HW09- CS 189

1.
Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

I worked on this homework with Ehimare Okoyomon, Prashanth Ganeth, and Daniel Mockaitis. We worked by getting together throughout the week and communicating on facebook.

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}

Nicholas Lorio, 26089160

Question 6 Own Question.

How can we define entropy in the scope of machine learning?

Entropy is defined as the sum of probability of each label times the log probability of that same label. It is our measure of impurity in the decision tree. There is entropy in the probabilities of this tree which arises from the entropy/randomness of the probabilities which determine our classification labeling assignments. The outcome of these decisions are determined by tests performed on the attributes of our features.

At each node of the tree, a test is performed for every features. The feature that has the largest information gain, the difference in the entropy before and the entropy after the decision, is chosen for the decision split. ML favors features that produce decisions with low uncertainty/entropy. The process stops when the leaf node contains instances which all have the same class.

This methodology for ML helps to label for classification. We desire to minimize the incorrect labeling which in turns means we wish to minimize entropy in our decisions.

HW09

189 (Z.Q. Math) $L(F_{opt}(x), y) = \begin{cases} 0, & i = y \\ \lambda_c, & i \neq y \\ \lambda_r, & i = c+1 \end{cases}$

①

$$R(F(x)|x) = \sum_{i=1}^{c} L(F(x), i) P(Y=i|x)$$

$$= L(F(x), i) + \sum_{j \neq i}^{c} L(F(x), j) P(Y=j|x)$$

$$R(i|x) = 0 \cdot P(i|x) + \lambda_c \cdot \sum_{j \neq i}^{c} P(j|x) = \lambda_c \sum_{j \neq i}^{c} P(j|x)$$

$P(i|x)$ is the largest $\therefore$ we know that choosing any other class assignment j would make our risk bigger as

$$\sum_{j \neq i} P(j|x) \leq \sum_{j \neq \text{any other } i\text{'s classes}} P(j|x)$$

② show that warrants the risk

$$R(i|x) \leq R(c+1|x)$$

$$\lambda_c \sum_{j \neq i} P(j|x) \leq \sum_{i}^{c} L(c+1|x) P(i|x)$$

$$\lambda_c \sum_{j \neq i} P(j|x) \leq \lambda_r \underbrace{\sum_{i=1}^{c} P(i|x)}_{1 \text{ encompasses all probabilities}}$$

$$\lambda_c (1 - P(i|x)) \leq \lambda_r$$

$$1 - P(i|x) \leq \frac{\lambda_r}{\lambda_c}, \quad P(i|x) \geq 1 - \frac{\lambda_r}{\lambda_c} \text{ as required.}$$

as such the given $F_{opt}(x)$ obtains minimum risk.

Nico Lerio
7008986

HW09

(2.a. intuition) $\min \sum_i L(F(x), i) P(y=i|x) = \min E(\text{risk})$

$F_{opt}(x)$

1. $P(y=i|x) \geq P(y=j|x)$ For all j, class w/ highest prob given x.

2. Choose class i if $P(y=i|x) \geq 1 - \frac{\lambda_d}{\lambda_c} = \frac{\lambda_c - \lambda_d}{\lambda_c}$

3. Choose Doubt else

$F_{opt}(x)$ -> Find $\max_i P(y=i|x)$ (1) → return i if $P(y=i|x) \geq 1 - \frac{\lambda_d}{\lambda_c}$ (2)

→ return $C+1$ else (3)

(1) We want to choose the class i which is most certain / has greatest probability of being associated w/ the given x.

(2) Assume $\lambda_d < \lambda_c$: if our penalty for misclarification (λ_c) is much greater than our penalty for doubt (λ_d) => $1 - \frac{\lambda_d}{\lambda_c}$ Becomes very close to 1. This means that we only choose i given x if we are very certain of our assignment. This is all respective to the values of λ_c, λ_d. If they are close to eachother but $\lambda_c > \lambda_d$ then we are more likely to choose i.

(3) We classify our x into $C+1$ if this level of certainty is not reached. It is not worth the penalty risk to guess.

∴ we optimize our risk $f_{opt}(x)$

2.b. if $\lambda_i = 0$ then under our current decision rules we will always assign class i to x.

i) => Need to amend decision rule ②

(2*) We should always select class $C+1$ since there is no longer any penalty associated with doubt.

ii) $\lambda_d > \lambda_c$ then $P(Y=i|x) \geq 1 - \frac{\lambda_d}{\lambda_c}$ = negative

in this case we will always guess / assign class i to x regardless of how high the probability is. This because all probability are between 0 & 1 and will always be greater than a negative value.

- this is not intuitive as it defeats the point of trying to minimize risk associated with an incorrect classification assignment. (for $\lambda_d > \lambda_c$ case)

- the $\lambda_d = 0$ case is not intuitive as we effectively do not complete any classification whatsoever & are back to where we started if we classifiers assign every x to the $C+1$ (Doubt class)

3. a. label 1 w/ $p = \pi_1$ $\quad x \sim N(\mu_1, \Sigma)$

$x \in \mathbb{R}^d$

label 2 w/ $p = \pi_2$

$\mu_1 \in \mathbb{R}^d$

$\mu_2 \in \mathbb{R}^d$

$\Sigma \in \mathbb{R}^{d \times d}$

(Same as Σ)

i) MLE(L), $\hat{L}_{MLE} = \arg\max_L P(\text{Data} \mid \text{true})$

$P(\text{Data} \mid \text{label}) =$

- using $Q_k(x)$ as defined in Note 18 we can determine the decision boundary for this case.

- MLE has NO priors.

$$Q_1(x) = Q_2(x)$$

$$\ln(\cdot) - \tfrac{1}{2}(x-\hat{\mu}_1)^T \sigma^{-2} I (x-\hat{\mu}_1) - \tfrac{1}{2}\ln(|\sigma^2 I|) = \ln(\cdot)$$

$$- \tfrac{1}{2}(x-\hat{\mu}_2)^T \sigma^{-2} I (x-\hat{\mu}_2) - \tfrac{1}{2}\ln(|\sigma^2 I|)$$

$$(x-\hat{\mu}_1)^T \sigma^{-2} I (x-\hat{\mu}_1) = (x-\hat{\mu}_2)^T \sigma^{-2} I (x-\hat{\mu}_2)$$

$$(x-\hat{\mu}_1)^T (x-\hat{\mu}_1) = (x-\hat{\mu}_2)^T (x-\hat{\mu}_2)$$

- Our Decision boundary is the set of points $x \in X$ for which $\|x-\hat{\mu}_1\|_2^2 = \|x-\hat{\mu}_2\|_2^2$

$$\hat{L}_{MLE} = \begin{cases} 1, & \|x-\hat{\mu}_1\|_2^2 > \|x-\hat{\mu}_2\|_2^2 \\ 2, & \|x-\hat{\mu}_1\|_2^2 < \|x-\hat{\mu}_2\|_2^2 \\ 1 \text{ or } 2, & \|x-\hat{\mu}_1\|_2^2 = \|x-\hat{\mu}_2\|_2^2 \end{cases}$$

ii) MAP w/ priors $P(1)$ & $P(2)$

$Q_1(x) = Q_2(x)$ $\quad$ [let $P(1) = \pi_1$ & $P(2) = \pi_2$, $\pi_2 = 1 - \pi_1$]

$$\ln(P(1)) - \tfrac{1}{2}(x-\hat{\mu}_1)^T \hat{\Sigma}^{-1}(x-\hat{\mu}_1) - \tfrac{1}{2}\ln(|\hat{\Sigma}|)$$

$$\ln(P(2)) - \tfrac{1}{2}(x-\hat{\mu}_2)^T \hat{\Sigma}^{-1}(x-\hat{\mu}_2) - \tfrac{1}{2}\ln(|\hat{\Sigma}|)$$

$$2\ln(P(1)) - (x-\hat{\mu}_1)^T \hat{\Sigma}^{-1}(x-\hat{\mu}_1) = 2\ln(P(2)) - (x-\hat{\mu}_2)^T \hat{\Sigma}^{-1}(x-\hat{\mu}_2)$$

$$(x^T\Sigma^{-1} - \hat{\mu}_1^T\hat{\Sigma}^{-1})(x-\hat{\mu}_1) = x^T\Sigma^{-1}x - x^T\hat{\Sigma}^{-1}\hat{\mu}_1 - \hat{\mu}_1^T\hat{\Sigma}^{-1}x + \hat{\mu}_1^T\Sigma^{-1}\hat{\mu}_1$$

$$2\ln P(1) - x^T\Sigma^{-1}x + 2x^T\hat{\Sigma}^{-1}\hat{\mu}_1 - \hat{\mu}_1^T\hat{\Sigma}^{-1}\hat{\mu}_1 = 2\ln P(2) - x^T\Sigma^{-1}x + 2x^T\hat{\Sigma}^{-1}\hat{\mu}_2 - \hat{\mu}_2^T\hat{\Sigma}^{-1}\hat{\mu}_2$$

$$2(\ln(P(1)) - \ln(P(2)) + x^T\Sigma^{-1}(\hat{\mu}_1 - \hat{\mu}_2) - (\hat{\mu}_1^T\hat{\Sigma}^{-1}\hat{\mu}_1 - \hat{\mu}_2^T\hat{\Sigma}^{-1}\hat{\mu}_2) = 0$$

$$x^T\underbrace{\hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_2)}_{w} + \left(\ln\left(\frac{P(1)}{P(2)}\right) - \left(\frac{\hat{\mu}_1^T\hat{\Sigma}^{-1}\hat{\mu}_1 - \hat{\mu}_2^T\hat{\Sigma}^{-1}\hat{\mu}_2}{2}\right)\right) = 0$$

w : linear function, $F(x) = w^Tx + k = x^Tw + k$

$$\hat{L}_{MAP} = \begin{cases} 1, & \text{“} > \text{”} \\ 2, & \text{“ from below } < \text{”} \\ 1 \text{ or } 2, & 2\ln(P(1)) - (x-\hat{\mu}_1)^T\Sigma^{-1}(x-\hat{\mu}_1) = 2\ln(P(2)) - (x-\hat{\mu}_2)^T\Sigma^{-1}(x-\hat{\mu}_2) \end{cases}$$

$\ln(1) = 0$ so when the priors are equal then MAP gives same Decision boundary rules as MLE

$F(x) = w^T x + k \quad x \sim N(\mu_L, \Sigma)$ w/ $L = 1, 2$

3.b. $F(x) = w^T(x - v)$, let $w^T = (\Sigma^{-1}(\hat{\mu}_1 - \hat{\mu}_2))^T$

- let $y \in \mathbb{R}^2$ be one hot vector $y_i = e_k$ return thus $y_i = k$
- $\pi_1 = \pi_2 = 1/2$: Not zero mean
- Compute $\Sigma_{xx}, \Sigma_{xy}, \Sigma_{yy}$ as Function of μ_1, μ_2, Σ
- tower property of expectation - Hint

$\begin{bmatrix} 0 \\ k^{th} \text{ spot } 1 \\ 0 \end{bmatrix}$ std Basis each label is a std basis vector

- $\Sigma_{xx} = \Sigma$
- $\Sigma_{xy} = E[(x - E(x))(y - E(y))] = E[xy^T] - E[x]E[y]^T$

$E[x] = E[E[x|L=i]] = \frac{1}{2}E[x|L=1] + \frac{1}{2}E[x|L=2] = \frac{\mu_1 + \mu_2}{2}$ ($d \times 1$ vector)

$E[y] = E[E[y|L=i]] = \frac{1}{2} \cdot E(y|L=1) + \frac{1}{2}E(y|L=2)$

$\frac{1}{2}\underset{2\times 1}{\begin{bmatrix} 1 \\ 0 \end{bmatrix}} + \frac{1}{2}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \underset{2\times 1}{\begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}}$

$E[xy^T] = E[E[xy^T|L=i]] = \frac{1}{2}E[xy^T|L=1] + \frac{1}{2}E[xy^T|L=2]$

$= \frac{1}{2}\left(\underset{d\times 1}{\mu_1} \cdot \underset{1\times 2}{[1 \ 0]} + \underset{d\times 1 \cdot 1\times 2}{\mu_2 [0 \ 1]} \right) = \frac{1}{2}[\vec{\mu}_1 \ \vec{0}] + \frac{1}{2}[\vec{0} \ \vec{\mu}_2] = \left[\frac{\mu_1}{2} \ \frac{\mu_2}{2}\right]$

$\Sigma_{xy} = \underset{d\times 2}{\left[\mu_1/2 \ \mu_2/2\right]} - \underset{d\times 1 \ \ 1\times 2}{\frac{\mu_1 + \mu_2}{2}[1/2 \ 1/2]}$

$= [\mu_1/2 \ \mu_2/2] - \left[\frac{\mu_1+\mu_2}{4} \ \frac{\mu_1+\mu_2}{4}\right] = \boxed{\Sigma_{xy} = \underset{d\times 2}{\left[\frac{\mu_1 - \mu_2}{4} \ \frac{\mu_2 - \mu_1}{4}\right]}}$

3.b. cont.

- $\Sigma_{YY} = E[YY^T] - E[Y]E[Y]^T$

w/ computations from earlier & the following

$$E[YY^T] = E[E(YY^T \mid L=1)] = \frac{1}{2} E[YY^T \mid L=1] + \frac{1}{2} E[YY^T \mid L=2]$$

$$= \frac{1}{2} \underset{2\times 1}{\begin{bmatrix} 1 \\ 0 \end{bmatrix}} \underset{1\times 2}{\begin{bmatrix} 1 & 0 \end{bmatrix}} + \frac{1}{2} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$= \frac{1}{2} \left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}$$

$$\Sigma_{YY} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} - \underset{\begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix}}{\begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \end{bmatrix}} = \begin{bmatrix} 1/4 & -1/4 \\ -1/4 & 1/4 \end{bmatrix}$$

3.c. Part 1

$$\rho(U^T X, V^T Y) = \frac{Cov(U^T X, V^T Y)}{(Var(U^T X)\, Var(V^T Y))^{1/2}}$$

$$Cov(U^T X, V^T Y) = E[(U^T X - E[U^T X])(V^T Y - E[V^T Y])^T] =$$

$$= E[U^T (X - E[X])(Y^T V - E[Y]^T V)^T]$$

$$= U^T E[(X - E(X))(Y^T - E[Y^T])^T] V$$

By similar logic

$$Var(U^T X) = U^T Var(X) U \qquad \rho = \frac{U^T \Sigma_{XY} V}{(U^T \Sigma_{XX} U\, V^T \Sigma_{YY} V)^{1/2}}$$

$$Var(V^T Y) = V^T Var(Y) V$$

- lets Simplify this: $\rho = \dfrac{U^T \frac{1}{4}[\mu_1 - \mu_2 \quad \mu_2 - \mu_1] V}{(U^T \Sigma U\, V^T \frac{1}{4}[\,\cdot\,] V)^{1/2}}$

$$\propto \frac{U^T [\mu_1 \; \mu_2] \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} V}{(U^T \Sigma U\, V^T [\,\cdot\,] V)^{1/2}}$$

- Square tops & Bottom of the expression. (for ρ^2)

$$\propto \frac{(U^T [\mu_1 \; \mu_2] \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} V)^2}{U^T \Sigma U\, V^T [\,\cdot\,] V} = \frac{(U^T [\mu_1 \; \mu_2] \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} V)^2}{U^T \Sigma U\, V^T}$$

- $\partial/\partial U = 2(U^T [\mu_1 \; \mu_2])^1 \cdot [\mu_1 \; \mu_2] \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \cdot (U^T \Sigma U)^{-1}$

$+ (U^T [\mu_1 \; \mu_2])^2 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} (-1) \cdot (U^T \Sigma U)^{-2} \cdot U^T (\Sigma + \Sigma^T)$

- Σ is symmetric so $= 2\Sigma$

3.C.I. cont.

• Set equal to zero, solving for U^*

$$2U^T[\mu_1,\mu_2][\mu_1,\mu_2]\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}\cdot(U^T\Sigma U)^{-1} =$$

$$(U^T[\mu_1,\mu_2])^2\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}(U^T\Sigma U)^{-2}2U^T\Sigma$$

$$U^T[\mu_1,\mu_2]^2\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}(U^T\Sigma U)^{-1} = U^T{}^2[\mu_1,\mu_2]^2\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}(U^T\Sigma U)^{-2}U^T\Sigma$$

$$\Sigma^{-1}U^{T-1}U^T[\mu_1,\mu_2]^2\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}(U^T\Sigma U)^{-1} = U^T{}^2[\mu_1,\mu_2]^2\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}(U^T\Sigma U)^{-2}$$

$$\Sigma^{-1}[\mu_1,\mu_2]^2\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix} = U^T{}^2[\mu_1,\mu_2]^2\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}(U^T\Sigma U)^{-1}$$

$$\Sigma^{-1}[\mu_1,\mu_2]\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}(U^T\Sigma U)\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}^{-1}[\mu_1,\mu_2]^{-2} = U^T{}^2U^*$$

$$U^* = \left(\left(\Sigma^{-1}[\mu_1,\mu_2]\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}U^T\Sigma U\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}^{-1}[\mu_1,\mu_2]^{-2}\right)^T\right)^{1/2}$$

thus $U^* \propto \Sigma^{-1}[\mu_1,\mu_2]$ as required.

3.C. part two

$$w^* \propto \Sigma^{-1}(\mu_1 - \mu_2)$$

$$\& \; w = \Sigma^{-1}(\mu_1 - \mu_2)$$

thus $w^* \propto w$

$$Y = F(x) = w^T(x - v), \; f(x) = w^*(x - v)$$

thus determing Decision boundary LDA & perform CCA are equivalent.

given Σ_{xx} Σ_{xy} Σ_{yy} $\begin{bmatrix} 1/4 & -1/4 \\ -1/4 & 1/4 \end{bmatrix}^T = \begin{bmatrix} 1/4 & -1/4 \\ -1/4 & 1/4 \end{bmatrix}$ 3. extra

3.c.

$$\max_{u,v} \rho(u^T X, v^T Y) = \frac{Cov(u^T X, v^T Y)}{(Var(u^T X)\, Var(v^T Y))^{1/2}} = \frac{u^T Cov(X,Y)\, v}{(u^T Var(x)\, u\, v^T Var(y)\, v)^{1/2}}$$

* From Note 11

$$= \frac{u^T \Sigma_{xy} v}{(u^T \Sigma_{xx} u\, v^T \Sigma_{yy} v)^{1/2}}$$

• let us perform a change of variables as follows:

$$a = \Sigma_{xx}^{+1/2} u \;,\; b = \Sigma_{yy}^{+1/2} v \Rightarrow u = \Sigma_{xx}^{-1/2} a \;,\; v = \Sigma_{yy}^{-1/2} b$$

So $\rho = \dfrac{a^T \Sigma_{xx}^{-1/2\,T} \Sigma_{xy} \Sigma_{yy}^{-1/2} b}{(a^T a\; b^T b)^{1/2}}$

• Σ_{xx} is DSD, symmetric
so $\Sigma_{xx}^T = \Sigma_{xx}$
• Σ_{yy} symmetric: $\Sigma_{yy}^T = \Sigma_{yy}$

• By Cauchy Schwarz: $|\langle u, v\rangle|^2 \le \langle u, u\rangle \langle v, v\rangle$

$\langle\,,\rangle$ def dot product

$$\left(a^T \Sigma_{xx}^{-1/2} \Sigma_{xy} \Sigma_{yy}^{-1/2}\right) b \le \left(a^T \Sigma_{xx}^{-1/2} \Sigma_{xy} \Sigma_{yy}^{-1/2} \left(a^T \Sigma_{xx}^{-1/2} \Sigma_{xy} \Sigma_{yy}^{-1/2}\right)^T\right)^{1/2} \left(b^T b\right)^{1/2}$$

$$\le \left(a^T \Sigma_{xx}^{-1/2} \Sigma_{xy} \Sigma_{yy}^{-1/2} \Sigma_{yy}^{-1/2} \Sigma_{xy}^T \Sigma_{xx}^{-1/2} a\right)^{1/2} \left(b^T b\right)^{1/2}$$

thus $\rho \le \dfrac{\left(a^T \Sigma_{xx}^{-1/2} \Sigma_{xy} \Sigma_{yy}^{-1} \Sigma_{xy}^T \Sigma_{xx}^{-1/2} a\right)^{1/2}}{(a^T a)^{1/2}}$

• this is the Rayleigh Quotient $R(x) = \dfrac{x^T A x}{x^T x}$ & is bounded by $\lambda_{min}(A)$ & $\lambda_{max}(A)$

$\Rightarrow$ a is an eigenvector of $\left(\Sigma^{-1/2} \Sigma_{xy} \Sigma_{yy}^{-1} \Sigma_{xy}^T \Sigma_{xx}^{-1/2}\right)$

• Rayleigh Quotient is maximized by the eigenvector w/ the largest eigenvalue. Thus when we $\max_{u,v} \rho$ then a is the eigenvector w/ the corresponding largest eigenvalue.

3.C. cont,

- Using the canonical basis

$$a = \Sigma_{xx}^{1/2} U \quad , \quad U = \Sigma_{xx}^{-1/2} a$$

- U^* is an eigenvector of $\Sigma_{xx}^{-1} \Sigma_{xy} \Sigma_{yy}^{-1} \Sigma_{xy}^T$

$$= \Sigma^{-1} \frac{1}{4} \begin{bmatrix} M_1 - M_2 & M_2 - M_1 \end{bmatrix} \begin{bmatrix}$$

$$\Sigma_{yy}^{-1} = \begin{bmatrix} 1/4 & -1/4 \\ -1/4 & +1/4 \end{bmatrix}^{-1} = \underline{\quad I \quad}$$

- Σ_{yy} is NOT invertible therefore this method will not work.

S.a. $X_1 \dots X_n$ iid R.V Finite set $\{0, \dots, M\}$ $p_j = P(X_i = j)$ For $j = \{0, 1, \dots, M\}$

w/ $f^\wedge$ as empirical type is $\binom{\wedge}{f_0^{(\wedge)}, f_1^{(\wedge)}, \dots, f_M^{(\wedge)}}$

'crude Stirling approx $k! \approx \left(\frac{k}{e}\right)^k$

$f^\wedge$ is a vector w/ M entries where ith entry is $f_i^{(\wedge)}$

Show: $\binom{\wedge}{f^\wedge} \overset{①}{=} \binom{\wedge}{f_0^{(\wedge)}, f_1^{(\wedge)}, \dots, f_M^{(\wedge)}} \approx \overset{②}{\exp}\left(\wedge H\left(f^{(\wedge)}/\wedge\right)\right)$

w/ $H(p) = \sum_{j=0}^{M} p_j \ln\left(\frac{1}{p_j}\right)$

① $\frac{\wedge!}{f_0^{(\wedge)}! \, f_1^{(\wedge)}! \cdots f_M^{(\wedge)}!} \approx \frac{\left(\frac{\wedge}{e}\right)^\wedge}{\left(\frac{f_0^{(\wedge)}}{e}\right)^{f_0^{(\wedge)}} \cdots \left(\frac{f_M^{(\wedge)}}{e}\right)^{f_M^{(\wedge)}}}$

$\approx \frac{\wedge^\wedge}{e^\wedge} \cdot \frac{e^{f_0^{(\wedge)}} \cdot \dots \cdot e^{f_M^{(\wedge)}}}{\left(f_0^\wedge\right)^{f_0^{(\wedge)}} \cdots \left(f_M^{(\wedge)}\right)^{f_M^{(\wedge)}}} = \frac{\wedge^\wedge \exp\left(\sum_{j=0}^{M} f_j^{(\wedge)}\right) e^{1/\wedge}}{\left(f_0^\wedge\right)^{f_0^{(\wedge)}} \cdots \left(f_M^{(\wedge)}\right)^{f_M^{(\wedge)}}} = \frac{\wedge^\wedge e^{-\wedge} e^{\sum f_j^{(\wedge)}}}{\prod_{j=0}^{M} f_j^{(\wedge) f_j^{(\wedge)}}}$

by Stirling approx

② $\exp\left(\wedge H\left(f^\wedge/\wedge\right)\right)$

$= \exp\left(\wedge \sum_{j=0}^{M} \frac{f_j^{(\wedge)}}{\wedge} \cdot \ln\left(\frac{\wedge}{f_j^\wedge}\right)\right) = \exp\left(\sum_{j=0}^{M} f_j^{(\wedge)} \ln\left(\frac{\wedge^\wedge}{f_j^{(\wedge)\wedge}}\right)\right)$

$= \frac{\exp\left(\sum f_j^{(\wedge)} \ln(\wedge^\wedge)\right)}{\exp\left(\sum f_j^{(\wedge)} \ln\left(f_j^{(\wedge)}\right)\right)} = \frac{\wedge^\wedge \exp\left(\sum f_j^{(\wedge)}/\wedge\right)}{\exp\left(\sum \ln\left(f_j^{(\wedge)\wedge \frac{1}{\wedge} \cdot f_j^{(\wedge)}}\right)\right)}$

(2) $= \dfrac{\Lambda^{\Lambda} e^{\sum f_j^{(\Lambda)}/\Lambda}}{\sum \exp\left(\frac{1}{\Lambda} \sum f_j^{(\Lambda)} f_j^{(\Lambda)}\right)} = \dfrac{\Lambda^{\Lambda} e^{\sum f_j^{(\Lambda)}/\Lambda}}{e^{\ln(f_0^{f_0})} \cdot e^{\ln(f_1^{f_1})} \cdots}$

$= \dfrac{\Lambda^{\Lambda} e^{\frac{1}{\Lambda}\sum f_j^{(\Lambda)}}}{\left(f_0^{f_0^{(\Lambda)}} \cdot \ldots \cdot f_M^{f_M^{(\Lambda)}}\right)} = \dfrac{\Lambda^{\Lambda} e^{\frac{1}{\Lambda} \cdot \sum f_j^{(\Lambda)}}}{\prod_{j=0}^{M} f_j^{(\Lambda) f_j^{(\Lambda)}}}$

$= \dfrac{\Lambda^{\Lambda} e^{-\Lambda} e^{\sum_{j=0}^{M} f_j^{(\Lambda)}}}{\prod_{j=0}^{M} f_j^{(\Lambda) f_j^{(\Lambda)}}}$

Showing approximate equivalence as round

5.b show $\lim_{n\to\infty} \log \mathbb{P}\left(F^{(n)} = f^{(n)}\right) = -KL(f,p) = -\sum_i f_i \log\left(\frac{f_i}{p_i}\right)$

$$\lim_{n\to\infty} \frac{1}{n} \log\left(\binom{n}{f_0^{(n)}, \ldots, f_M^{(n)}} \prod_{j=1}^{M} p_j^{f_j^{(n)}}\right)$$

use crude Stirling approximation

$$\approx \lim_{n\to\infty} \frac{1}{n} \log\left(\exp\left(n \sum_{j=0}^{M} \left(\frac{f_j^{(n)}}{n}\right) \ln\left(\frac{n}{f_j^{(n)}}\right)\right) \prod_{j=1}^{M} p_j^{f_j^{(n)}}\right)$$

$$= \lim_{n\to\infty} \frac{n}{n} \sum_{j=0}^{M} \frac{f_j^{(n)}}{n} \ln\left(\frac{n}{f_j^{(n)}}\right) + \sum_{j=1}^{M} \frac{f_j^{(n)}}{n} \log(p_j)$$

$$= \lim_{n\to\infty} \frac{f_0^{(n)}}{n} \ln\left(\frac{n}{f_0^{(n)}}\right) + \sum_{j=1}^{M}\left(\frac{f_j^{(n)}}{n} \ln\left(\frac{n}{f_j^{(n)}}\right) + \frac{f_j^{(n)}}{n} \ln(p_j)\right)$$

$$\lim_{n\to\infty} \frac{1}{n} f^{(n)} = f \text{ by definition}$$

$$= \lim_{n\to\infty} \frac{f_0^{(n)}}{n} \ln\left(\frac{n}{f_0^{(n)}}\right) + \sum_{j=1}^{M}\left(\frac{f_j^{(n)}}{n}\left(\ln\left(\frac{n}{f_j^{(n)}}\right) + \ln(p_j)\right)\right)$$

$$+ \sum_{j=1}^{M} \frac{f_j^{(n)}}{n} \ln\left(\frac{n \cdot p_j}{f_j^{(n)}}\right)$$

$$= \lim_{n\to\infty} -\frac{f_0^{(n)}}{n} \ln\left(\frac{f_0^{(n)}}{n}\right) - \sum_{j=1}^{M} \frac{f_j^{(n)}}{n} \ln\left(\frac{f_j^{(n)}}{n \cdot p_j}\right)$$

$$= -f_0 \ln(f_0) - \sum_{j=1}^{M} f_j \ln\left(\frac{f_j}{p_j}\right) \quad \blacksquare$$

$$= -\sum_{j=0}^{M} f_j \ln\left(\frac{f_j}{p_j}\right) \quad \text{as required.}$$

5.c. Show: $KL(p, q_\theta) = C - \sum_{x \in X} \sum_{y \in Y} p(x,y) \log(q_\theta(y|x))$

$q_\theta(y|x) = \frac{q_\theta(x,y)}{q(x)} =$

$KL(p, q_\theta) = \sum_i p_i \log\left(\frac{p_i}{q_{\theta i}}\right)$ • given that p has prob distrib $p(x,y)$ & $q_\theta(x,y)$ has distrib $q_\theta(x,y)$

$= \sum_{x \in X} \sum_{y \in Y} p(x,y) \log\left(\frac{p(x,y)}{q_{\theta i}(x,y)}\right)$

• $\log\left(\frac{p(x,y)}{q_\theta(x,y)}\right) = \log(p(x,y)) - \log(q_\theta(x,y))$

$= \log(p(x,y)) - \log(q_\theta(y|x)\, q(x))$

$= \log(p(x,y)) - \log(q_\theta(y|x)) - \log(q(x))$

$= \sum\sum p(x,y)[\log(p(x,y)) - \log(q_\theta(y|x)) - \log(q(x))]$

$= \underbrace{\sum\sum(p(x,y)\log(p(x,y))}_{C_2} - p(x,y)\log(q_\theta(y|x)) - \underbrace{p(x,y)\log(q(x))}_{C_1})$

$= C - \sum_{x \in X} \sum_{y \in Y} p(x,y) \log(q_\theta(y|x))$ as required

as C is some constant independent of θ.

logistic regression

S.D. $Y_i = +1$ or $Y_i = -1$ predicting labels from features x_i using

transition probability model: $q_\theta(y_i | x_i) = \frac{1}{1 + \exp(-y_i \theta^T x_i)}$

- assume all x_i are distinct

use c, show that w/ $p(x,y) = \begin{cases} 1/n & \text{if } x = x_i \text{ \& } y = y_i \quad i = 1, 2, \dots, n \\ 0 & \text{else} \end{cases}$

we get $\min_\theta KL(p, q_\theta) = \min_\theta -\frac{1}{n} \sum_i \log(q_\theta(y_i | x_i))$

From c,

$$\min_\theta c - \sum_x \sum_y p(x,y) \log q_\theta(y|x) \propto \min_\theta -\sum_{x_i} \sum_{y_i} p(x_i, y_i) \log \left(\frac{1}{1 + \exp(-y_i \theta^T x_i)} \right)$$

$$= \min_\theta - \sum_i \frac{1}{n} \log \left((1 + \exp(-y_i \theta^T x_i))^{-1} \right) + 0$$

$$= \min_\theta \theta - \frac{1}{n} \sum_i \log(p_\theta(y_i | x_i)) \quad \text{as required}$$

or $= 0$ if $p(x,y) = 0$ in the else scenario.

Question 4

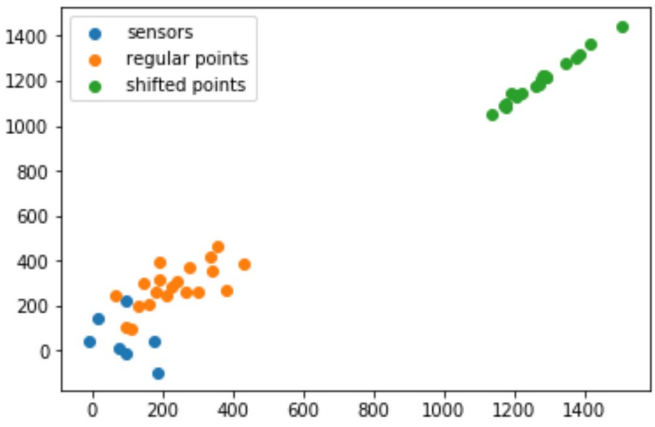
In [17]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import scipy.spatial


# Gradient descent optimization
# The learning rate is specified by eta
class GDOptimizer(object):
    def __init__(self, eta):
        self.eta = eta

    def initialize(self, layers):
        pass

    # This function performs one gradient descent step
    # layers is a list of dense layers in the network
    # g is a list of gradients going into each layer before the nonlinear activatio
n
    # a is a list of of the activations of each node in the previous layer going
    def update(self, layers, g, a):
        m = a[0].shape[1]
        for layer, curGrad, curA in zip(layers, g, a):
            layer.updateWeights(-self.eta / m * np.dot(curGrad, curA.T))
            layer.updateBias(-self.eta / m * np.sum(curGrad, 1).reshape(layer.b.sha
pe))


# Cost function used to compute prediction errors
class QuadraticCost(object):

    # Compute the squared error between the prediction yp and the observation y
    # This method should compute the cost per element such that the output is the
    # same shape as y and yp
    @staticmethod
    def fx(y, yp):
        return 0.5 * np.square(yp - y)

    # Derivative of the cost function with respect to yp
    @staticmethod
    def dx(y, yp):
        return y - yp


# Sigmoid function fully implemented as an example
class SigmoidActivation(object):
    @staticmethod
    def fx(z):
        return 1 / (1 + np.exp(-z))

    @staticmethod
    def dx(z):
        return SigmoidActivation.fx(z) * (1 - SigmoidActivation.fx(z))


# Hyperbolic tangent function
class TanhActivation(object):

    # Compute tanh for each element in the input z
    @staticmethod
    def fx(z):
        return np.tanh(z)
```

4.a

In [24]:
```
import numpy as np
import scipy.spatial


######################################################################
## Models used for predictions.
######################################################################
def compute_update(single_obj_loc, sensor_loc, single_distance):
    """
    Compute the gradient of the log-likelihood function for part a.

    Input:
    single_obj_loc: 1 * d numpy array.
    Location of the single object.

    sensor_loc: k * d numpy array.
    Location of sensor.

    single_distance: k dimensional numpy array.
    Observed distance of the object.

    Output:
    grad: d-dimensional numpy array.

    """
    loc_difference = single_obj_loc - sensor_loc  # k * d.
    phi = np.linalg.norm(loc_difference, axis=1)  # k.
    grad = loc_difference / np.expand_dims(phi, 1)  # k * 2.
    update = np.linalg.solve(grad.T.dot(grad), grad.T.dot(single_distance - phi))

    return update


def get_object_location(sensor_loc, single_distance, num_iters=20, num_repeats=10):
    """
    Compute the gradient of the log-likelihood function for part a.

    Input:

    sensor_loc: k * d numpy array. Location of sensor.

    single_distance: k dimensional numpy array.
    Observed distance of the object.

    Output:
    obj_loc: 1 * d numpy array. The mle for the location of the object.

    """
    obj_locs = np.zeros((num_repeats, 1, 2))
    distances = np.zeros(num_repeats)
    for i in range(num_repeats):
        obj_loc = np.random.randn(1, 2) * 100
        for t in range(num_iters):
            obj_loc += compute_update(obj_loc, sensor_loc, single_distance)

        distances[i] = np.sum((single_distance - np.linalg.norm(obj_loc - sensor_lo
c, axis=1))**2)
        obj_locs[i] = obj_loc

    obj_loc = obj_locs[np.argmin(distances)]

    return obj_loc[0]
```

4.b

In [19]:
```
def main():
    np.random.seed(0)
    sensor_loc = generate_sensors()
    regular_loc, _ = generate_dataset(
        sensor_loc,
        num_sensors=sensor_loc.shape[0],
        spatial_dim=2,
        num_data=20,
        original_dist=True,
        noise=1)
    shifted_loc, _ = generate_dataset(
        sensor_loc,
        num_sensors=sensor_loc.shape[0],
        spatial_dim=2,
        num_data=20,
        original_dist=False,
        noise=1)

    plt.scatter(sensor_loc[:, 0], sensor_loc[:, 1], label="sensors")
    plt.scatter(regular_loc[:, 0], regular_loc[:, 1], label="regular points")
    plt.scatter(shifted_loc[:, 0], shifted_loc[:, 1], label="shifted points")
    plt.legend()
    plt.savefig("dataset.png")
    plt.show()


if __name__ == "__main__":
    main()
```



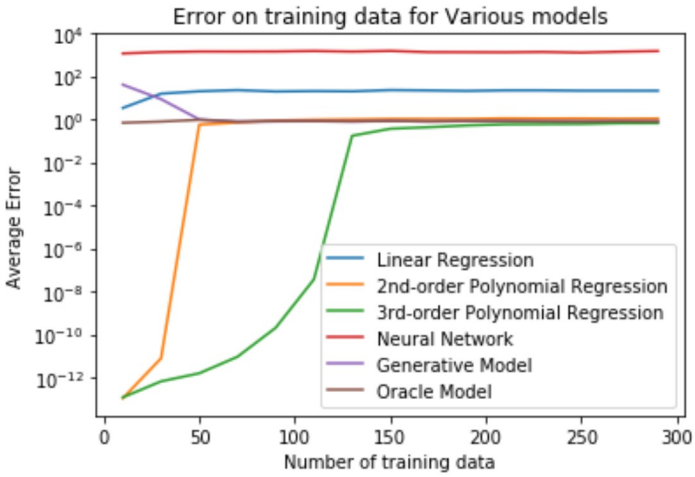
4.c

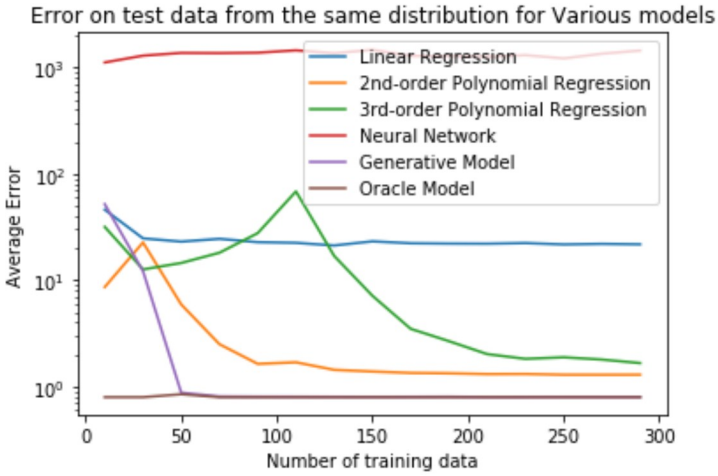
In [26]:
```
def main():
    ################################################################################
    ######################PLOT PART 1###############################################
    ################################################################################
    np.random.seed(0)

    ns = np.arange(10, 310, 20)
    replicates = 5
    num_methods = 6
    num_sets = 3
    mses = np.zeros((len(ns), replicates, num_methods, num_sets))

    def generate_data(sensor_loc, k=7, d=2, n=1, original_dist=True, noise=1):
        return generate_dataset(
            sensor_loc,
            num_sensors=k,
            spatial_dim=d,
            num_data=n,
            original_dist=original_dist,
            noise=noise)

    for s in range(replicates):
        sensor_loc = generate_sensors()
        X_test, Y_test = generate_data(sensor_loc, n=1000)
        X_test2, Y_test2 = generate_data(
            sensor_loc, n=1000, original_dist=False)
        for t, n in enumerate(ns):
            X, Y = generate_data(sensor_loc, n=n)  # X [n * 2] Y [n * 7]
            Xs_test, Ys_test = [X, X_test, X_test2], [Y, Y_test, Y_test2]
            ### Linear regression:
            mse = linear_regression(X, Y, Xs_test, Ys_test)
            mses[t, s, 0] = mse

            ### Second-order Polynomial regression:
            mse = poly_regression_second(X, Y, Xs_test, Ys_test)
            mses[t, s, 1] = mse

            ### 3rd-order Polynomial regression:
            mse = poly_regression_cubic(X, Y, Xs_test, Ys_test)
            mses[t, s, 2] = mse

            ### Neural Network:
            mse = neural_network(X, Y, Xs_test, Ys_test)
            mses[t, s, 3] = mse

            ### Generative model:
            mse = generative_model(X, Y, Xs_test, Ys_test)
            mses[t, s, 4] = mse

            ### Oracle model:
            mse = oracle_model(X, Y, Xs_test, Ys_test, sensor_loc)
            mses[t, s, 5] = mse

            print('{}th Experiment with {} samples done...'.format(s, n))

    ### Plot MSE for each model.
    plt.figure()
    regressors = [
        'Linear Regression', '2nd-order Polynomial Regression',
        '3rd-order Polynomial Regression', 'Neural Network',
        'Generative Model', 'Oracle Model'
    ]
    for a in range(6):
        plt.plot(ns, np.mean(mses[:, :, a, 0], axis=1), label=regressors[a])
```

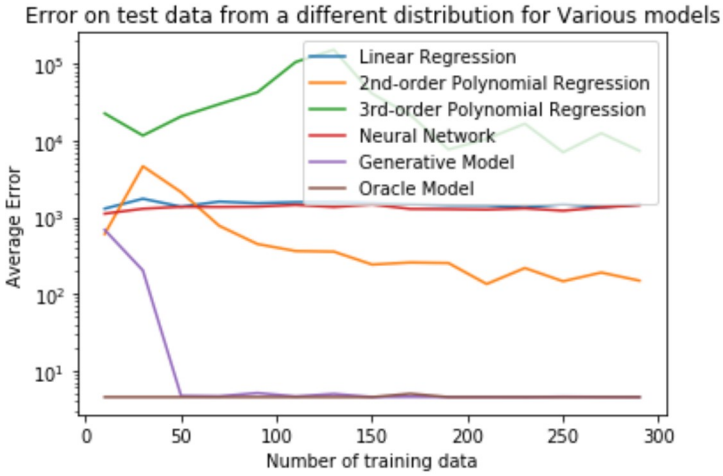
```
C start time: 2018-03-23 20:32:13.431749
```

```
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:142: Futur
eWarning: `rcond` parameter will change to the default of machine precision time
s ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None
`, to keep using the old, explicitly pass `rcond=-1`.
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:172: Futur
eWarning: `rcond` parameter will change to the default of machine precision time
s ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None
`, to keep using the old, explicitly pass `rcond=-1`.
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:201: Futur
eWarning: `rcond` parameter will change to the default of machine precision time
s ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None
`, to keep using the old, explicitly pass `rcond=-1`.
```

```
0th Experiment with 10 samples done...
0th Experiment with 30 samples done...
0th Experiment with 50 samples done...
0th Experiment with 70 samples done...
0th Experiment with 90 samples done...
0th Experiment with 110 samples done...
0th Experiment with 130 samples done...
0th Experiment with 150 samples done...
0th Experiment with 170 samples done...
0th Experiment with 190 samples done...
0th Experiment with 210 samples done...
0th Experiment with 230 samples done...
0th Experiment with 250 samples done...
0th Experiment with 270 samples done...
0th Experiment with 290 samples done...
1th Experiment with 10 samples done...
1th Experiment with 30 samples done...
1th Experiment with 50 samples done...
1th Experiment with 70 samples done...
1th Experiment with 90 samples done...
1th Experiment with 110 samples done...
1th Experiment with 130 samples done...
1th Experiment with 150 samples done...
1th Experiment with 170 samples done...
1th Experiment with 190 samples done...
1th Experiment with 210 samples done...
1th Experiment with 230 samples done...
1th Experiment with 250 samples done...
1th Experiment with 270 samples done...
1th Experiment with 290 samples done...
2th Experiment with 10 samples done...
2th Experiment with 30 samples done...
2th Experiment with 50 samples done...
2th Experiment with 70 samples done...
2th Experiment with 90 samples done...
2th Experiment with 110 samples done...
2th Experiment with 130 samples done...
2th Experiment with 150 samples done...
2th Experiment with 170 samples done...
2th Experiment with 190 samples done...
2th Experiment with 210 samples done...
2th Experiment with 230 samples done...
2th Experiment with 250 samples done...
2th Experiment with 270 samples done...
2th Experiment with 290 samples done...
3th Experiment with 10 samples done...
3th Experiment with 30 samples done...
3th Experiment with 50 samples done...
3th Experiment with 70 samples done...
3th Experiment with 90 samples done...
3th Experiment with 110 samples done...
3th Experiment with 130 samples done...
3th Experiment with 150 samples done...
3th Experiment with 170 samples done...
3th Experiment with 190 samples done...
3th Experiment with 210 samples done...
3th Experiment with 230 samples done...
3th Experiment with 250 samples done...
3th Experiment with 270 samples done...
3th Experiment with 290 samples done...
4th Experiment with 10 samples done...
4th Experiment with 30 samples done...
4th Experiment with 50 samples done...
4th Experiment with 70 samples done...
```







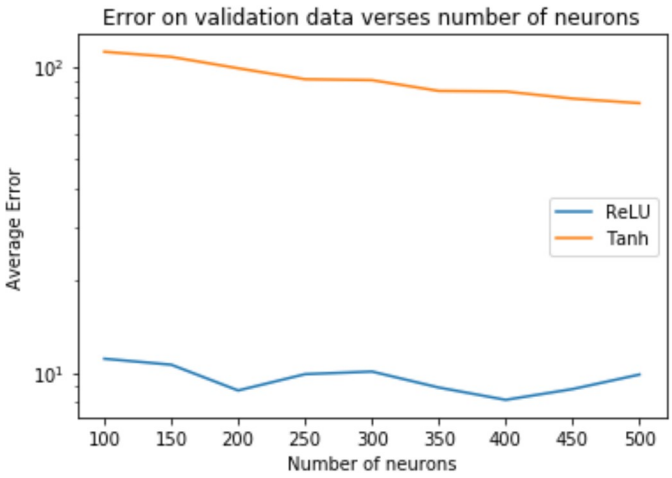

```
C end time: 2018-03-23 21:01:20.650690
```

Briefly describe your observations. What are the strengths and weaknesses of each model?

4.d

In [20]:
```
activations = dict(ReLU=ReLUActivation,
                       tanh=TanhActivation,
                       linear=LinearActivation)
lr = dict(ReLU=0.0000007,tanh=0.00001,linear=0.005)

def neural_network(X, Y, X_test, Y_test, num_neurons, activation):
    """
    This function performs neural network prediction.
    Input:
        X: independent variables in training data.
        Y: dependent variables in training data.
        X_test: independent variables in test data.
        Y_test: dependent variables in test data.
        num_neurons: number of neurons in each layer
        activation: type of activation, ReLU or tanh
    Output:
        mse: Mean square error on test data.
    """
    ## YOUR CODE HERE
    #################
    model = Model(X.shape[1])
    model.addLayer(DenseLayer(num_neurons, activations[activation]))
    model.addLayer(DenseLayer(num_neurons, activations[activation]))
    model.addLayer(DenseLayer(Y.shape[1],LinearActivation()))
    model.initialize(QuadraticCost())
    model.train(X,Y,500,GDOptimizer(eta=lr[activation]))
    Y_pred = model.predict(X_test)
    mse = np.mean(np.sqrt(np.sum((Y_pred - Y_test)**2, axis=1)))
    print(mse)
    return mse


##############################################################################
########################PLOT PART 2###########################################
##############################################################################
def generate_data(sensor_loc, k=7, d=2, n=1, original_dist=True, noise=1):
    return generate_dataset(
        sensor_loc,
        num_sensors=k,
        spatial_dim=d,
        num_data=n,
        original_dist=original_dist,
        noise=noise)


np.random.seed(0)
n = 200
num_neuronss = np.arange(100, 550, 50)
mses = np.zeros((len(num_neuronss), 2))

# for s in range(replicates):

sensor_loc = generate_sensors()
X, Y = generate_data(sensor_loc, n=n)  # X [n * 2] Y [n * 7]
X_test, Y_test = generate_data(sensor_loc, n=1000)
for t, num_neurons in enumerate(num_neuronss):
    ### Neural Network:
    mse = neural_network(X, Y, X_test, Y_test, num_neurons, "ReLU")
    mses[t, 0] = mse

    mse = neural_network(X, Y, X_test, Y_test, num_neurons, "tanh")
    mses[t, 1] = mse

    print('Experiment with {} neurons done...'.format(num_neurons))
```

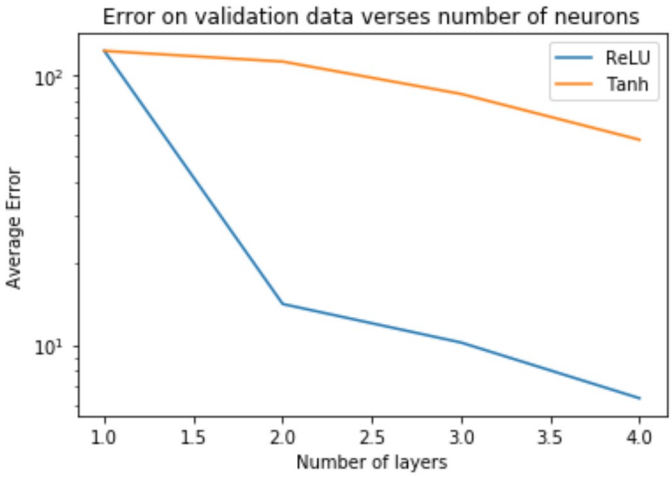
```
11.109407859943905
111.7986116835294
Experiment with 100 neurons done...
10.621046340776095
107.56930690987294
Experiment with 150 neurons done...
8.750118083604022
98.74964449460397
Experiment with 200 neurons done...
9.891959901212498
90.95010879210426
Experiment with 250 neurons done...
10.088487136454873
90.3092697911239
Experiment with 300 neurons done...
8.94582778492702
83.29509262423834
Experiment with 350 neurons done...
8.154480920424648
82.91806908671163
Experiment with 400 neurons done...
8.842727791373825
78.61146230571298
Experiment with 450 neurons done...
9.86359222938948
75.96352666746543
Experiment with 500 neurons done...
```



4.e

In [21]:
```
activations = dict(ReLU=ReLUActivation,
                       tanh=TanhActivation,
                       linear=LinearActivation)
lr = dict(ReLU=0.0000007,tanh=0.00001,linear=0.005)

def neural_network(X, Y, X_test, Y_test, num_layers, activation):
    """
    This function performs neural network prediction.
    Input:
        X: independent variables in training data.
        Y: dependent variables in training data.
        X_test: independent variables in test data.
        Y_test: dependent variables in test data.
        num_layers: number of layers in neural network
        activation: type of activation, ReLU or tanh
    Output:
        mse: Mean square error on test data.
    """
    ## YOUR CODE HERE
    #################
    # Have K hidden layers and L neurons per layer.
    # Want total # weights to be 10000
    # X.shape[1]*L + (k-1)*(L*L) + L*Y.shape[1] = 10000
    # => (k-1)*L^2 + L*(X.shape + Y.shape[1]) - 10000 = 0
    # Let x = X.shape[1], y = Y.shape[1].
    # Quadratic formula:
    # L = (-(x + y) +- sqrt((x+y)**2 - 4*(k-1)*(-10000))) / 2*(k-1)
    # Since expression on the right is positive and >= (x+1) for k >= 0,
    # only positive makes sense.
    k = num_layers
    x = X.shape[1]
    y = Y.shape[1]
    L = (-(x + y) + np.sqrt((x + y)**2 + 40000*(k-1)))/2*(k-1)
    L = int(L)

    model = Model(X.shape[1])
    for i in range(num_layers):
        model.addLayer(DenseLayer(L, activations[activation]))
    model.addLayer(DenseLayer(Y.shape[1],LinearActivation()))
    model.initialize(QuadraticCost())
    model.train(X,Y,500,GDOptimizer(eta=lr[activation])) # Learning rate of 0.02

    Y_pred = model.predict(X_test)
    mse = np.mean(np.sqrt(np.sum((Y_pred - Y_test)**2, axis=1)))
    print(mse)
    return mse


##########################################################################
#######################PLOT PART 2########################################
##########################################################################
def generate_data(sensor_loc, k=7, d=2, n=1, original_dist=True, noise=1):
    return generate_dataset(
        sensor_loc,
        num_sensors=k,
        spatial_dim=d,
        num_data=n,
        original_dist=original_dist,
        noise=noise)


np.random.seed(0)
n = 200
num_layerss = [1, 2, 3, 4]
```

```
123.34916925310644
123.37896279647254
Experiment with 1 layers done...
14.182293197206054
112.62096067500933
Experiment with 2 layers done...
10.202500813624377
85.39172088219182
Experiment with 3 layers done...
6.338764513589846
57.74814604922777
Experiment with 4 layers done...
```



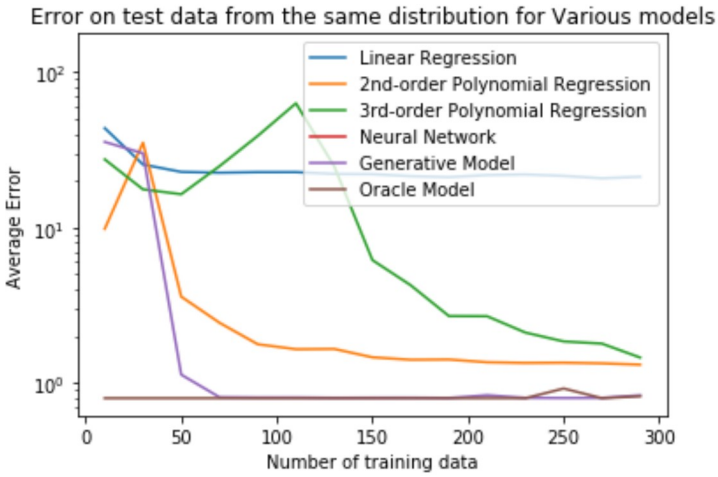
4.f

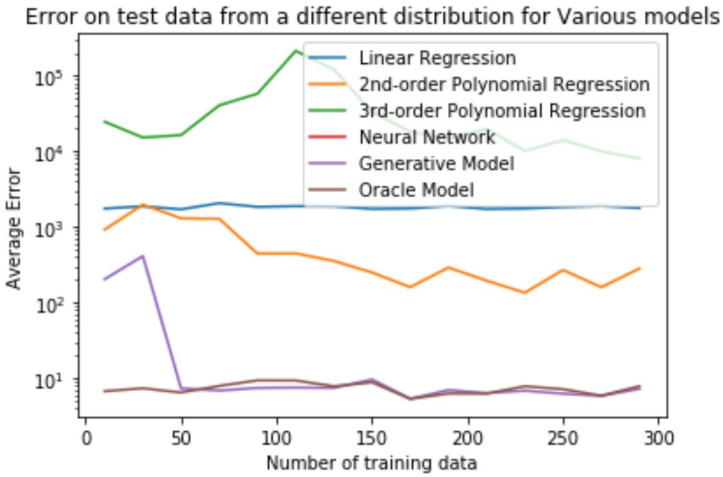
In [27]:
```
activations = dict(ReLU=ReLUActivation,
                       tanh=TanhActivation,
                       linear=LinearActivation)
lr = dict(ReLU=0.0000007,tanh=0.00001,linear=0.005)

def neural_network(X, Y, Xs_test, Ys_test):
    """
    This function performs neural network prediction.
    Input:
        X: independent variables in training data.
        Y: dependent variables in training data.
        X_test: independent variables in test data.
        Y_test: dependent variables in test data.
        num_layers: number of layers in neural network
        activation: type of activation, ReLU or tanh
    Output:
        mse: Mean square error on test data.
    """
    ## YOUR CODE HERE
    #################

    num_layers = 4
    activation = "ReLU"

    k = num_layers
    x = X.shape[1]
    y = Y.shape[1]
    L = (-(x + y) + np.sqrt((x + y)**2 + 40000*(k-1)))/2*(k-1)
    L = int(L)

    model = Model(X.shape[1])
    for i in range(num_layers):
        model.addLayer(DenseLayer(L, activations[activation]))
    model.addLayer(DenseLayer(Y.shape[1],LinearActivation()))
    model.initialize(QuadraticCost())
    model.train(X,Y,500,GDOptimizer(eta=lr[activation])) # Learning rate of 0.02

    Y_pred = model.predict(X_test)
    mse = np.mean(np.sqrt(np.sum((Y_pred - Y_test)**2, axis=1)))
    print(mse)
    return mse


def main():
    ###########################################################################
    ######################PLOT PART 1##########################################
    ###########################################################################
    np.random.seed(0)

    ns = np.arange(10, 310, 20)
    replicates = 5
    num_methods = 6
    num_sets = 3
    mses = np.zeros((len(ns), replicates, num_methods, num_sets))

    def generate_data(sensor_loc, k=7, d=2, n=1, original_dist=True, noise=1):
        return generate_dataset(
            sensor_loc,
            num_sensors=k,
            spatial_dim=d,
            num_data=n,
            original_dist=original_dist,
            noise=noise)
```

```
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:142: Futur
eWarning: `rcond` parameter will change to the default of machine precision time
s ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None
`, to keep using the old, explicitly pass `rcond=-1`.
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:172: Futur
eWarning: `rcond` parameter will change to the default of machine precision time
s ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None
`, to keep using the old, explicitly pass `rcond=-1`.
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:201: Futur
eWarning: `rcond` parameter will change to the default of machine precision time
s ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None
`, to keep using the old, explicitly pass `rcond=-1`.
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:36: Runtim
eWarning: overflow encountered in square
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:224: Runti
meWarning: invalid value encountered in multiply
/usr/local/lib/python3.5/dist-packages/numpy/core/_methods.py:32: RuntimeWarning
: overflow encountered in reduce
  return umr_sum(a, axis, dtype, out, keepdims)
/home/nlorio/.local/lib/python3.5/site-packages/ipykernel_launcher.py:77: Runtim
eWarning: invalid value encountered in greater
```

```
nan
0th Experiment with 10 samples done...
12.454824686649197
0th Experiment with 30 samples done...
nan
0th Experiment with 50 samples done...
nan
0th Experiment with 70 samples done...
20.41352178421555
0th Experiment with 90 samples done...
nan
0th Experiment with 110 samples done...
nan
0th Experiment with 130 samples done...
nan
0th Experiment with 150 samples done...
nan
0th Experiment with 170 samples done...
6.5905904523171195
0th Experiment with 190 samples done...
9.44426796016046
0th Experiment with 210 samples done...
8.066762207381036
0th Experiment with 230 samples done...
nan
0th Experiment with 250 samples done...
6.940402625030421
0th Experiment with 270 samples done...
7.2149912532420695
0th Experiment with 290 samples done...
187.06122990828078
1th Experiment with 10 samples done...
nan
1th Experiment with 30 samples done...
163.8230704844111
1th Experiment with 50 samples done...
nan
1th Experiment with 70 samples done...
145.32426742219718
1th Experiment with 90 samples done...
202.54499202115264
1th Experiment with 110 samples done...
nan
1th Experiment with 130 samples done...
175.0197396715583
1th Experiment with 150 samples done...
160.60171355672915
1th Experiment with 170 samples done...
nan
1th Experiment with 190 samples done...
175.11112759655146
1th Experiment with 210 samples done...
nan
1th Experiment with 230 samples done...
161.60692438275012
1th Experiment with 250 samples done...
nan
1th Experiment with 270 samples done...
195.2119073597044
1th Experiment with 290 samples done...
144.17268685961153
2th Experiment with 10 samples done...
nan
2th Experiment with 30 samples done...
```

```
/usr/local/lib/python3.5/dist-packages/matplotlib/scale.py:114: RuntimeWarning:
invalid value encountered in less_equal
  out[a <= 0] = -1000
```





HW09_Q4_Observations

4.C

Briefly describe your observations. What are the strengths and weaknesses of each Model?

Error on training data for various models:

Generative model has lowest average error, Oracle model quickly approaches this same error magnitude. Neural network performs the worst in terms of average error. The higher order polynomial regressions approach the same magnitude as generative and oracle.

With the exception of 2nd and 3rd order polynomial regression, average error stays relatively consistent as number of data training data increases.

Linear regression:

Error: Stable as number of training data increases
Same Dist: Stable as number of training data increases
Diff Dist: Stable as number of training data increases

Roughly same magnitude of error for all

2nd Order:
Error: Stable as number of training data increases
Same Dist: Peaks earlier than 3rd
Diff Dist: General trend towards decereasing error

3rd Order:
Error: Stable as number of training data increases
Same Dist: Error increases than decreases
Diff Dist: Multiple peaks in error, general trend towards decreases error

Neural Network:
Error: Stable as number of training data increases
Same Dist: Stable as number of training data increases
Diff Dist: Stable as number of training data increases

Generative Model:
Error: Lowest error magnitude

Same Dist: Lowest error magnitude
Diff Dist: Lowest error magnitude

Oracle Model:
Error: Lowest error magnitude
Same Dist: Lowest error magnitude
Diff Dist: Lowest error magnitude

4. D

Best choice for number of neurons is 200 or 400 for ReLu and is 500 for Tanh.

4.E

Best choice for number of layers is 2 or 4. For Tanh best is 4.