\nabla = \begin{pmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{pmatrix}$, $\vec{\xi} = (\xi_1, \xi_2)^T$, ξ_1 is between x and x_0, ξ_2 is between y and y_0.

(e) To see that we can do universal approximation, first just consider that we want to approximate the function in a single straight line path. For this problem, we will use the function

$f(\vec{x}) = e^x \sin y$

where $\vec{x} = [x, y]^T$. We're interested in the path $\vec{x}(t) = [\frac{1}{\sqrt{2}}t, \frac{1}{\sqrt{2}}t]^T$. **Write the 3rd order Taylor expansion of $f(\vec{x}(t))$ for the variable t about the point $t_0 = 0$.** Use the chain rule.

$$
\begin{aligned}
f_x(\vec{x}) = f_{xx}(\vec{x}) = f_{xxx}(\vec{x}) &= e^x \sin y \\
f_{yyy}(\vec{x}) &= -e^x \cos y \\
f_y(\vec{x}) = f_{yx}(\vec{x}) &= f_{yxx}(\vec{x}) \\
= f_{xy}(\vec{x}) = f_{xxy}(\vec{x}) = f_{xyx}(\vec{x}) &= e^x \cos y \\
f_{xyy}(\vec{x}) = f_{yy}(\vec{x}) = f_{yxy}(\vec{x}) = f_{yyx}(\vec{x}) &= -e^x \sin y
\end{aligned}
$$

Because

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}(e^x \sin y) &= \frac{1}{\sqrt{2}} e^x (\sin y + \cos y) \\
\frac{\mathrm{d}}{\mathrm{d}t}(e^x \cos y) &= \frac{1}{\sqrt{2}} e^x (\cos y - \sin y) \\
f(\vec{x}(t)) &= f(0,0) + f_x(0,0)x + f_y(0,0)y + \\
&\quad \frac{1}{2!}[f_{xx}(0,0)x^2 + f_{xy}(0,0)xy + \\
&\quad f_{yx}(0,0)xy + f_{yy}(0,0)y^2] + \cdots \\
&= \frac{1}{\sqrt{2}}x + \frac{1}{\sqrt{2}}y + \frac{1}{2\sqrt{2}}(x^2 + 2xy - y^2) \\
&\quad + \frac{1}{6\sqrt{2}}(x^3 + 3x^2 y - xy^2 - y^3) + \frac{1}{24\sqrt{2}}(\varepsilon_1^4 + 4\varepsilon_1^3\varepsilon_2 - 6\varepsilon_1^2\varepsilon_2^2 - 4\varepsilon_1\varepsilon_2^3 + \varepsilon_2^4)
\end{aligned}
$$

(f) Similar to part **(b)**, **determine the degree D for the polynomial $\phi_D(\vec{x}(t))$ from the Taylor series about the point $t_0 = 0$ such that $|f(\vec{x}(t)) - \phi_D(\vec{x}(t))| \leq \varepsilon$ on the interval $t \in [0, 3]$ for the function from the previous part.**

$$
\begin{aligned}
|f(\vec{x}(t)) - \psi_D(\vec{x}(t))| &= \left| \frac{[(\vec{x}-\vec{x}_0)^T\nabla]^{D+1}}{(D+1)!} f(\vec{\xi}) \right| \\
&\leqslant \frac{3^{D+1}}{\sqrt{2}(D+1)!} \left| \sum_{i=0}^{D+1} \binom{D+1}{i} \nabla_x^i \nabla_y^{D+1-i}(\vec{\xi}) \right| \\
&\leqslant \frac{3^{D+1}}{\sqrt{2}(D+1)!} \sum_{i=0}^{D+1} \binom{D+1}{i} \\
&= \frac{6^{D+1}}{\sqrt{2}(D+1)!} \\
&\leqslant \varepsilon
\end{aligned}
$$

Similarly,

$$
D \geqslant \frac{\log_e(2\sqrt{2\pi}\varepsilon)}{log_e(6)-1} - 1
$$

(g) BONUS: **Sketch how the argument above can be extended to show that multivariate polynomials are universal function approximators for sufficiently smooth functions of many variables.**

$\forall\, \vec{x} \in \mathbb{R}^n$

$$
\begin{aligned}
f(\vec{x}) &= \sum_{i=0}^{\infty} \frac{[(\vec{x}-\vec{x}_0)^T\nabla]^i}{i!} f(\vec{x}) \\
&= \sum_{i=0}^{m} \frac{[(\vec{x}-\vec{x}_0)^T\nabla]^i}{i!} f(\vec{x}) + \frac{[(\vec{x}-\vec{x}_0)^T\nabla]^{m+1}}{(m+1)!} f(\vec{\xi})
\end{aligned}
$$

When $f(\vec{x})$ is sufficiency smooth and $\|\nabla^{m+1} f(\vec{x})\|_\infty \leqslant T$, then we can use m-degree polynomial $\sum_{i=0}^{m} \frac{[(\vec{x}-\vec{x}_0)^T\nabla]^i}{i!} f(\vec{x})$ to estimate $f(\vec{x})$.

And the error is bounded by

$$
\begin{aligned}
\left| \frac{[(\vec{x}-\vec{x}_0)^T\nabla]^{m+1}}{(m+1)!} f(\vec{\xi}) \right| &\leqslant \frac{n\|\vec{x}-\vec{x}_0\|_\infty T}{(m+1)!} \sum_{i_1,\cdots,i_n} \binom{m+1}{i_1} \binom{m+1-i_1}{i_2} \cdots \binom{m+i-\sum_{j=1}^{n-1} i_j}{i_n} \\
&= \frac{n\|\vec{x}-\vec{x}_0\|_\infty T n^{m+1}}{(m+1)!} \\
&\leqslant \varepsilon
\end{aligned}
$$

Similarly,

$$
D \geqslant \frac{\log_e\left(\frac{2\sqrt{\pi}\varepsilon}{nT\|\vec{x}-\vec{x}_0\|_\infty}\right)}{log_e(n)-1} - 1
$$

So given $\varepsilon > 0$, we can choose $m \in \mathbb{N}$ to approximate the real function.

5 Jaina and her giant peaches

In another alternative universe, Jaina is a mage testing how long she can fly a collection of giant peaches. She has n training peaches – with masses given by $x_1, x_2, \ldots x_n$ – and flies all the peaches once to collect training data. The experimental flight time of peach i is given by y_i. She believes that the flight time is well approximated by a polynomial function of the mass, and her goal is to fit a polynomial of degree D to this data. Include all text responses and plots in your write-up.

(a) **Show how Jaina's problem can be formulated as a linear regression problem.**

Given D - the degree of the approximating polynomial function. What we need finally is the coefficients of the polynomial because we assume that

$$\begin{aligned}\hat{y} &= a_0 + a_1 x + \cdots + a_D x^D \\ &= \begin{pmatrix} a_0 & a_1 & \cdots & a_D \end{pmatrix} \begin{pmatrix} 1 \\ x \\ \vdots \\ x^D \end{pmatrix} \\ &= A\vec{x}\end{aligned}$$

So when we fit the training data, it will look like

$$\begin{aligned}\begin{pmatrix} \hat{y_1} \\ \hat{y_2} \\ \vdots \\ \hat{y_n} \end{pmatrix} &= \begin{pmatrix} 1 & x_1 & \cdots & x_1^D \\ 1 & x_2 & \cdots & x_2^D \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & \cdots & x_n^D \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_D \end{pmatrix} \\ &= XA\end{aligned}$$

Our goal is to let $\vec{\hat{y}}$ close to $\vec{y} = Y$, i.e.

$$A = \arg\min_A \|XA - Y\|_2^2$$

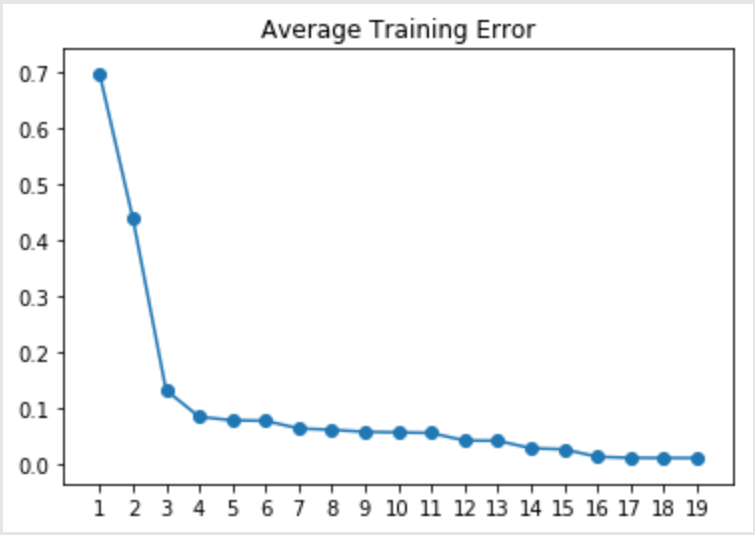
a linear regression problem.

(b) You are given data of the masses $\{x_i\}_{i=1}^n$ and flying times $\{y_i\}_{i=1}^n$ in the "x_train" and "y_train" keys of the file 1D_POLY.MAT, respectively, with the masses centered and normalized to lie in the range $[-1,1]$. **Write a routine to do a least-squares fit (taking care to include a constant term) of a polynomial function of degree D to the data.** Letting f_D denote the fitted polynomial, **plot the average training error $R(D) = \frac{1}{n}\sum_{i=1}^n (y_i - f_D(x_i))^2$ against D in the range $D \in \{1,2,3,\ldots,n-1\}$.**

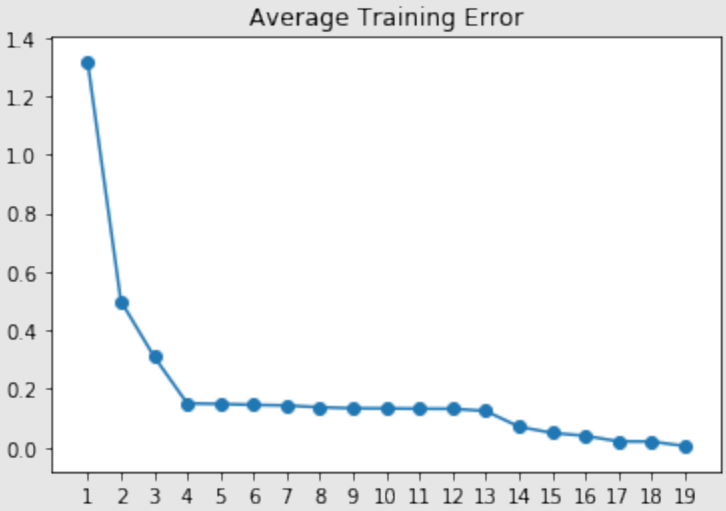
(1) gennerate X. The first column is $\vec{1}^T$, the second column is $\vec{x}^T$, the third column is $(x_1^2, \cdots, x_n^2)^T \cdots$ the $(D+1)^{th}$ column is $(x_1^D, \cdots, x_n^D)^T$.

(2) use $np.linalg.solve()$ to get the solution y of $X^T XA = X^T y$.

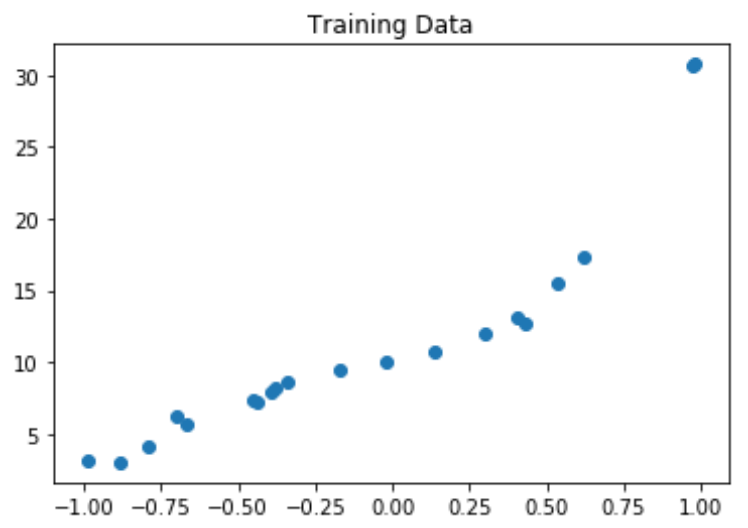
Code is given in .ipynb.



Bonus



(c) **How does the average training error behave as a function of D, and why? What happens if you try to fit a polynomial of degree n with a standard matrix inversion method?**



The average training error decreases when D increases. It is because any function that is smooth enough can be well approximated by a polynomial with enough degree. So when D increases, the polynomial can approximate the real function better.

When fitting a polynomial of degree n with a standard matrix inversion method, it fails because now

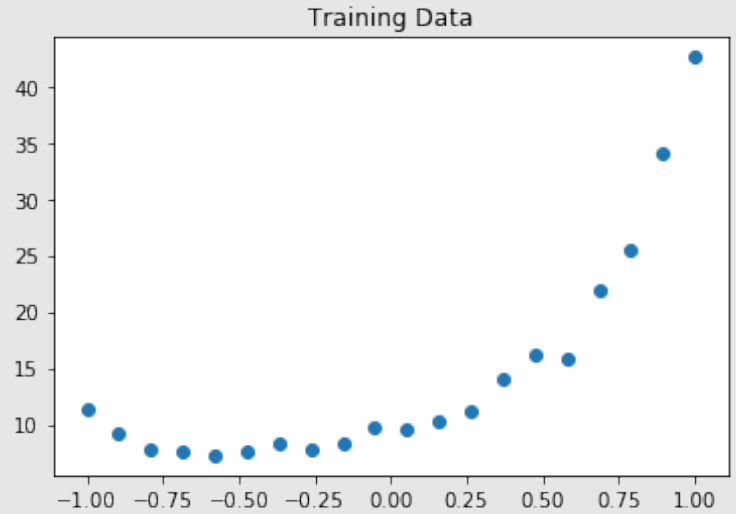
$$rank(X_{n\times(n+1)}) \leqslant n$$

and

$$rand[(X^TX)_{(n+1)\times(n+1)}] \leqslant n$$

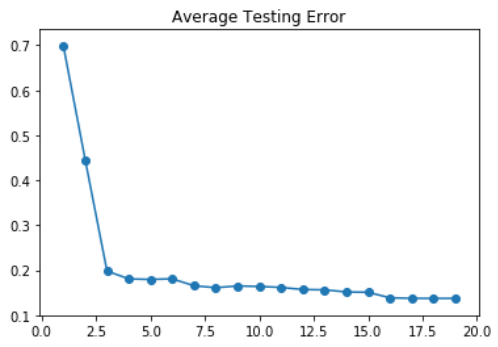
i.e. X^TX is not full rank. So X^TX is non-invertible.

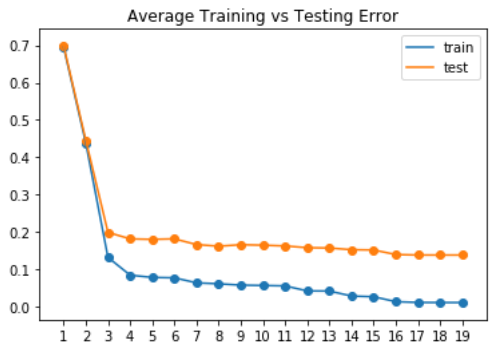
Bonus



And the answer is as above.

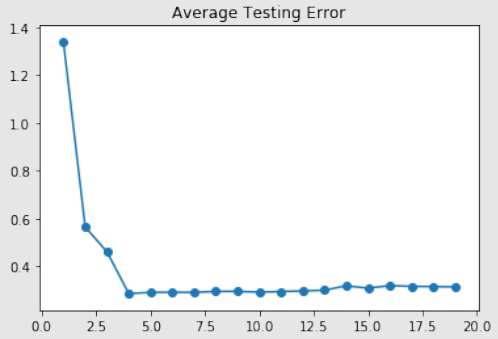
(d) Jaina has taken Mystical Learning 189, and so decides that she needs to run another experiment before deciding that her prediction is true. She runs another fresh experiment of flight times using the same peaches, to obtain the data with key "y_fresh" in 1D_POLY.MAT. Denoting the fresh flight time of peach i by $\tilde{y}_i$, **plot the average error $\tilde{R}(D) = \frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - f_D(x_i))^2$ for the same values of D as in part (b) using the polynomial approximations f_D also from the previous part. How does this plot differ from the plot in (b) and why?**

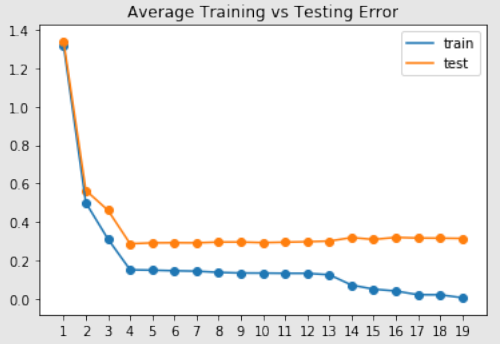




We can see that when $D = 1$ or 2, the errors of training and testing set are close. However, when $D \geqslant 3$ the average testing error is much larger than training error, i.e. it may be overfitting.

Bonus





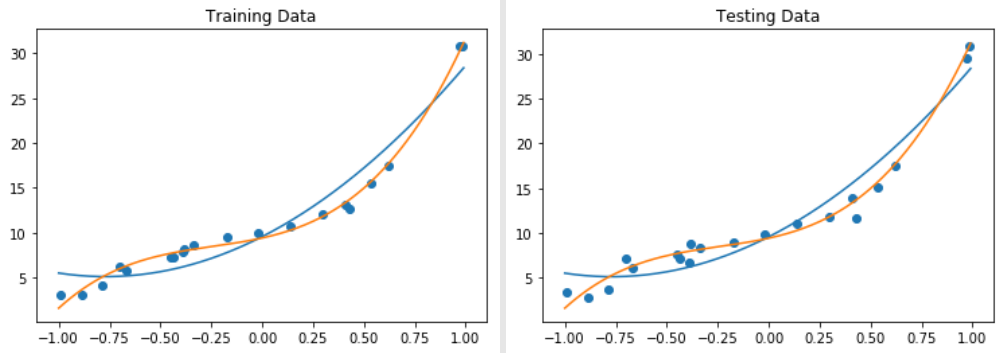
The average testing error goes down when $D \leqslant 4$ and goes up when $D > 4$.

(e) **How do you propose using the two plots from parts (b) and (d) to "select" the right polynomial model for Jaina?**

When $D = 1$ or 2, the errors of training and testing set are close. However, when $D > 3$ the average testing error is much larger than training error, i.e. it may be overfitting. Therefore, I would select a polynomial of 2 or 3 degree.

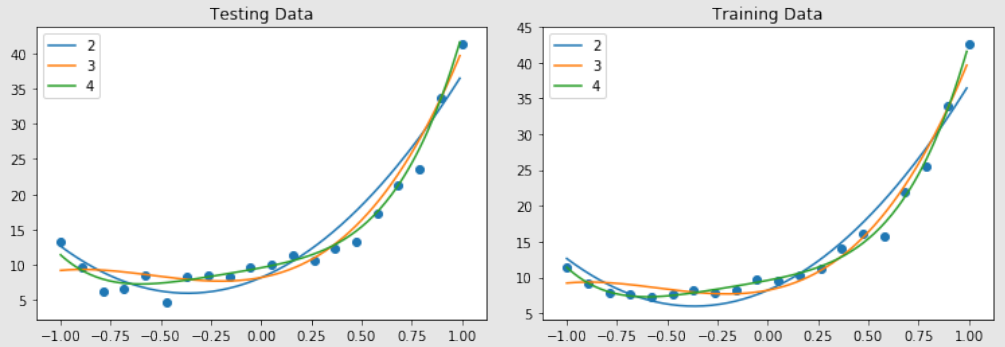
$$f_2(x) = 9.55793996 + 11.58152382x + 7.51419184x^2$$

$$f_3(x) = 9.4055568 + 5.0249049x + 7.2323537x^2 + 10.03785197x^3$$



From the plots, I think the $3-degree$ polynomial is better.

Bonus



$$f_2(x) = 8.20077407] + 12.13018879x + 16.56561306x^2$$

$$f_3(x) = 8.20077407 + 5.45186371x + 16.56561306x^2 10.10425546 + x^3$$

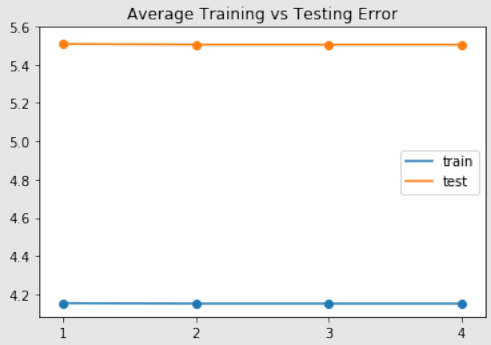
$$f_4(x) = 9.58634937 + 5.45186371x + 3.87983398x^2 + 10.10425546x^3 + 13.50335457x^4$$

The average testing error goes down first and goes up later. So the best choice will be $D = 4$.

(f) Jaina has a new hypothesis – the flying time is actually a function of the mass, smoothness, size, and sweetness of the peach, and some multivariate polynomial function of all of these parameters. The data in POLYNOMIAL_REGRESSION_SAMPLES.MAT (100000×5) with columns corresponding to the 5 attributes of the peach. **Use 4-fold cross-validation to decide which of $D \in \{1,2,3,4\}$ is the best fit for the data provided**. For this part, compute the polynomial coefficients via ridge regression with penalty $\lambda = 0.1$, instead of ordinary least squares.

By 4-fold cross-validation, the errors are

λ :	0.05	0.1	0.15	0.2
$R_{train} =$	8.84330811	8.83752421	8.83743868	8.83743797
$R_{test} =$	15.26962938	15.262095	15.26206574	15.26206531

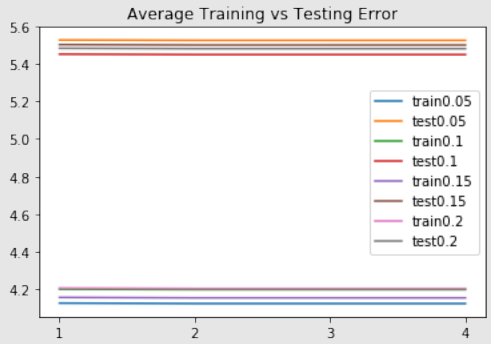


The difference between choices of D is small. So we can choose the smallest degree, i.e. $D = 1$.

(g) Now **redo the previous part, but use 4-fold cross-validation on all combinations of $D \in \{1,2,3,4\}$ and $\lambda \in \{0.05, 0.1, 0.15, 0.2\}$** - this is referred to as a grid search. **Find the best D and λ that best explains the data using ridge regression.**

By 4-fold cross-validation, the errors are

$$\begin{array}{rcl} \lambda: & \begin{matrix} 0.05 & 0.1 & 0.15 & 0.2 \end{matrix} \\ R_{train} = & \begin{bmatrix} 4.17715134 & 4.16155954 & 4.23717952 & 4.15828094 \\ 4.17498689 & 4.15889791 & 4.23502513 & 4.15571776 \\ 4.17494433 & 4.15887629 & 4.23501107 & 4.15570349 \\ 4.17494403 & 4.15887606 & 4.23501093 & 4.15570335 \end{bmatrix} \\ R_{test} = & \begin{bmatrix} 5.45819726 & 5.47804897 & 5.43555745 & 5.47684755 \\ 5.45566785 & 5.47695999 & 5.43362865 & 5.4750144 \\ 5.45566975 & 5.47695483 & 5.43362232 & 5.47501832 \\ 5.4556696 & 5.47695489 & 5.43362226 & 5.47501829 \end{bmatrix} \end{array}$$



We can see that the difference between choices of D is small. However, $\lambda = 0.1$ gives the best testing error.

So the best choice should be $D = 1$, $\lambda = 0.1$.

6 Your Own Question

Write your own question, and provide a thorough solution.

This problem should show your understanding of a key concept in the class.

Writing your own problems is a very important way to really learn material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.

Problem: What is the defference between ridge regression and Lasso regression?

Solutions:

Ridge regression adds a $L-2$ constrain to objective function while Lasso regression adds a $L-1$ term.

The form of ridge regression is given in problem 2.

The form of Lasso regreesion is given by

$$w = \arg\min_{w} \|Xw - y\|_2^2 + \lambda \|w\|_1$$

By derivating, we get

$$(X^T X + \lambda \cdot sign(w))w = X^T y$$

where $sign(w) = \sum_{i=1}^{n} |w_i|$.

Pros of Lasso:

(1) It can work with problems that ridge regreesion can deal with.

(2) Because the constrain is stronger (there exists angular points in the hyperparameter space), w_i can be 0. So it can be used to choose variables.

Cons of Lasso:

(1) Hard to compute because of $|w|$. Usually need coordinate descent method or Least Angle Regression (LARS) to compute a optimal solution.

(2) It depends on the algorithm. So the choice of hyperparameters is important.

HW2

September 8, 2017

Old Data

1 (a)

2 (b)

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        import scipy.io as sio
```

Read data

```
In [2]: datamat3 = sio.loadmat("./hw02-data/1D_poly.mat")
        x = datamat3['x_train']
        y = datamat3['y_train']
        y_test = datamat3['y_fresh']
        N = len(x)
```

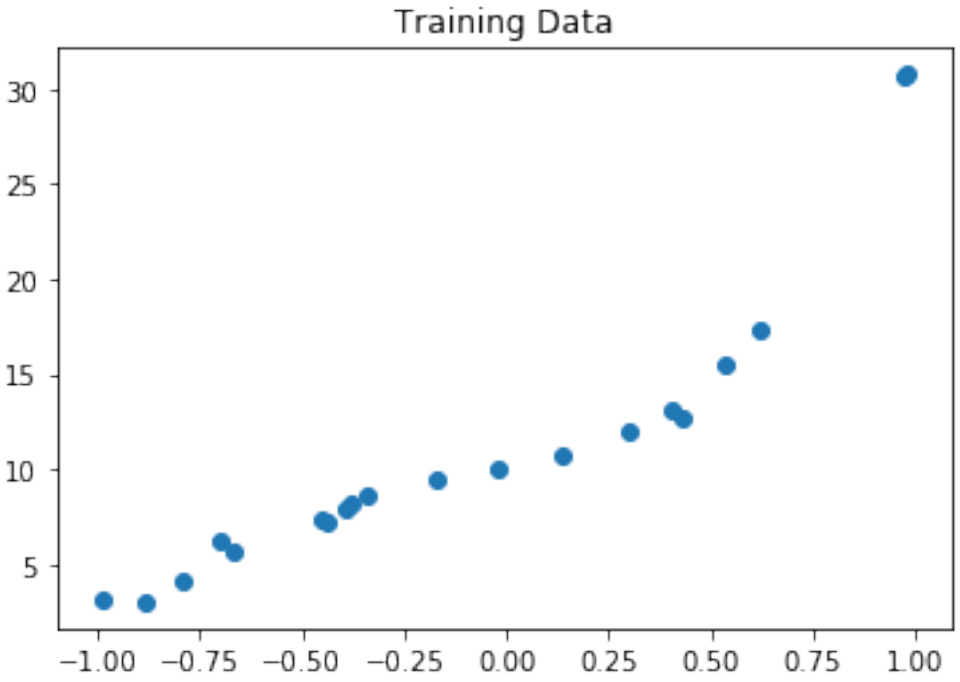
Regression function

```
In [3]: def regression(x,y,y_test,D):
            X = np.ones((len(x),1))
            for i in range(1,D+1):
                X = np.hstack((X,x**i))
            A = np.linalg.solve(X.T.dot(X),X.T.dot(y))
            yhat = X.dot(A)

            Rmean = np.linalg.norm(yhat-y)/len(x)
            Rmean_pred = np.linalg.norm(yhat-y_test)/len(x)

            return {'A':A,'X':X,'error':Rmean,'error_test':Rmean_pred}
```

3 (c)

Plot the training data

```
In [4]: fig = plt.figure()
        ax = fig.add_subplot(111)
        plt.title('Training Data')
        plt.scatter(x,y)
        plt.show()
```

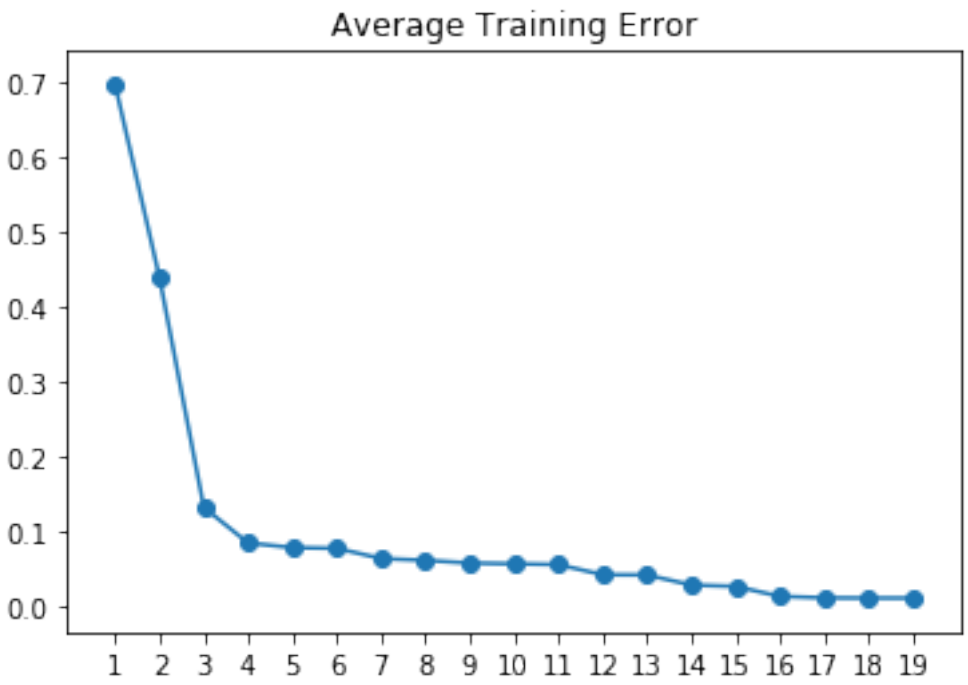


Regression

```
In [5]: result = []
        error = []
        error_test = []
        for i in range(1,N):
            result += [regression(x, y, y_test, i)]
            error += [result[i-1]['error']]
            error_test += [result[i-1]['error_test']]
```

Plot the average training data

```
In [6]: fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.set_xticks(np.arange(1,20,1))
        plt.title('Average Training Error')
        plt.plot(np.arange(1,20,1),error)
        plt.scatter(np.arange(1,20,1),error)

Out[6]: <matplotlib.collections.PathCollection at 0x113786860>
```



Choose n-degree polynomial makes training error much small

```
In [7]: nmodel = regression(x, y, y_test, 20)
        print(nmodel['error'])
```

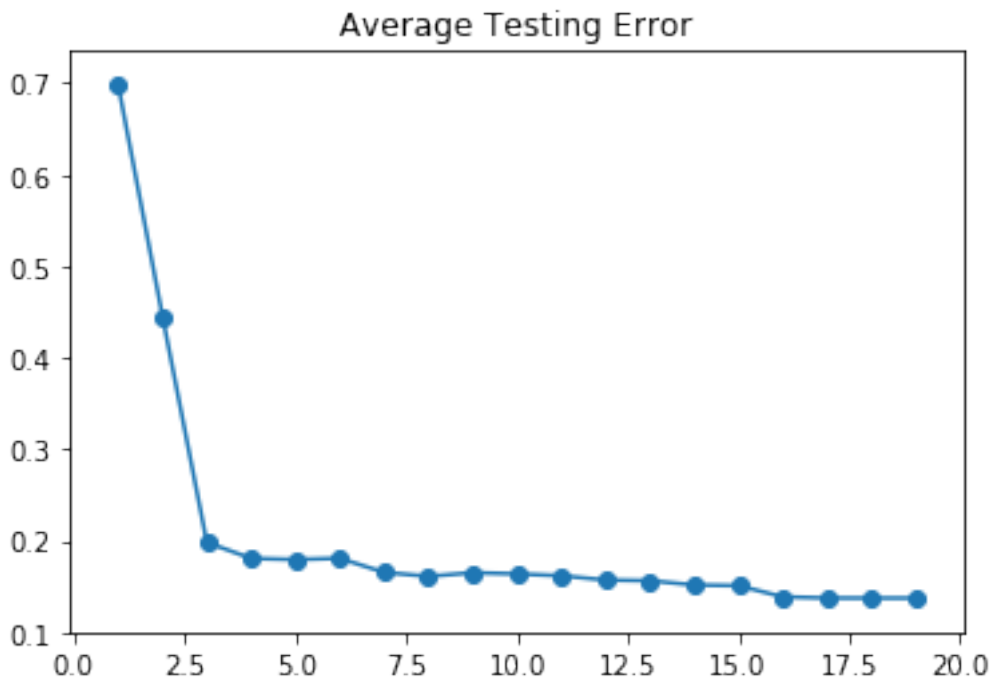
```
0.0116303988867
```

4 (d)

Plot the average testing error

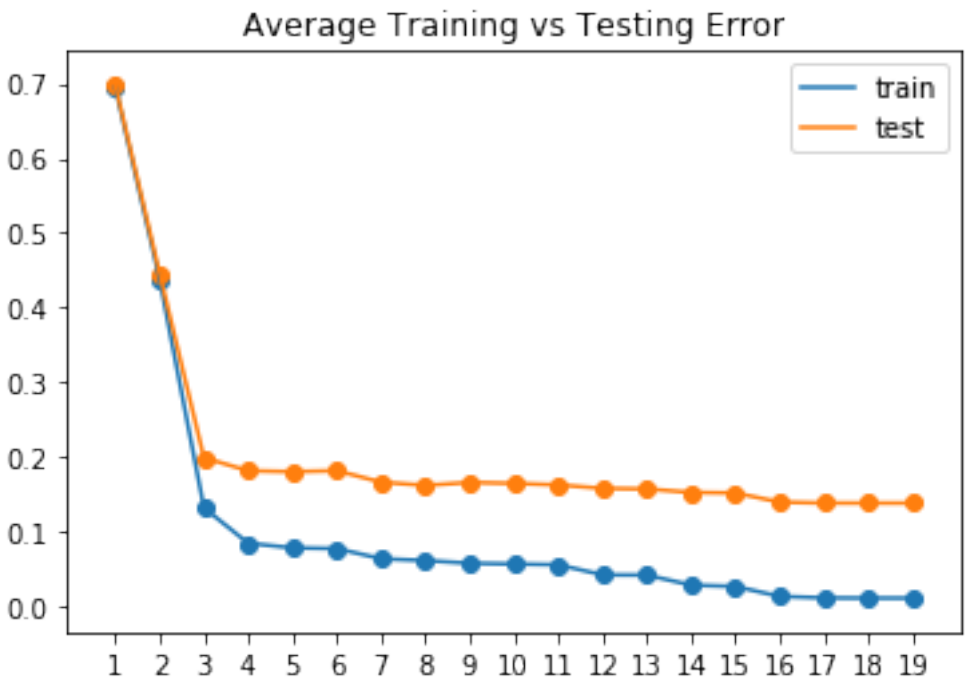
```
In [8]: ax = fig.add_subplot(111)
        ax.set_xticks(np.arange(1,20,1))
        plt.title('Average Testing Error')
        plt.plot(np.arange(1,20,1),error_test)
        plt.scatter(np.arange(1,20,1),error_test)
```

```
Out[8]: <matplotlib.collections.PathCollection at 0x11362bc88>
```



Plot the average traning error and average testing error

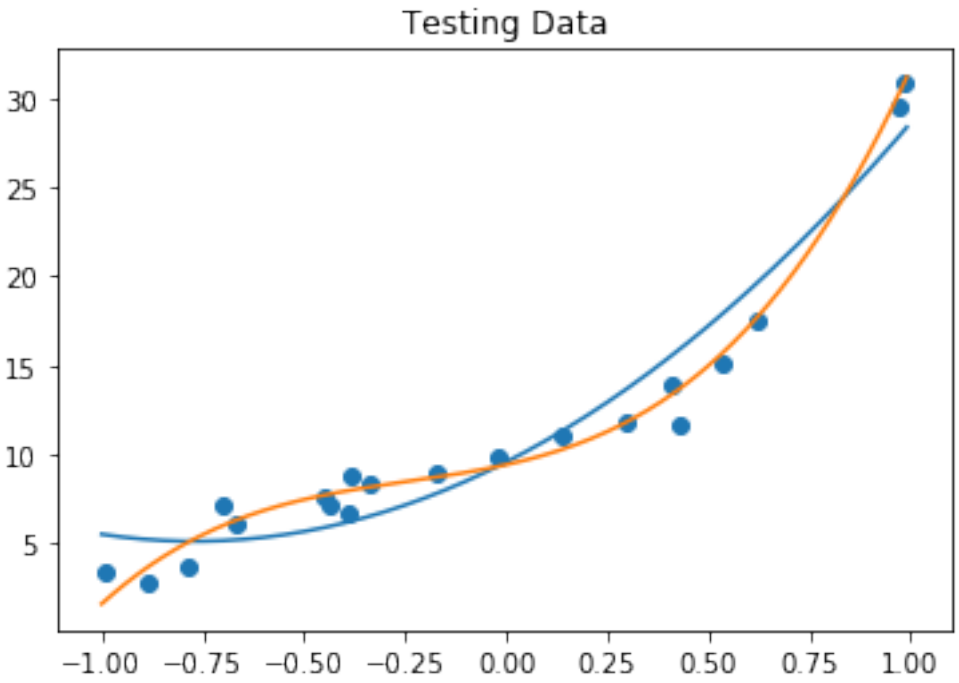
```
In [9]: fig2 = plt.figure()
        ax2 = fig2.add_subplot(111)
        plt.plot(np.arange(1,20,1),error,label='train')
        plt.scatter(np.arange(1,20,1),error)
        plt.plot(np.arange(1,20,1),error_test,label='test')
        plt.scatter(np.arange(1,20,1),error_test)
        ax2.set_xticks(np.arange(1,20,1))
        plt.legend()

        plt.title('Average Training vs Testing Error')

Out[9]: <matplotlib.text.Text at 0x1136f1e80>
```

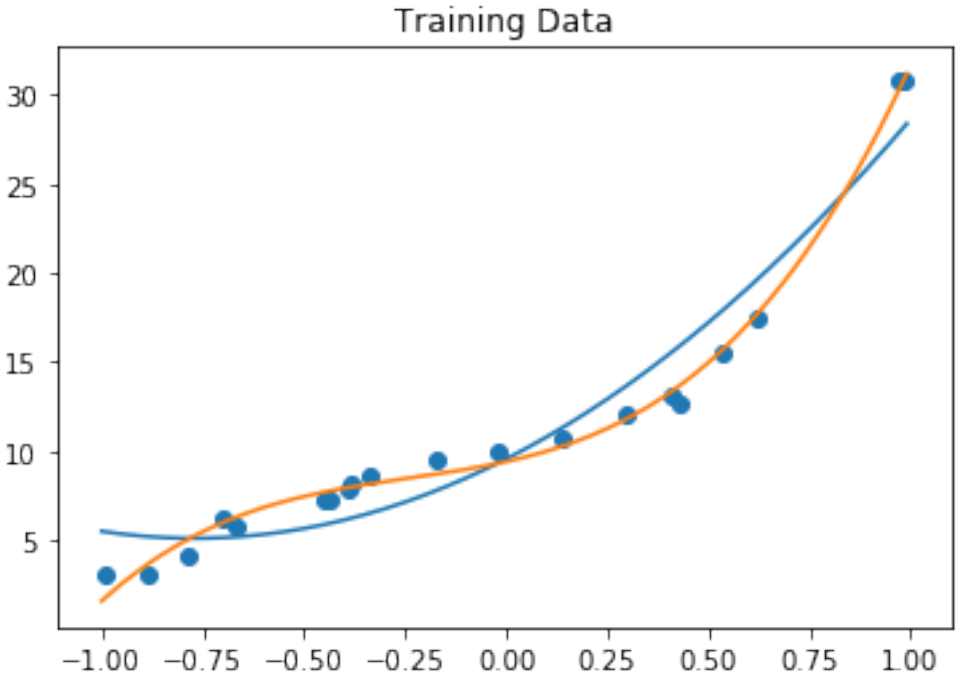


5 (e)

Plot the 2-degree and 3-degree polynomial

```
In [10]: fig3 = plt.figure()
         ax3 = fig3.add_subplot(111)
         xx = np.arange(-1,1,0.01)
         xx = xx.reshape(200,1)
         X3 = np.ones((len(xx),1))
         for i in range(1,3):
             X3 = np.hstack((X3,xx**i))
         plt.plot(xx,X3.dot(result[1]['A']))
         X4 = np.hstack((X3,xx**3))
         plt.plot(xx,X4.dot(result[2]['A']))
         plt.scatter(x,y_test)
         plt.title('Testing Data')

         fig4 = plt.figure()
         ax4 = fig4.add_subplot(111)
         plt.plot(xx,X3.dot(result[1]['A']))
         plt.plot(xx,X4.dot(result[2]['A']))
         plt.scatter(x,y)
         plt.title('Training Data')

Out[10]: <matplotlib.text.Text at 0x113adb710>
```






```
In [11]: print('coefficients of 2-degree polynomial:')
         print(result[1]['A'])
```

```
        print('coefficients of 3-degree polynomial:')
        print(result[2]['A'])
```

```
coefficients of 2-degree polynomial:
[[  9.55793996]
 [ 11.58152382]
 [  7.51419184]]
coefficients of 3-degree polynomial:
[[  9.4055568 ]
 [  5.0249049 ]
 [  7.2323537 ]
 [ 10.03785197]]
```

6 (f)

```
In [12]: import random
         import itertools
         datamat4 = sio.loadmat("./hw02-data/polynomial_regression_samples.mat")
         x2 = datamat4['x']
         y2 = datamat4['y']
```

```
In [13]: N = len(x2)
```

```
In [14]: def cross_validation(N):
             k = 4
             r = np.random.permutation(N)
             n = N // k

             data = [{'id_train':np.hstack((r[:n],r[2*n:])),'id_test':r[n:2*n]}]\
             + [{'id_train':np.hstack((r[:i*n],r[(i+1)*n:])),'id_test':r[i*n:(i+1)*n]} for i in
             + [{'id_train':r[3*n:],'id_test':r[:3*n]}]
             return data
```

```
In [15]: def regreession_multi(X,y,id_train,id_test,lamb):

             X_train = X[id_train,:]
             y_train = y[id_train]
             X_test = X[id_test,:]
             y_test = y[id_test]
             n = len(X_train.T)
             A = np.linalg.solve(X_train.T.dot(X_train)+lamb*np.identity(n),X_train.T.dot(y_trai
             yhat = X_train.dot(A)

             Rmean = np.linalg.norm(yhat-y_train)*1000/len(id_train)
             yhat_test = X_test.dot(A)
             Rmean_test = np.linalg.norm(yhat_test-y_test)*1000/len(id_test)
             return {'A':A,'error':Rmean,'error_test':Rmean_test}
```

In [16]:
```
lam = 0.1
error2 = np.zeros([4,1])
error_test2 = np.zeros([4,1])

n = 1
for _ in range(n):

    for data in cross_validation(N):
        X2 = np.ones((N,1))
        X2 = np.hstack((X2,x2))

        reg = regreession_multi(X2,y2,data['id_train'],data['id_test'],lam)
        error2[0] += reg['error']
        error_test2[0] += reg['error_test']
        for i in range(2,5):
            for j in itertools.combinations_with_replacement(np.arange(5), i):
                xn = np.ones((1,N))
                for k in j:
                    xn = xn*x2[:,k]
                X2 = np.hstack((X2,xn.reshape(N,1)))

            reg = regreession_multi(X2,y2,data['id_train'],data['id_test'],lam)
            error2[i-1] += reg['error']
            error_test2[i-1] += reg['error_test']
```

In [17]:
```
print(error2)
print(error_test2)
```

```
[[ 4.16840888]
 [ 4.16557283]
 [ 4.16552281]
 [ 4.16552245]]
[[ 5.46254864]
 [ 5.46031359]
 [ 5.46031335]
 [ 5.46031332]]
```

In [18]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(1,5,1),error2,label='train')
plt.scatter(np.arange(1,5,1),error2)
plt.plot(np.arange(1,5,1),error_test2,label='test')
plt.scatter(np.arange(1,5,1),error_test2)
ax.set_xticks(np.arange(1,5,1))
plt.legend()

plt.title('Average Training vs Testing Error')
```

Out[18]: <matplotlib.text.Text at 0x113b39a58>

In [19]:
```
error3 = np.zeros([4,4])
error_test3 = np.zeros([4,4])

lam_list = [0.05,0.1,0.15,0.2]
n = 1

for l in range(4):
    for _ in range(n):
        for data in cross_validation(N):
            X2 = np.ones((N,1))
            X2 = np.hstack((X2,x2))

            reg = regreession_multi(X2,y2,data['id_train'],data['id_test'],lam_list[l])
            error3[0,l] += reg['error']
            error_test3[0,l] += reg['error_test']
            for i in range(2,5):
                for j in itertools.combinations_with_replacement(np.arange(4), i):
                    xn = np.ones((1,N))
                    for k in j:
                        xn = xn*x2[:,k]
                    X2 = np.hstack((X2,xn.reshape(N,1)))

                reg = regreession_multi(X2,y2,data['id_train'],data['id_test'],lam_list
```

```
                        error3[i-1,l] += reg['error']
                        error_test3[i-1,l] += reg['error_test']
```

In [31]:
```
print(error3)
print(error_test3)
```

```
[[ 4.17715134  4.16155954  4.23717952  4.15828094]
 [ 4.17498689  4.15889791  4.23502513  4.15571776]
 [ 4.17494433  4.15887629  4.23501107  4.15570349]
 [ 4.17494403  4.15887606  4.23501093  4.15570335]]
[[ 5.45819726  5.47804897  5.43555745  5.47684755]
 [ 5.45566785  5.47695999  5.43362865  5.4750144 ]
 [ 5.45566975  5.47695483  5.43362232  5.47501832]
 [ 5.4556696   5.47695489  5.43362226  5.47501829]]
```

In [20]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
for i in range(4):
    plt.plot(np.arange(1,5,1),error3[:,i],label='train%s'%lam_list[i])

    plt.plot(np.arange(1,5,1),error_test3[:,i],label='test%s'%lam_list[i])

ax.set_xticks(np.arange(1,5,1))
plt.legend()

plt.title('Average Training vs Testing Error')
```

Out[20]: <matplotlib.text.Text at 0x1146326a0>

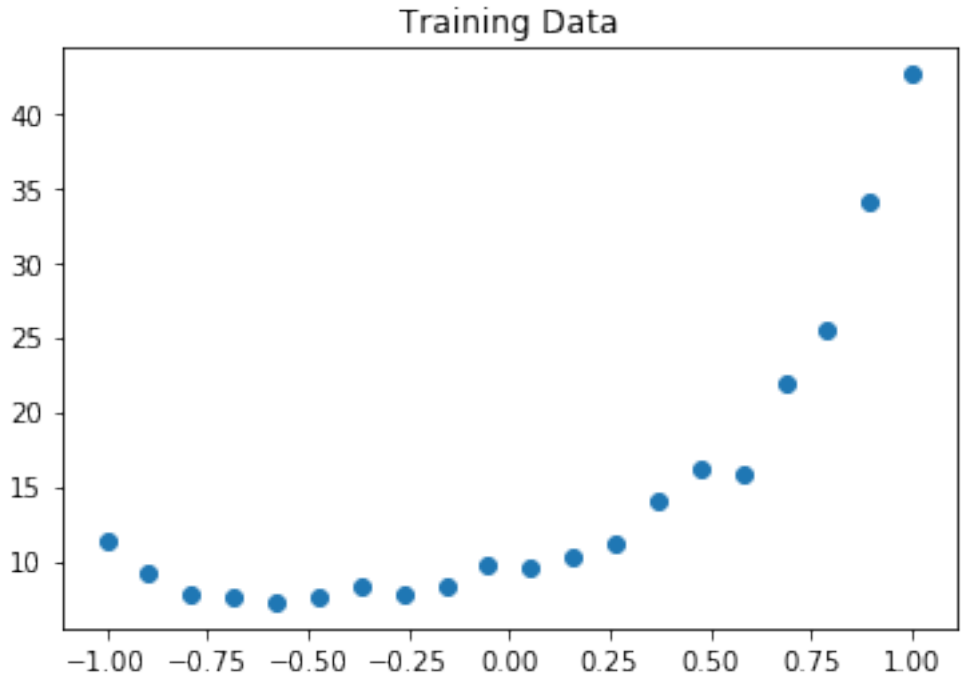
7 Bounus

New 1-D Data

```
In [21]: datamat5 = sio.loadmat("./hw02-data/1D_poly_new.mat")
         xn = datamat5['x_train']
         yn = datamat5['y_train']
         yn_test = datamat5['y_fresh']
         N = len(yn)
         xn = xn.reshape((N,1))

In [22]: datamat5

Out[22]: {'__globals__': [],
          '__header__': b'MATLAB 5.0 MAT-file Platform: posix, Created on: Thu Sep  7 14:29:37 2
          '__version__': '1.0',
          'x_train': array([[-1.        , -0.89473684, -0.78947368, -0.68421053, -0.57894737,
                  -0.47368421, -0.36842105, -0.26315789, -0.15789474, -0.05263158,
                   0.05263158,  0.15789474,  0.26315789,  0.36842105,  0.47368421,
                   0.57894737,  0.68421053,  0.78947368,  0.89473684,  1.        ]]),
          'y_fresh': array([[ 13.32997656],
                 [  9.69177878],
                 [  6.13008597],
                 [  6.59286006],
                 [  8.59227616],
                 [  4.68187364],
                 [  8.31154698],
                 [  8.50896101],
                 [  8.22909206],
                 [  9.74321854],
                 [  9.9737423 ],
                 [ 11.33524697],
                 [ 10.65433427],
                 [ 12.37383451],
                 [ 13.29269622],
                 [ 17.31711477],
                 [ 21.33661794],
                 [ 23.50094073],
                 [ 33.58199526],
                 [ 41.27664699]]),
          'y_train': array([[ 11.43517643],
                 [  9.15939136],
                 [  7.79121686],
                 [  7.64604858],
                 [  7.2842167 ],
```

```
[  7.64649196],
[  8.2666563 ],
[  7.86691196],
[  8.30315368],
[  9.72401144],
[  9.50953025],
[ 10.25750217],
[ 11.22305493],
[ 14.01833207],
[ 16.10368296],
[ 15.80407223],
[ 21.97631689],
[ 25.46609601],
[ 34.03048174],
[ 42.56554904]])}
```

In [23]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
plt.title('Training Data')
plt.scatter(xn,yn)
plt.show()
```

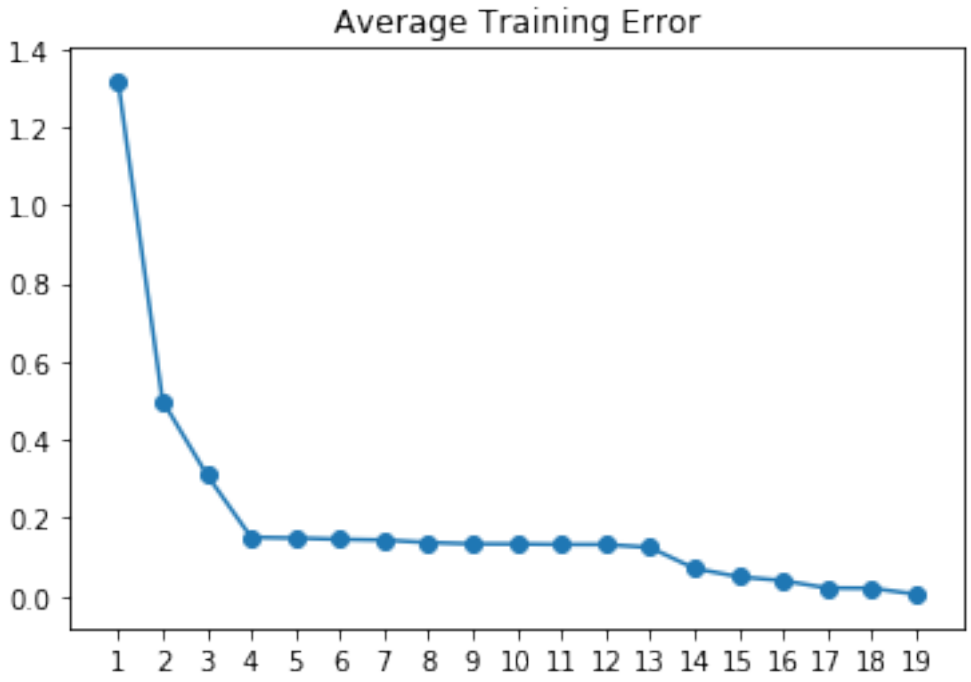


In [24]:
```
resultn = []
errorn = []
errorn_test = []
```

```
        for i in range(1,N):
            resultn += [regression(xn, yn, yn_test, i)]
            errorn += [resultn[i-1]['error']]
            errorn_test += [resultn[i-1]['error_test']]
```

In [25]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_xticks(np.arange(1,20,1))
plt.title('Average Training Error')
plt.plot(np.arange(1,20,1),errorn)
plt.scatter(np.arange(1,20,1),errorn)
```

Out[25]: <matplotlib.collections.PathCollection at 0x114814550>

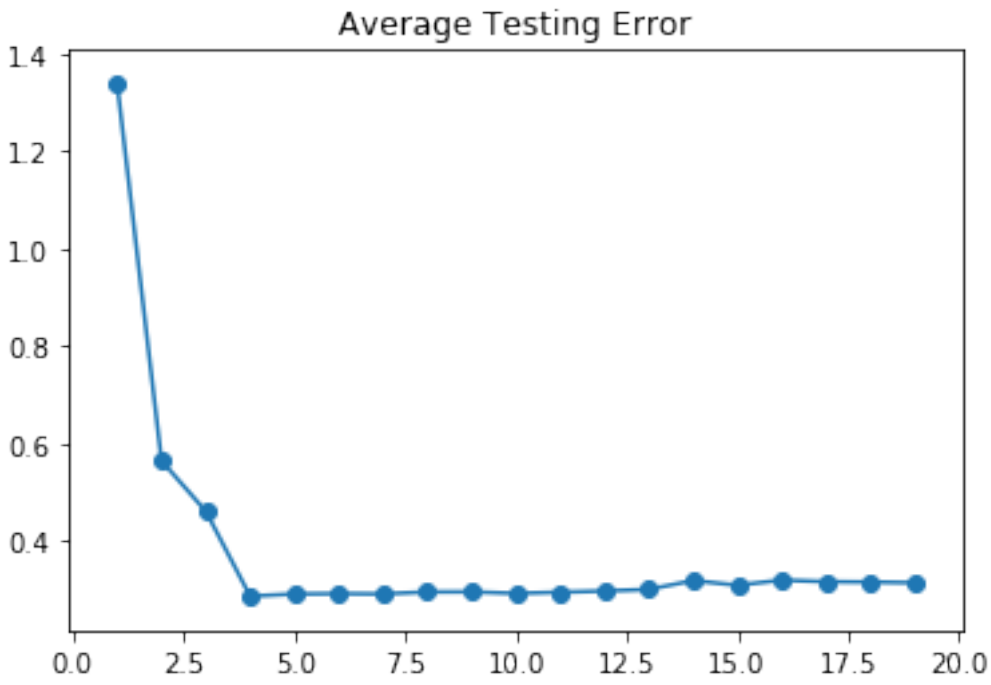


In [26]:
```
nmodel = regression(xn, yn, yn_test, 20)
print(nmodel['error'])
```

```
0.00544490084323
```

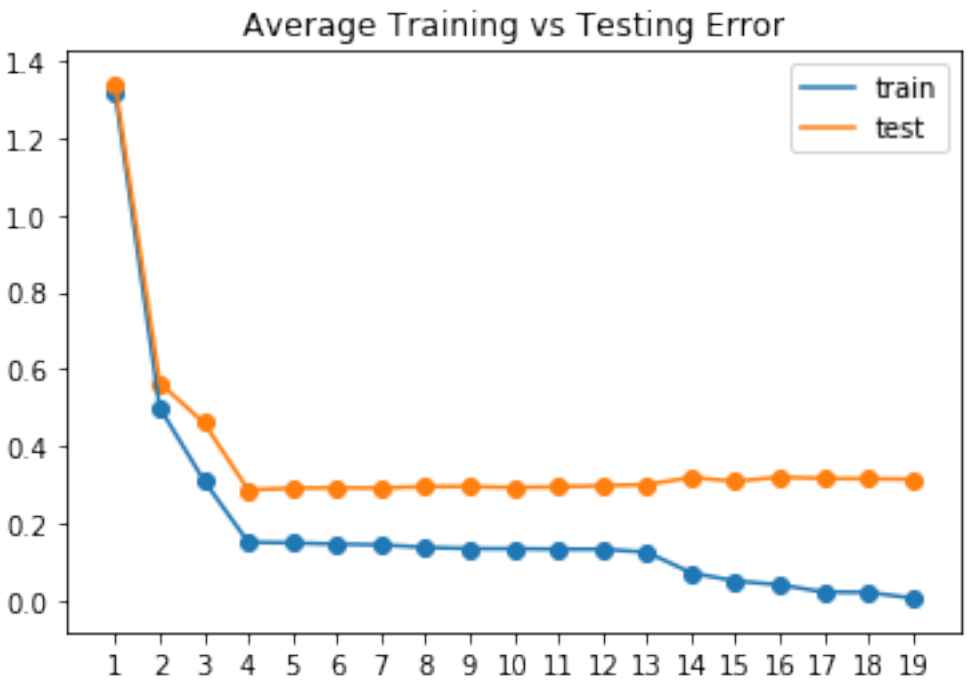
In [27]:
```
ax = fig.add_subplot(111)
ax.set_xticks(np.arange(1,20,1))
plt.title('Average Testing Error')
plt.plot(np.arange(1,20,1),errorn_test)
plt.scatter(np.arange(1,20,1),errorn_test)
```

Out[27]: <matplotlib.collections.PathCollection at 0x114788eb8>



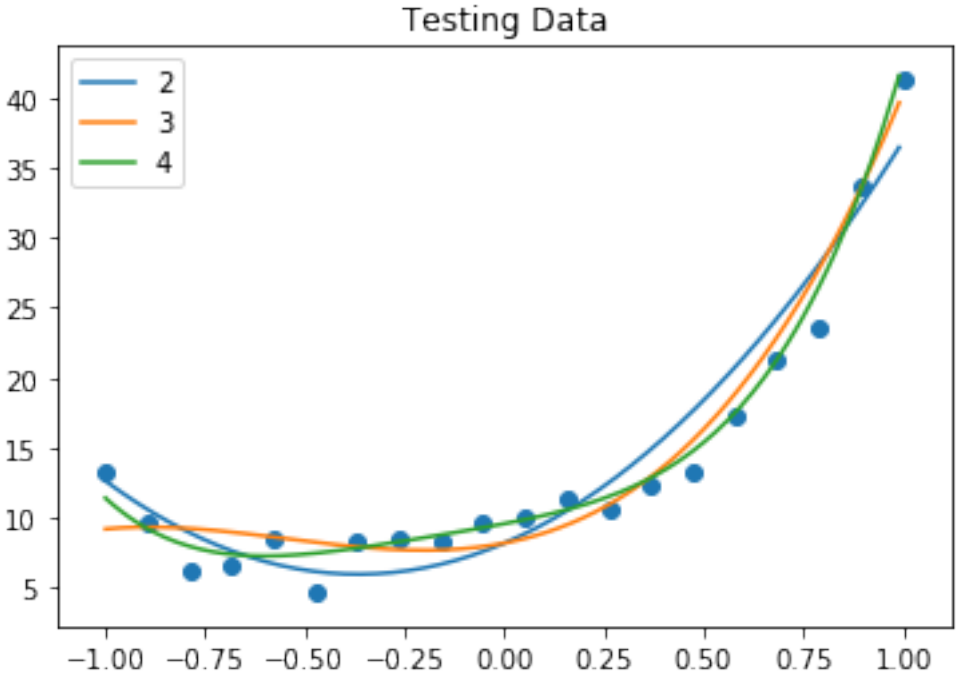
In [28]:
```
fig2 = plt.figure()
ax2 = fig2.add_subplot(111)
plt.plot(np.arange(1,20,1),errorn,label='train')
plt.scatter(np.arange(1,20,1),errorn)
plt.plot(np.arange(1,20,1),errorn_test,label='test')
plt.scatter(np.arange(1,20,1),errorn_test)
ax2.set_xticks(np.arange(1,20,1))
plt.legend()

plt.title('Average Training vs Testing Error')
```

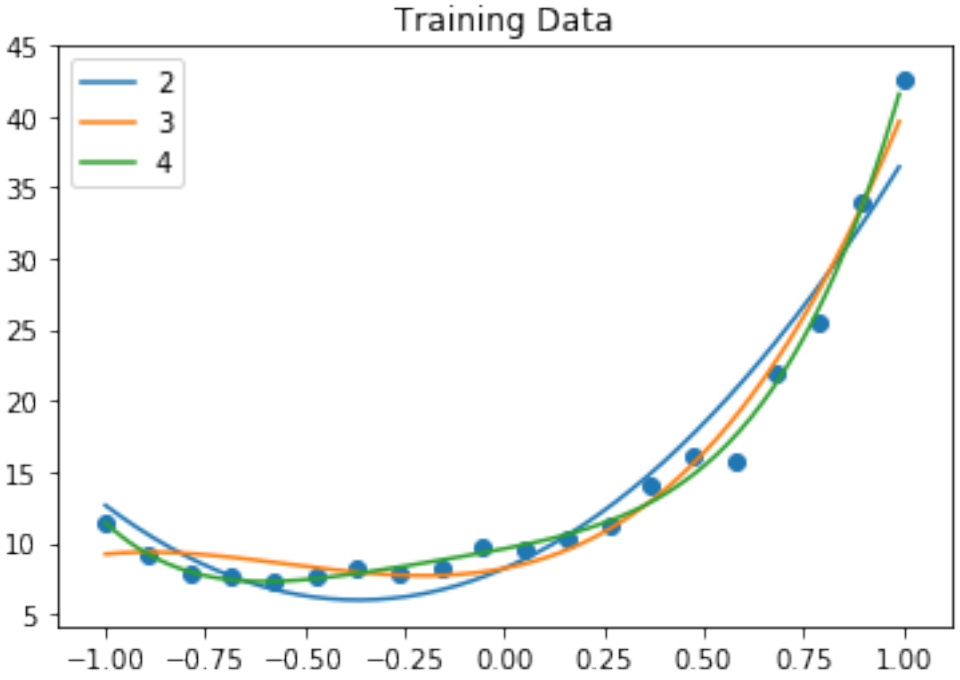
Out[28]: <matplotlib.text.Text at 0x11487d9e8>



In [29]:
```
fig3 = plt.figure()
ax3 = fig3.add_subplot(111)
xx = np.arange(-1,1,0.01)
xx = xx.reshape(200,1)
X3 = np.ones((len(xx),1))
for i in range(1,3):
    X3 = np.hstack((X3,xx**i))
plt.plot(xx,X3.dot(resultn[1]['A']),label='2')
X4 = np.hstack((X3,xx**3))
plt.plot(xx,X4.dot(resultn[2]['A']),label='3')
X5 = np.hstack((X4,xx**4))
plt.plot(xx,X5.dot(resultn[3]['A']),label='4')
plt.scatter(xn,yn_test)
plt.title('Testing Data')
plt.legend()

fig4 = plt.figure()
ax4 = fig4.add_subplot(111)
plt.plot(xx,X3.dot(resultn[1]['A']),label='2')
plt.plot(xx,X4.dot(resultn[2]['A']),label='3')
plt.plot(xx,X5.dot(resultn[3]['A']),label='4')
plt.scatter(xn,yn)
plt.legend()
plt.title('Training Data')
```

Out[29]: <matplotlib.text.Text at 0x1149efc50>





```
In [30]: print('coefficients of 2-degree polynomial:')
         print(resultn[1]['A'])
         print('coefficients of 3-degree polynomial:')
         print(resultn[2]['A'])
         print('coefficients of 4-degree polynomial:')
         print(resultn[3]['A'])

coefficients of 2-degree polynomial:
[[  8.20077407]
 [ 12.13018879]
 [ 16.56561306]]
coefficients of 3-degree polynomial:
[[  8.20077407]
 [  5.45186371]
 [ 16.56561306]
 [ 10.10425546]]
coefficients of 4-degree polynomial:
[[  9.58634937]
 [  5.45186371]
 [  3.87983398]
 [ 10.10425546]
 [ 13.50335457]]
```