CS 189 Introduction to Machine Learning
Fall 2017 HW9

This homework is due **Monday, October 30 at 10pm.**

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.

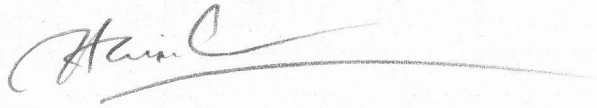
(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

None
Comments : None

(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 solns are entirely in my words & that I have not looked at another student's solns. I have credited all external sources in this write up.



Problem # 2 Classification policy

a.) The Bayes decision rule minimizes the risk

$$R(f(x)|x) = E[L(f(x), y)] = \sum_{i=1}^{c} L(f(x), y)\, P(Y=i|x)$$

Consider the given rule: 2 scenarios

1. given $P(Y=i|x) \geq P(Y=j|x)\ \forall j$ and $P(Y=i|x) \geq 1 - \frac{\lambda_r}{\lambda_s}$

if we choose class i, the risk is: ← policy

$$R_1(i|x) = \sum_{j=1}^{c} L(\underbrace{f(x)}_{i}, j)\, P(Y=j|x)$$

$$= \cancel{L(i,i)\, P(Y=i|x)}^{\to 0} + \sum_{\substack{j=1 \\ j\neq i}}^{c} \underbrace{L(j,i)}_{\lambda_s} P(Y=j|x)$$

$$= \lambda_s \sum_{\substack{j=1 \\ j\neq i}}^{c} P(Y=j|x)$$

if we choose class $j' \neq i$ the risk is: ← non-policy

$$R_2(j'|x) = \sum_{j=1}^{c} L(f(x), j)\, P(Y=j|x)$$

$$= \lambda_s P(Y=i|x) + \lambda_s \sum_{\substack{j=1 \\ j\neq i,j'}}^{c} P(Y=j|x) + \cancel{L(j',j')\, P(Y=j'|x)}^{\to 0}$$

$$= \lambda_s P(Y=i|x) + \lambda_s \sum_{\substack{j=1 \\ j\neq i,j'}}^{c} P(Y=j|x)$$

if we choose doubt, the risk is: ← non-policy

$$R_3(c+1|x) = \sum_{j=1}^{c} L(\overset{c+1}{f(x)}, j)\, P(Y=j|x)$$

$$= \lambda_r \underbrace{\sum_{j=1}^{c} P(Y=j|x)}_{1} = \lambda_r$$

now we compare R_1, R_2 & R_3:

$$R_1 = \lambda_s \sum_{\substack{j=1 \\ j\neq i}}^{c} P(Y=j|x) = \lambda_s P(Y=j'|x) + \sum_{\substack{j=1 \\ j\neq i,j'}}^{c} P(Y=j|x)$$

$$\left(\begin{array}{l}\text{since } P(Y=j|x) \leq P(Y=i|x) \\ \text{given } \forall\, j\end{array}\right) \leq \underbrace{\lambda_s P(Y=i|x) + \sum_{\substack{j=1 \\ j\neq i,j'}}^{c} P(Y=j|x)}_{R_2}$$

$$R_1 = \lambda_s \sum_{\substack{j=1 \\ j\neq i}}^{c} P(Y=j|x) + \lambda_s P(Y=i|x) - \lambda_s P(Y=i|x)$$

$$= \lambda_s \underbrace{\sum_{j=1}^{c} P(Y=j|x)}_{1} - \lambda_s \underbrace{P(Y=i|x)}_{\geq 1-\frac{\lambda_r}{\lambda_s}}$$

$$\leq \lambda_s - \lambda_s\left(1-\frac{\lambda_r}{\lambda_s}\right) = \lambda_s - \lambda_s + \lambda_r = \lambda_r$$

$$= R_3$$

i.e $R_1 \leq R_3$

Thus, R_1 is minimum

2. Given $P(Y=i|x) \geq P(Y=j|x)\ \forall\, j$ and $P(Y=i|x) < 1-\frac{\lambda_r}{\lambda_s}$

$$\Rightarrow P(Y=j|x) < 1-\frac{\lambda_r}{\lambda_s} \quad \forall\, j$$

(we can always find the largest $P(Y=i|x)$ thus

$P(Y=i|x) \geq P(Y=j|x)\ \forall\, j$ always holds)

If we choose doubt ← policy

$$R_1 = \lambda_r$$

If we choose i or j' ← non-policy

$$R_2 = \lambda_s \sum_{\substack{j=1 \\ j \neq i \text{ or } j'}}^{c} P(Y=j|x)$$

Since $P(Y=j'|x) < 1 - \frac{\lambda_r}{\lambda_s} \quad \forall j'$

$$\lambda_s P(Y=j'|x) < \lambda_s - \lambda_r$$

$$\lambda_r < \lambda_s \underbrace{\sum_{j=1}^{c} P(Y=j|x)}_{1} - \lambda_s P(Y=j'|x)$$

$$\lambda_s \sum_{\substack{j=1 \\ j \neq i \text{ or } j'}}^{c} P(Y=j|x)$$

$$R_1 < R_2$$

Thus R_1 is minimum

In all scenario, the given policy has the minimum risk, thus the policy is the Bayes decision rule

b.) If $\lambda_r = 0$, we should always choose doubt, since the risk for choosing doubt is $R = 0$ minimum

Intuitively, it is correct because we are not charged for doubt

If $\lambda_r > \lambda_s$, we should never choose doubt because the risk $R = \lambda_r > \lambda_s > \lambda_s \underbrace{\sum_{j=1}^{c} P(Y=j|x)}_{1} - \lambda_s P(Y=j'|x)$

R' : cost for choosing arbitrary j'

Thus the risk for choosing doubt is always larger than the risk for choosing arbitrary j'

Intuitively, it is correct because the cost for doubt is too large.

Problem #3 LDA & CCA

a.) The Bayes decison rule minimize the risk

$$R(f(X)|X) = E[L(f(X), X)]$$
$$= \sum_{\ell=1}^{n} P(L=\ell|X)\, L(f(X), \ell)$$

Assume the loss function:

$$L(f(X), \ell) = \begin{cases} 0 & \text{if } f(X) = \ell \\ 1 & \text{if } f(X) \neq \ell \end{cases}$$

Thus to minimize the risk, we have to choose the label ℓ corresponding to maximum $P(L=\ell|X)$

i.e $$f(X) = \underset{\ell}{\operatorname{argmax}}\, P(L=\ell|X) \quad \leftarrow \text{MAP}$$
$$= \underset{\ell}{\operatorname{argmax}} \frac{P(X|L=\ell)\,P(L=\ell)}{\int_\ell P(X|L=\ell)\,P(L=\ell)\,d\ell} \quad \leftarrow \text{const}$$
$$= \underset{\ell}{\operatorname{argmax}} \log P(X|L=\ell) + \underbrace{\log P(L=\ell)}_{\log \pi_\ell}$$

$$P(X|L=\ell) = \frac{1}{(\sqrt{2\pi})^d |\Sigma|^{1/2}} \exp\left(\frac{-1}{2}(X-\mu_\ell)^T \Sigma^{-1}(X-\mu_\ell)\right)$$

$$\Rightarrow f(X) = \underset{\ell}{\operatorname{argmin}} \left(\log(\sqrt{2\pi})^d |\Sigma|^{1/2} + \frac{1}{2}(X-\mu_\ell)\Sigma^{-1}(X-\mu_\ell) - \log \pi_\ell \right)$$

Since we have only 2 label $\{1, 2\}$ so we will choose label 1 if

$$\log(\sqrt{2\pi})^d |\Sigma|^{1/2} + \frac{1}{2}(X-\mu_1)^T\Sigma^{-1}(X-\mu_1) - \log \pi_1$$
$$\leq \log(\sqrt{2\pi})^d |\Sigma|^{1/2} + \frac{1}{2}(X-\mu_2)^T\Sigma^{-1}(X-\mu_2) - \log \pi_2$$

$$X^T\Sigma^{-1}X - 2\mu_1^T\Sigma^{-1}X + \mu_1^T\Sigma^{-1}\mu_1 - 2\log\pi_1 \leq X^T\Sigma^{-1}X - 2\mu_2^T\Sigma^{-1}X + \mu_2^T\Sigma^{-1}\mu_2 - 2\log\pi_2$$

$$2(\mu_1^T - \mu_2^T)\Sigma^{-1}X + 2\log\frac{\pi_1}{\pi_2} - \mu_1\Sigma^{-1}\mu_1 + \mu_2^T\Sigma^{-1}\mu_2 \geq 0$$

MAP → $f(x)$ linear in X

In summary, if $f(x) \geq 0$ choose label 1, else label 2

For MLE, we try to maximize $P(X \mid L=\ell)$. Similar to above but w/o $P(L=\ell)$ term. Thus

$$\arg\max_\ell P(X \mid L=\ell) = \arg\min_\ell \left(\log(\sqrt{2\pi})^d |\Sigma|^{1/2} + \frac{1}{2}(X-\mu_\ell)^T\Sigma^{-1}(X-\mu_\ell) \right)$$

The final expression is

$$2(\mu_1^T - \mu_2^T)\Sigma^{-1}X - \mu_1^T\Sigma^{-1}\mu_1 + \mu_2^T\Sigma^{-1}\mu_2 \geq 0$$

MLE → $f(x)$

if $f(x) \geq 0$ choose label 1, else label 2.

Two decision rules are the same if $\log\frac{\pi_1}{\pi_2} = 0$

i.e $\frac{\pi_1}{\pi_2} = 1$ or $\pi_1 = \pi_2$

(6) $\Sigma_{xx} = E[(x-\mu_x)(x-\mu_x)^T]$ denote $l_1=1$, $l_2=2$

$\mu_x = E[X] = \int_x X P(X)\,dX = \int_x X\,(P(X|l_1)P(l_1) + P(X|l_2)P(l_2))\,dx$ ($P(l_1)=\pi_1$, $P(l_2)=\pi_2$)

$= \pi_1 \int_x X P(X|l_1)\,dX + \pi_2 \int X P(X|l_2)\,dX$

$= \pi_1 E[X|l_1] + \pi_2 E[X|l_2]$

$= \pi_1 \mu_1 + \pi_2 \mu_2$

if $\pi_1 = \pi_2 = \frac{1}{2} \Rightarrow \mu_x = \frac{\mu_1+\mu_2}{2}$

Similar to $E[XX^T] = \int_x XX^T P(X)\,dX$

$= \pi_1 E[XX^T|l_1] + \pi_2 E[XX^T|l_2]$

$\Sigma_{xx} = E[XX^T - X\mu_x^T - \mu_x X^T + \mu_x\mu_x^T]$

$= E[XX^T] - E[X\mu_x^T] - E[\mu_x X^T] + E[\mu_x\mu_x^T]$

$E[X]\mu_x^T = \mu_x E[X^T] = \mu_x\mu_x^T$

$= E[XX^T] - \mu_x\mu_x^T$

$= \pi_1 E[XX^T|l_1] + \pi_2 E[XX^T|l_2] - \mu_x\mu_x^T$

$\Sigma = V[X|l_1] = E[XX^T|l_1] - \mu_1\mu_1^T$

$= V[X|l_2] = E[XX^T|l_2] - \mu_2\mu_2^T$

$\Rightarrow E[XX^T|l_1] = \Sigma + \mu_1\mu_1^T$

$E[XX^T|l_2] = \Sigma + \mu_2\mu_2^T$

$$\Sigma_{XX} = (\pi_1 + \pi_2)\Sigma + \pi_1 \mu_1 \mu_1^T + \pi_2 \mu_2 \mu_2^T - (\pi_1 \mu_1 + \pi_2 \mu_2)(\pi_1 \mu_1 + \pi_2 \mu_2)^T$$

if $\pi_1 = \pi_2 = \frac{1}{2}$

$$\Sigma_{XX} = \Sigma + \frac{1}{2}\mu_1 \mu_1^T + \frac{1}{2}\mu_2 \mu_2^T - \frac{\mu_1 \mu_1^T + \mu_1 \mu_2^T + \mu_2 \mu_1^T + \mu_2 \mu_2^T}{4}$$

$$= \Sigma + \frac{(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T}{4}$$

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

$Y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ given $L = l_1 \Rightarrow E[Y|l_1] = \mu_{y_1} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$$V[Y|l_1] = E[YY^T|l_1] - \mu_{y_1}\mu_{y_1}^T = 0$$

$$\Rightarrow E[YY^T|l_1] = \mu_{y_1}\mu_{y_1}^T = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$Y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ given $L = l_2 \Rightarrow E[Y|l_2] = \mu_{y_2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$$E[YY^T|l_2] = \mu_{y_2}\mu_{y_2}^T = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$E[YY^T] = \pi_{y_1} E[YY^T|l_1] + \pi_{y_2} E[YY^T|l_2]$$

$$= \begin{bmatrix} \pi_{y_1} & 0 \\ 0 & \pi_{y_2} \end{bmatrix}$$

assume $\pi_{y_1} = \pi_{y_2} = 1/2 \Rightarrow E[YY^T] = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}$

$$\mu_y = E[Y] = \pi_{y_1} E[Y|l_1] + \pi_{y_2} E[Y|l_2]$$

$$= \begin{bmatrix} \pi_{y_1} \\ \pi_{y_2} \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$$

$$\Sigma_{YY} = E[YY^T] - \mu_y \mu_y^T = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} - \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}\begin{bmatrix} 1/2 & 1/2 \end{bmatrix}$$

$$\Sigma_{yy} = \begin{bmatrix} 1/4 & -1/4 \\ -1/4 & 1/4 \end{bmatrix}$$

$$\Sigma_{xy} = E\left[(X-\mu_x)(Y-\mu_y)^T\right] = E[XY^T] - \mu_x \mu_y^T$$

(d) From part (b)

$$\Sigma_{xx} = (\pi_1 + \pi_2)\Sigma + \pi_1 \mu_1 \mu_1^T + \pi_2 \mu_2 \mu_2^T - (\pi_1 \mu_1 + \pi_2 \mu_2)(\pi_1 \mu_1 + \pi_2 \mu_2)^T$$

$$\Sigma_{yx} = \begin{bmatrix} 1/4 & -1/4 \\ -1/4 & 1/4 \end{bmatrix}$$

(e) Procedure:

1.) Calculate $\mu_x = \frac{1}{n} \sum_{i=1}^{n} x_i$

2.) Calculate $\Sigma_{xx} = E[(x - \mu_x)(x - \mu_x)^T]$

we know $\Sigma_{YY} = \begin{bmatrix} 1/4 & -1/4 \\ -1/4 & 1/4 \end{bmatrix}$

$\mu_Y = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$

Calculate

$$\Sigma_{XY} = E[(x - \mu_x)(Y - \mu_y)^T]$$

3.) From Σ_{XX} Σ_{YY} Σ_{XY}

Apply question (c) to determine w^*

4.) → predict x_{test} base on w^*

Problem #4

a.) See code attach

b.) See code attached

Description:

the second-order & third order has the best performance though the initial error is quite large.

The linear order has similar performance with neural network.

The generative model has largest error

Strengths and weaknesses

1. Generative: large error ×, no need location of sensors ✓

2.) linear / second / third order: pretty simple to implement and quite stable to run but need more # of sample (n not too small)

3.) Neural network: unstable, complicated model and need a lot of case

c.) See code attached

the best $\ell = 150$ //

d.) $(I+1)\ell + \underbrace{(\ell+1)\ell + \dots +}_{k-1 \text{ time}} (\ell+1) * O$

↑ input # ↑ output #

$= (I+1)\ell + (k-1)\ell(\ell+1) + (\ell+1) * O$

$I = 7 \quad O = 2$

$\Rightarrow 8\ell + (k-1)(\ell^2+1) + 2(\ell+1) = 10000$

↑ quadratic eqn in terms of ℓ

k best. is $k = 2$ //

See code & plot attached

e.) Choose $k = 2$

$\ell = 150$

error is ~15 ← small

See code attached

Problem # 5

See code attached!

% ≤ 92.18 %

Problem #6

Solve prob. 36 for QDA using MAP. $\Sigma_1 \neq \Sigma_2$.

and prove that the decision function $f(x)$ is quadratic

Solution

The decision function is

$$f(x) = \underset{\ell}{\operatorname{argmax}}\, P(L=\ell \mid x) \qquad \swarrow \text{MAP}$$

$$= \underset{\ell}{\operatorname{argmax}}\, \frac{P(x \mid L=\ell)\, P(L=\ell)}{\int_\ell P(x \mid L=\ell)\, P(L=\ell)\, d\ell}$$

$$= \underset{\ell}{\operatorname{argmax}}\, \log P(x \mid L=\ell) + \log P(L=\ell)$$

$$\Rightarrow f(x) = \underset{\ell}{\operatorname{argmin}} \Big(\log\left(\sqrt{2\pi}\right)^d |\Sigma_\ell|^{1/2} + \frac{1}{2}(x-\mu_\ell)\Sigma_\ell^{-1}(x-\mu_\ell) - \log \pi_\ell \Big)$$

$$f^*(x)$$

We will choose label 1 if

$$f^*(x \mid k=\ell_1) < f^*(x \mid L=\ell_2)$$

$$\log(\sqrt{2\pi})^d |\Sigma_1|^{1/2} + \frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1) - \log \pi_1$$

$$\leq \log(\sqrt{2\pi})^d |\Sigma_2|^{1/2} + \frac{1}{2}(x-\mu_2)^T \Sigma_2^{-1}(x-\mu_2) - \log \pi_2$$

$$x^T \Sigma_1^{-1} x - 2\mu_1^T \Sigma_1^{-1} x + \mu_1^T \Sigma_1^{-1} \mu_1 - 2\log \pi_1 + \log(2\pi)^d |\Sigma_1|^{1/2}$$

$$\leq x^T \Sigma_2^{-1} x - 2\mu_2^T \Sigma_2^{-1} x + \mu_2^T \Sigma_2^{-1} \mu_2 - 2\log \pi_2 + \log(\sqrt{2\pi})^d |\Sigma_2|^{1/2}$$

$$x^T(\Sigma_1^{-1} - \Sigma_2^{-1})x - 2(\mu_1^T \Sigma_1^{-1} - \mu_2^T \Sigma_2^{-1})x - 2\log \frac{\pi_1}{\pi_2}$$

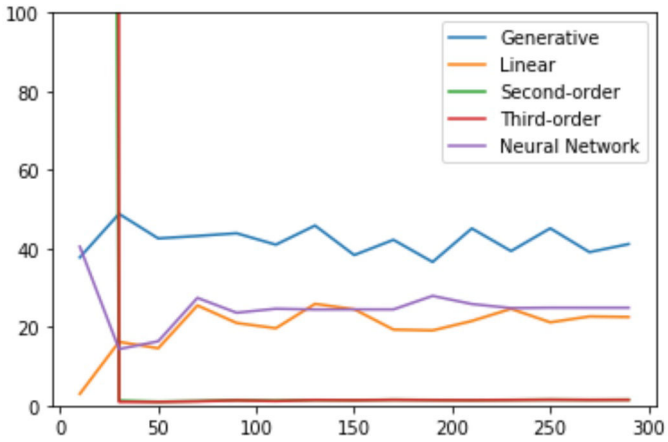
$$+ \mu_1^T \Sigma_1^{-1} \mu_1 + \mu_2^T \Sigma_2^{-1} \mu_2 + \log \frac{|\Sigma_1|^{1/2}}{|\Sigma_2|^{1/2}} \leq 0$$

decision rule

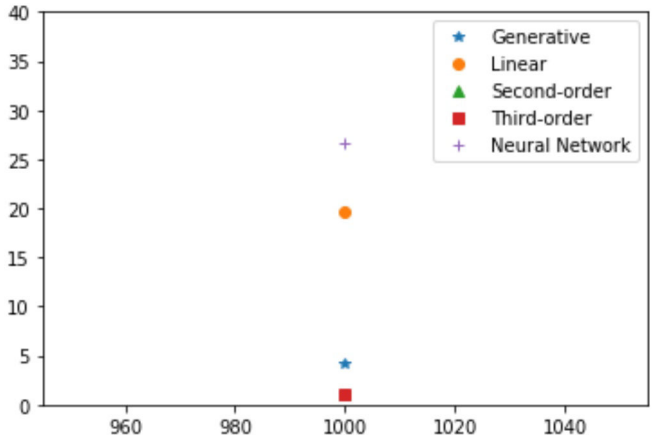
and it is quadratic.

In [107]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.spatial
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import math
import warnings;

warnings.simplefilter('ignore')

# 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 activa
tion
    # 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):
            update = np.dot(curGrad,curA.T)
            updateB = np.sum(curGrad,1).reshape(layer.b.shape)
            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 th
e
    # 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
```

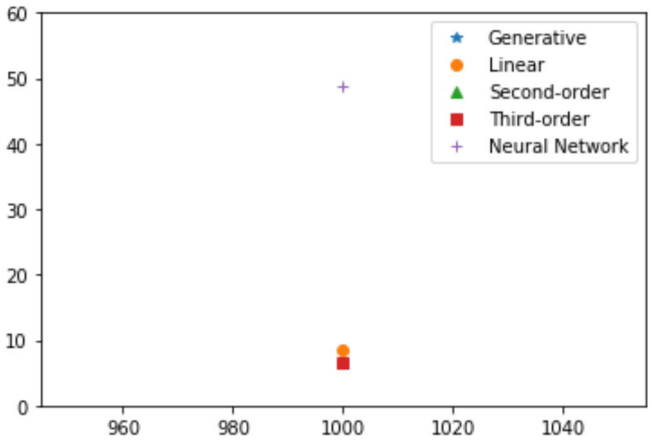
In [108]:
```
##########
## main ##
##########

# 4(b).1

n_train = range(10, 291, 20)

sensor_loc = generate_sensors()
errors = np.zeros( (15, 5), "float" )
np.random.seed(0)
for i, n in enumerate(n_train):
    distance, obj_loc = generate_dataset(sensor_loc, num_data=n)
    # Generative Model
    errors[i, 0] = GM_solver(distance, obj_loc)
    # OLS Linear:
    _, errors[i, 1] = OLS_polynomial_solver(1, distance, obj_loc)
    # OLS Second-order
    _, errors[i, 2] = OLS_polynomial_solver(2, distance, obj_loc)
    # OLS Third-order
    _, errors[i, 3] = OLS_polynomial_solver(3, distance, obj_loc)
    # Neural Network Model
    errors[i, 4] = NN_solver(distance, obj_loc)

plt.figure()
plt.plot(n_train, errors[:, 0], label="Generative")
plt.plot(n_train, errors[:, 1], label="Linear")
plt.plot(n_train, errors[:, 2], label="Second-order")
plt.plot(n_train, errors[:, 3], label="Third-order")
plt.plot(n_train, errors[:, 4], label="Neural Network")
plt.ylim((0, 100))
plt.legend(loc="best")
plt.show()
```



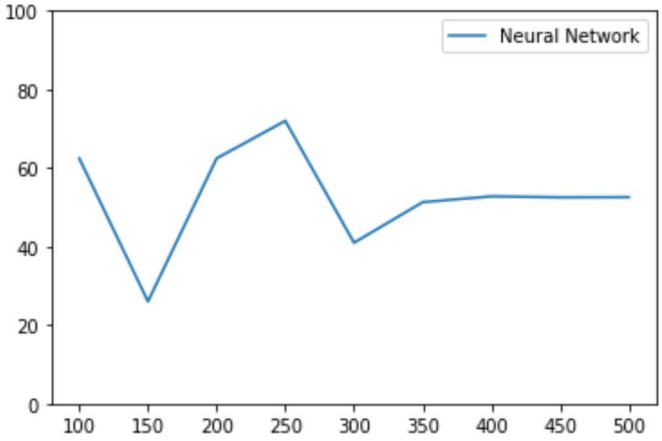
In [116]:
```
##########
## main ##
##########

# 4(b).2

n_test = [1000]

sensor_loc = generate_sensors()
errors = np.zeros( (1, 5), "float" )
np.random.seed(0)
for i, n in enumerate(n_test):
    distance, obj_loc = generate_dataset(sensor_loc, num_data=n)
    # Generative Model
    errors[i, 0] = GM_solver(distance, obj_loc)
    # OLS Linear:
    _, errors[i, 1] = OLS_polynomial_solver(1, distance, obj_loc)
    # OLS Second-order
    _, errors[i, 2] = OLS_polynomial_solver(2, distance, obj_loc)
    # OLS Third-order
    _, errors[i, 3] = OLS_polynomial_solver(3, distance, obj_loc)
    # Neural Network Model
    errors[i, 4] = NN_solver(distance, obj_loc)

plt.figure()
plt.plot(n_test, errors[:, 0], "*", label="Generative")
plt.plot(n_test, errors[:, 1], "o", label="Linear")
plt.plot(n_test, errors[:, 2], "^", label="Second-order")
plt.plot(n_test, errors[:, 3], "s", label="Third-order")
plt.plot(n_test, errors[:, 4], "+", label="Neural Network")
plt.ylim((0, 40))
plt.legend(loc="best")
plt.show()
```



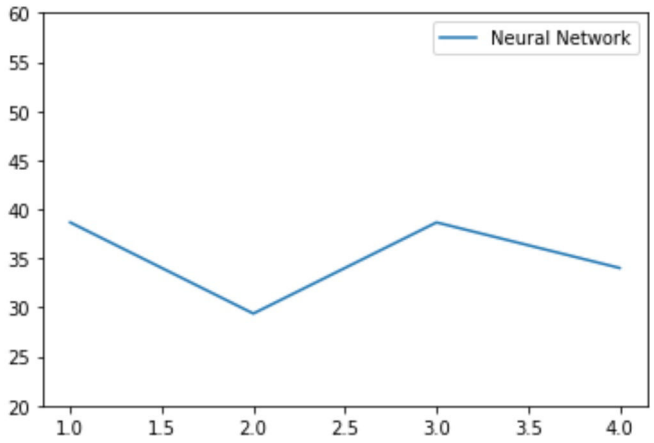
In [118]:
```
##########
## main ##
##########

# 4(b).3

n_test = [1000]

sensor_loc = generate_sensors()
errors = np.zeros( (1, 5), "float" )
np.random.seed(0)
for i, n in enumerate(n_test):
    distance, obj_loc = generate_dataset(sensor_loc, num_data=n, original_dist=
else)
    # Generative Model
    errors[i, 0] = GM_solver(distance, obj_loc)
    # OLS Linear:
    _, errors[i, 1] = OLS_polynomial_solver(1, distance, obj_loc)
    # OLS Second-order
    _, errors[i, 2] = OLS_polynomial_solver(2, distance, obj_loc)
    # OLS Third-order
    _, errors[i, 3] = OLS_polynomial_solver(3, distance, obj_loc)
    # Neural Network Model
    errors[i, 4] = NN_solver(distance, obj_loc)

plt.figure()
plt.plot(n_test, errors[:, 0], "*", label="Generative")
plt.plot(n_test, errors[:, 1], "o", label="Linear")
plt.plot(n_test, errors[:, 2], "^", label="Second-order")
plt.plot(n_test, errors[:, 3], "s", label="Third-order")
plt.plot(n_test, errors[:, 4], "+", label="Neural Network")
plt.ylim((0, 60))
plt.legend(loc="best")
plt.show()
```



In [134]:
```
##########
## main ##
##########

# 4(c)

def NN_solver_tune_1(l, distance, obj_loc, distance_test, obj_loc_test):
    x=distance
    y=obj_loc
    xtest=distance_test
    ytest=obj_loc

    activations = dict( eL = eL Activation,
                        tanh=TanhActivation,
                        linear=LinearActivation)
    lr = dict( eL =0.02,tanh=0.02,linear=0.005)
    names = [' eL ','linear','tanh']

    for key in [' eL ']:

        # Build the model
        activation = activations[key]
        model = Model(x.shape[1])
        model.addLayer( enseLayer(l,activation()))
        model.addLayer( enseLayer(l,activation()))
        model.addLayer( enseLayer(2,LinearActivation()))
        model.initialize( uadratic ost())

        # Train the model and display the results
        hist = model.train(x,y,500,G Optimizer(eta=lr[key]))
        y at = model.predict(x)
        squared_diff = np.square(y at - y)
        error = np.mean(np.sqrt(squared_diff[:, 0] + squared_diff[:, 1]))

    return error

l = range(100, 501, 50)
n_train = 200
n_test = 1000
errors = np.zeros( (len(l),), "float")

sensor_loc = generate_sensors()
np.random.seed(9001)
for i, ll in enumerate(l):
    distance, obj_loc = generate_dataset(sensor_loc, num_data=n_train)
    distance_test, obj_loc_test = generate_dataset(sensor_loc, num_data=n_test)
    # Neural Network Model
    errors[i] = NN_solver_tune_1(ll, distance, obj_loc, distance_test, obj_loc_t
est)
    mm = np.mean(errors[:i-1])

plt.figure()
plt.plot(l, errors, label="Neural Network")
plt.ylim((0, 100))
plt.legend(loc="best")
plt.show()
```


Neural Network
100
80
60
40
20
0
100
150
200
250
300
350
400
450
500

In [141]:
```
##########
## main ##
##########

# 4(d)

def NN_solver_tune_2(k, l, distance, obj_loc, distance_test, obj_loc_test):
    x=distance
    y=obj_loc
    xtest=distance_test
    ytest=obj_loc

    activations = dict( eL = eL Activation,
                        tanh=TanhActivation,
                        linear=LinearActivation)
    lr = dict( eL =0.02,tanh=0.02,linear=0.005)
    names = [' eL ','linear','tanh']

    for key in [' eL ']:

        # Build the model
        activation = activations[key]
        model = Model(x.shape[1])
        for _ in range(k):
            model.addLayer( enseLayer(l,activation()))
        model.addLayer( enseLayer(2,LinearActivation()))
        model.initialize( uadratic ost())

        # Train the model and display the results
        hist = model.train(x,y,500,G Optimizer(eta=lr[key]))
        y at = model.predict(x)
        squared_diff = np.square(y at - y)
        error = np.mean(np.sqrt(squared_diff[:, 0] + squared_diff[:, 1]))

    return error

k = range(1, 5)
n_train = 200
n_test = 200
errors = np.zeros( (len(k),), "float")

sensor_loc = generate_sensors()
np.random.seed(9001)
for i, kk in enumerate(k):
    if kk == 1:
        ll = 1000
    elif kk == 2:
        ll = 95
    elif kk == 3:
        ll = 67
    else:
        ll = 55
    distance, obj_loc = generate_dataset(sensor_loc, num_data=n_train)
    distance_test, obj_loc_test = generate_dataset(sensor_loc, num_data=n_test)
    # Neural Network Model
    errors[i] = NN_solver_tune_2(kk, ll, distance, obj_loc, distance_test, obj_l
oc_test)
    mm = np.mean(errors[:i-1])

plt.figure()
plt.plot(k, errors, label="Neural Network")
plt.ylim((20, 60))
```



In [147]:
```
##########
## main ##
##########

k = 2
l = 150

# 4(e)
sensor_loc = generate_sensors()
np.random.seed(9001)
n_train = 200
n_test = 1000
distance, obj_loc = generate_dataset(sensor_loc, num_data=n_train)
error = NN_solver_tune_2(k, l, distance, obj_loc, distance_test, obj_loc_test)
print(error)
```

```
15.201287385073272
```

In [146]:
```
## Problem 5

# Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================

"""A very simple MNIST classifier.

See extensive documentation at
https://www.tensorflow.org/get_started/mnist/beginners
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import sys

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

 LAGS = None


def main(_):
  # Import data
  mnist = input_data.read_data_sets( LAGS.data_dir, one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf. ariable(tf.zeros([784, 10]))
  b = tf. ariable(tf.zeros([10]))
  y = tf.matmul(x, W) + b

  # Define loss and optimizer
  y_ = tf.placeholder(tf.float32, [None, 10])

  # The raw formulation of cross-entropy,
  #
  #   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y)),
  #                                 reduction_indices=[1]))
  #
  # can be numerically unstable.
  #
  # So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
  # outputs of 'y', and then average across the batch.
  cross_entropy = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

```
 xtracting /tmp/tensorflow/mnist/input_data train-images-idx3-ubyte.gz
 xtracting /tmp/tensorflow/mnist/input_data train-labels-idx1-ubyte.gz
 xtracting /tmp/tensorflow/mnist/input_data t10k-images-idx3-ubyte.gz
 xtracting /tmp/tensorflow/mnist/input_data t10k-labels-idx1-ubyte.gz
0.9218

An exception has occurred, use  tb to see the full traceback.

System xit
```