MidTerm Assignment: notebook 1: Revision

Muhammad Wajahat Mirza

Net ID: mwm356

Total : 10 pts

Question 1.1. Statistical learning: Maximum likelihood (Total 5pts)

This exercise contains a pen and paper part and a coding part. You should submit the pen and paper either in lateX or take a picture of your written solution and join it to the Assignment folder.

We consider the dataset given below. This dataset was generated from a Gaussian distribution with a given mean $\mu = (\mu_1, \mu_2)$ and covariance matrix $\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$. We would like to recover the mean and variance from the data. In order to do this, use the following steps:

1. Write the general expression for the probability (multivariate (2D) Gaussian with diagonal covariance matrix) to observe a single sample
2. We will assume that the samples are independent and identically distributed so that the probability of observing the whole dataset is the product of the probabilties of observing each one of the samples $\left\{\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)})\right\}_{i=1}^{N}$. Write down this probability
3. Take the negative logarithm of this probability
4. Once you have taken the logarithm, find the expression for μ_1, μ_2, σ_1 and σ_2 by maximizing the probability.

I.1.1Solution: Mathematical Base

Solution Guide:
Equations in a Box are answers to questions asked above. Rest is the way to get them

Univariate Gaussian Distribution

1. Recall 1-dimensional Gaussian with mean paramete μ

$$p(x|\mu) = \frac{1}{\sqrt{2\pi}} exp\left[-\frac{1}{2}(x-\mu)^2\right]$$

1. This can also have variance parameter σ^2 that widens or narrows the Gaussian distribution

$$p(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left[-\frac{1}{2\sigma^2}(x-\mu)^2\right]$$

Multivariate Gaussian Distribution

1. This Gaussian can be extended to **Multivariate Gaussian** with co-variance matrix $\sum$

$$X = (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{D-1}, \vec{x}_D)$$
$$Moment-Parameterization : \mu = \mathbb{E}(X) = (\mu_1, \mu_2, \ldots, \mu_{D-1}, \mu_D)$$
$$\sigma^2 = \mathbb{E}[X - \mathbb{E}(X)]^2 = \mathbb{E}[X-\mu]^2$$
$$\Sigma = Cov(X) = \mathbb{E}\left[\vec{x} - \vec{\mu}\right]\left[\vec{x} - \vec{\mu}\right]^T$$
$$Mahalanobis-distance : \triangle^2 = \left[\vec{x} - \vec{\mu}\right]^T \Sigma^{-1}\left[\vec{x} - \vec{\mu}\right]$$

By Using: X, μ, σ^2, Σ i.e. equations 3 to 6, we get:

$$\boxed{p(\vec{x}|\vec{\mu},\Sigma) = \frac{1}{2\pi^{\frac{D}{2}}\sqrt{|\Sigma|}} exp\left[-\frac{1}{2}(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu})\right]}$$

where

$$\vec{x} \in \mathbb{R}^D, \vec{\mu} \in \mathbb{R}^D, \Sigma \in \mathbb{R}^{D\times D}$$

Diagonal Covariance Probability

1. Diagonal Covariance: Dimensions of x are independent product of multiple 1-D Gaussians

$$\boxed{p(\vec{\mathrm{x}} \mid \vec{\mu}, \Sigma) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi}\,\vec{\sigma}(d)} exp\left[-\frac{(\vec{\mathrm{x}}(d) - \vec{\mu}(d))^2}{2\vec{\sigma}(d)^2}\right]}$$

where

$$\Sigma = \begin{bmatrix} \vec{\mu}(1)^2 & 0 & 0 & 0 \\ 0 & \vec{\mu}(2)^2 & 0 & 0 \\ 0 & 0 & \vec{\mu}(3)^2 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \vec{\mu}(D)^2 \end{bmatrix}$$

Maximum Likelihood

1. To recover mean and variance, we COULD use standard Maximum Likelihood where probability of given data is maximized

$$X = (\vec{\mathrm{x}}_1, \vec{\mathrm{x}}_2, \ldots, \vec{\mathrm{x}}_{N-1}, \vec{\mathrm{x}}_N)$$

Let θ represent the parameters (μ, σ) of the two distributions. Then the probability of observing the data with parameterθ is called the likelihood.

$$p(X|\theta) = p(\vec{\mathrm{x}}_1, \vec{\mathrm{x}}_2, \ldots, \vec{\mathrm{x}}_N|\theta)$$

```
FOR independent Gaussian  samples
```

$$\boxed{p(X) = \prod_{i=1}^{N} p(\vec{\mathrm{x}}_i \mid \vec{\mu}_i, \Sigma_i)}$$

```
FOR identically Distributed
```

$$\boxed{p(X) = \prod_{i=1}^{N} p(\vec{\mathrm{x}}_i \mid \vec{\mu}, \Sigma)}$$

Negative Log-Maximum Likelihood

1. HOWEVER, rather than simple maximum likelihood, we use maximum of log-likelihood by taking log

$$\boxed{\sum_{i=1}^{N} \log \mathsf{p}(\vec{\mathrm{x}}_i | \vec{\mu}, \Sigma) = -\sum_{i=1}^{N} \log \frac{1}{2\pi^{\frac{D}{2}}\sqrt{|\Sigma|}} exp\left[-\frac{1}{2}(\vec{\mathrm{x}} - \vec{\mu})^T \Sigma^{-1} (\vec{\mathrm{x}} - \vec{\mu})\right]}$$

Finding vector $\vec{\mu}$ (μ_1, μ_2) by maximizing their probabilties:

1. **Max over μ**

$$\underset{\mu}{\operatorname{argmax}} = \frac{\partial}{\partial \mu}\left[-\sum_{i=1}^{N} \log \frac{1}{2\pi^{\frac{D}{2}}\sqrt{|\Sigma|}} exp\left[-\frac{1}{2}(\vec{\mathrm{x}} - \vec{\mu})^T \Sigma^{-1} (\vec{\mathrm{x}} - \vec{\mu})\right]\right] = 0$$

$$\therefore \frac{\partial}{\partial \mu}\left[\sum_{i=1}^{N} -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\vec{\mathrm{x}} - \vec{\mu})^T \Sigma^{-1} (\vec{\mathrm{x}} - \vec{\mu})\right]$$

$$\frac{\partial \vec{\mathrm{x}}^T \vec{\mathrm{x}}}{\partial \vec{\mathrm{x}}} = 2\vec{\mathrm{x}}^T \implies \frac{\partial}{\partial \mu}(\vec{\mathrm{x}} - \vec{\mu})^T(\vec{\mathrm{x}} - \vec{\mu}) = 2(\vec{\mathrm{x}} - \vec{\mu})^T$$

$$\sum_{i=1}^{N} \frac{1}{2} \times 2(\vec{\mathrm{x}} - \vec{\mu})^T \Sigma^{-1} = \vec{0}$$

Hence

$$\therefore \boxed{\vec{\mu} = \frac{1}{N}\sum_{i=1}^{N} \vec{\mathrm{x}}_i}$$

Finding vector $\vec{\Sigma}$ (σ_1, σ_2) matrix by maximizing their probabilties $\mathbf{\Sigma} = \left[\begin{array}{cc}

\sigma_1^2 & 0 \ 0 & \sigma_2^2 \end{array}\right]$

1. **Max over** Σ^{-1} by using Trace properties. Rewrite log-likelihood using **"Trace Trick"** and Let l be:

$$l = \sum_{i=1}^{N} -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\vec{\mathrm{x}} - \vec{\mu})^T\Sigma^{-1}(\vec{\mathrm{x}} - \vec{\mu})$$
$$\therefore -\frac{ND}{2}\log 2\pi + \frac{N}{2}\log|\Sigma^{-1}| - \frac{1}{2}\sum_{i=1}^{N}\mathrm{Tr}\left[(\vec{\mathrm{x}} - \vec{\mu})^T\Sigma^{-1}(\vec{\mathrm{x}} - \vec{\mu})\right]$$
$$\therefore -\frac{ND}{2}\log 2\pi + \frac{N}{2}\log|\Sigma^{-1}| - \frac{1}{2}\sum_{i=1}^{N}\mathrm{Tr}\left[(\vec{\mathrm{x}} - \vec{\mu})^T(\vec{\mathrm{x}} - \vec{\mu})\Sigma^{-1}\right]$$

Let

$$A = \Sigma^{-1}$$
$$\therefore -\frac{ND}{2}\log 2\pi + \frac{N}{2}\log|A| - \frac{1}{2}\sum_{i=1}^{N}\mathrm{Tr}\left[(\vec{\mathrm{x}} - \vec{\mu})^T(\vec{\mathrm{x}} - \vec{\mu})A\right]$$

Since $\frac{\partial \log|A|}{\partial A} = (A^{-1})^T; \frac{\partial \mathrm{Tr}(AB)}{\partial A} = B^T$

$$\frac{\partial l}{\partial A} = -0 + \frac{N}{2}(A^{-1})^T - \frac{1}{2}\sum_{i=1}^{N}\left[(\vec{\mathrm{x}} - \vec{\mu})(\vec{\mathrm{x}} - \vec{\mu})^T\right]^T$$
$$\frac{N}{2}\Sigma - \frac{1}{2}\sum_{i=1}^{N}(\vec{\mathrm{x}} - \vec{\mu})(\vec{\mathrm{x}} - \vec{\mu})^T$$
$$\therefore \frac{\partial l}{\partial A} = 0 \Longrightarrow \boxed{\Sigma = \frac{1}{N}(\vec{\mathrm{x}} - \vec{\mu})(\vec{\mathrm{x}} - \vec{\mu})^T}$$

I.1.2 Programming

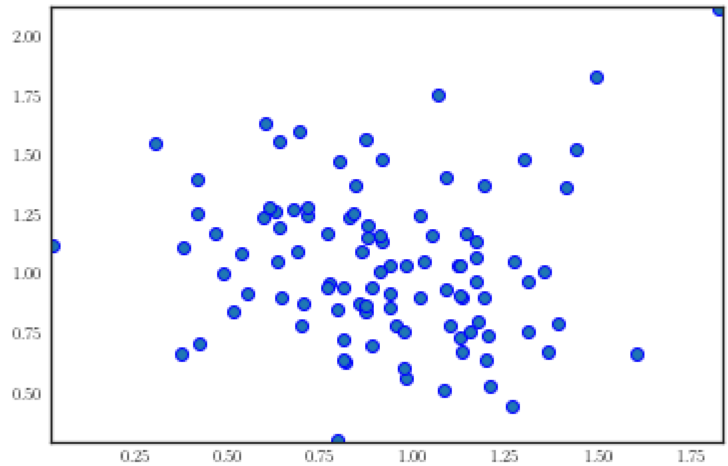
Code on following dataset to display gaussian distribution using log-maximum likelihood

Import Respective Libraries

In [902]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import matplotlib.pyplot as plt
from matplotlib import cm
from scipy.stats import multivariate_normal
```

Load Data

In [903]:
```
X = loadmat('dataNotebook1_Ex1.mat')['X']

plt.scatter(X[:,0], X[:,1])
plt.show()
```



5. Once you have you estimates for the parameters of the Gaussian distribution, plot the level lines of that distribution on top of the points by using the lines below.

Solution

Please note that my solution to above question also includes outliers into the Gaussian distribution.

Compute $\vec{\mu}, \sigma^2$ and use them for Scipy function multivariate_normal.pdf( )

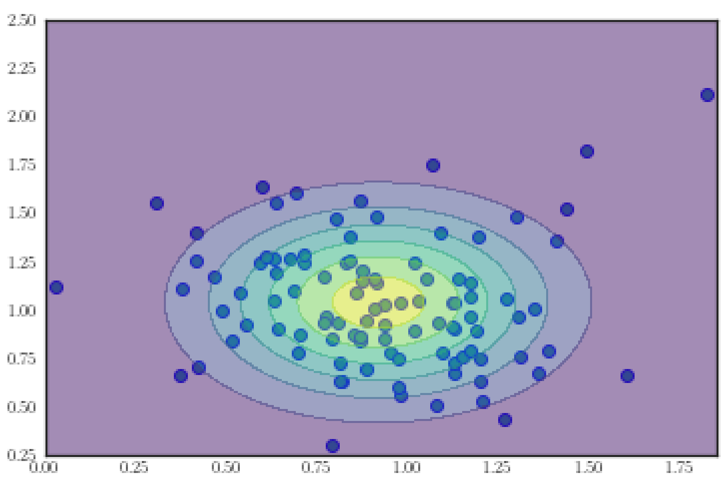
$$\boxed{\vec{\mu} = \frac{1}{N}\sum_{i=1}^{N}\vec{x}_i}$$

In [904]:
```
def compute_mu_scipy(X):

    N = len(X)
    mu = (1/N)*np.sum(X)

    return mu
```

In [905]:
```
def multivariate_normal_pdf_scipy(X):
    x1 = np.linspace(0, 1.85, 100)
    x2 = np.linspace(0.25, 2.5, 100)

    xx1, xx2 = np.meshgrid(x1, x2)

    from scipy.stats import multivariate_normal

    xmesh = np.vstack((xx1.flatten(), xx2.flatten())).T

    mu1 = compute_mu_scipy(X[:,0])
    mu2 = compute_mu_scipy(X[:,1])
    print("mu1 is: {} \nmu2 is: {}".format(mu1, mu2))

    sigma1 = np.std(X[:,0])
    sigma2 = np.std(X[:,1])

    sigma = np.zeros((2,2))
    sigma[0,0] = sigma1**2
    sigma[1,1] = sigma2**2

    print("Sigma1 is: {} \nSigma2 is: {} \nSigma Vector is: \n{}".format
(sigma1, sigma2, sigma))

    y = multivariate_normal.pdf(xmesh, mean=[mu1,mu2], cov=sigma)
    print("Returned Y is: ",y)
    return x1,x2,xx1, xx2, y
```

In [906]:
```
def plot_scipy_MND(X):

    x1,x2,xx1, xx2, y = multivariate_normal_pdf_scipy(X)
    plt.scatter(X[:,0], X[:,1])
    plt.contourf(xx1, xx2, np.reshape(y, (100, 100)), zdir='z', offset=-
0.15, cmap=cm.viridis, alpha=0.5)
    plt.show()
plot_scipy_MND(X)
```

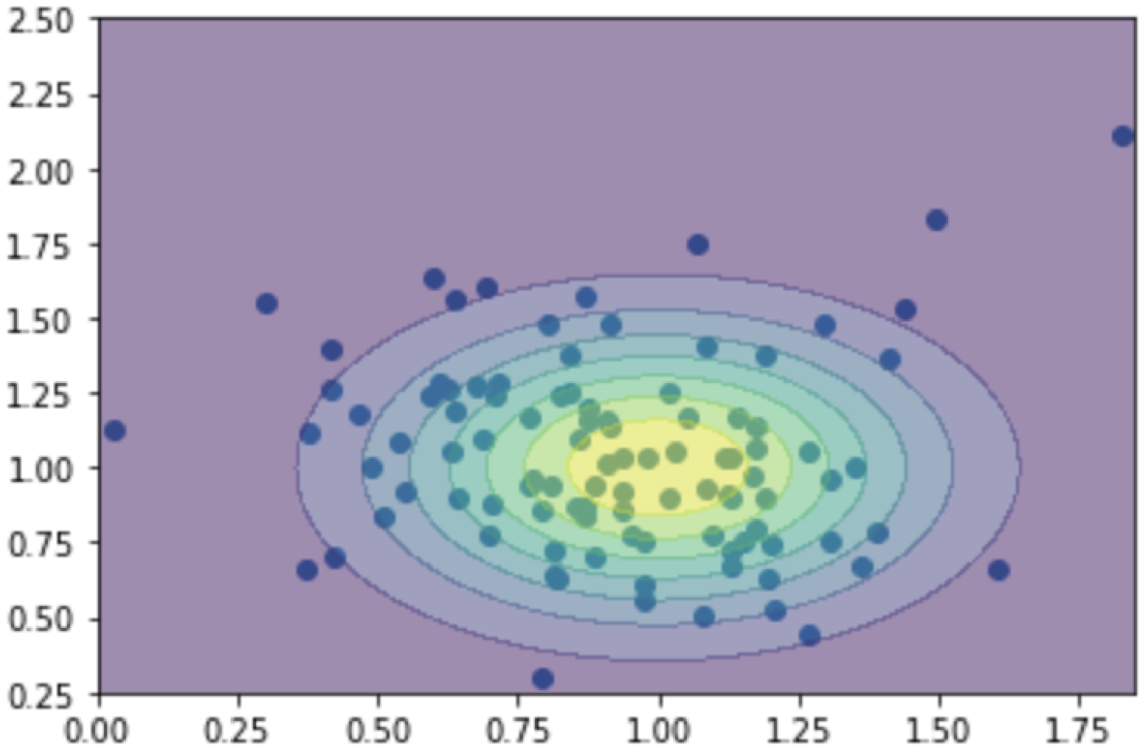
```
mu1 is: 0.9196903048948579
mu2 is: 1.0402703261447646
Sigma1 is: 0.30398425815578267
Sigma2 is: 0.3202124191447788
Sigma Vector is:
[[0.09240643 0.        ]
 [0.         0.10253599]]
Returned Y is:  [8.00385717e-04 9.62167394e-04 1.15228727e-03 ... 6.718
98822e-07
 5.59836359e-07 4.64704787e-07]

/Users/mirza/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launch
er.py:5: UserWarning: The following kwargs were not used by contour: 'z
dir', 'offset'
  """
```



From Professor: Solution should look like this

In [925]:
```
from IPython.display import Image
#<img src="solution_gaussian.png" width="400" />
Image('solution_gaussian.png')
```

Out[925]:



Optional Additional Work for Q 1.1 without using Scipy Library

Extra Optional Work: Compute $\vec{\mu}$

$$\boxed{\vec{\mu} = \frac{1}{N}\sum_{i=1}^{N}\vec{x}_i}$$

In [907]:
```
def compute_mu(X):

    N = len(X)
    mu = (1/N)*np.sum(X)
#     mu = mu.reshape(-1,1)

    return mu, N
```

Extra Optional Work: Compute Σ

$$\boxed{\Sigma = \frac{1}{N}(\vec{x} - \vec{\mu})(\vec{x} - \vec{\mu})^T}$$

In [908]:
```
def compute_sigma(X):

    mu, N = compute_mu(X)
    sigma = (1/N)*(X - mu)*(X-mu).T

    return sigma
```

Extra Optional Work: Multivariate Gaussian Distribution

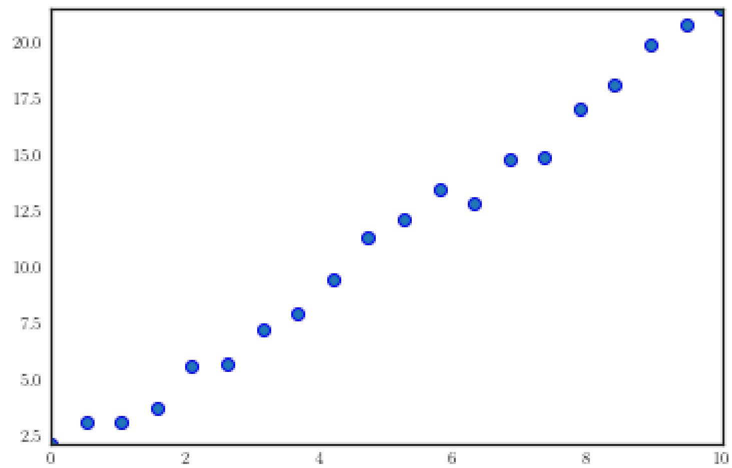
$$\boxed{p(\vec{x} \mid \vec{\mu}, \Sigma) = \frac{1}{\sqrt{2\pi^{D}|\Sigma|}} exp\left[-\frac{1}{2}(\vec{x} - \vec{\mu})^{T}\Sigma^{-1}(\vec{x} - \vec{\mu})\right]}$$

In [909]:
```
def multivariate_normal_pdf(X):

    X = X.reshape(-1,1)
    mu, N = compute_mu(X)

    sigma = compute_sigma(X)
    sigma_determinant = np.linalg.det(sigma)

    sigma_inverse = np.linalg.pinv(sigma)

    mu = mu.reshape(-1,1)
    instances, columns = sigma.shape


#     first_denominator = (2 * np.pi)**(np.true_divide(instances,2)) * n
p.sqrt(sigma_determinant)
    first_denominator = np.sqrt(((2 * np.pi)**(instances))*sigma_determi
nant)
    exponential_nominator = -(1/2) * (X - mu).T * sigma_inverse * (X - m
u)
    result = (np.true_divide(1, first_denominator)) * np.exp(exponential
_nominator)

    return result, sigma
```

In [910]:
```
def solve_for_results():

    value = 100
    X = np.linspace(0, 1.85, value)
    Y = np.linspace(0.25, 2.5, value)

    XX, YY = np.meshgrid(X, Y)

    data = [X, Y]
    Z = []
    for i in data:
        z, sigma = np.array(multivariate_normal_pdf(i))
        Z.append(z)

    return X,Y,Z,sigma

def plot_results():

    X, Y, Z = solve_for_results()

    fig = plt.figure(figsize = (10,10))
    ax = fig.gca(projection='3d')
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, linewidth=1, antialia
sed=True,
                    cmap=cm.viridis)

    cset = ax.contourf(X, Y, Z, zdir='z', offset=-0.15, cmap=cm.viridis)

    # Adjust the limits, ticks and view angle
    ax.set_zlim(-0.15,0.5)
    ax.set_zticks(np.linspace(0,0.2,5))
    ax.view_init(20, 25)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_title('Multivariate Gaussian Sigma = {}'.format(Sigma))

    plt.show()
# solve_for_results()
```

1.2. We consider the following linear regression problem. (Total 5pts)

In [911]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat

X_original = loadmat('MidTermAssignment_dataEx2.mat')['MidTermAssignment
_dataEx2']

plt.scatter(X_original[:,0], X_original[:,1])
plt.show()
```



Questions 1.2/2.1/2.2

Solve the ℓ_2 regularized linear regression problem through the normal equations (be careful that you have to take the $\ell_2$regularization into account). Then double-check your solution by comparing it with the regression function from scikit learn. Plot the result below.

Solution

Mathematical Base

1. Loss Function Equation

$$l(\beta) = \sum_{i=1}^{N} (t^{(i)} - X\vec{\beta})^2$$

`Vectorized Form`

$$\sum_{i=1}^{N} (V_i)^2 = \vec{\mathrm{v}}^T \vec{\mathrm{v}} \implies l(\beta) = (t^{(i)} - X\vec{\beta})^T (t^{(i)} - X\vec{\beta})$$

2. Normal Equation: After taking derivative of Loss func i.e. $l(\beta)$, Vectorized Normal Equ is

$$\vec{\beta} = (X^T X)^{-1} X^T \vec{\mathrm{t}}$$

3. Ridge Regularized Normal Equation:

$$\vec{\beta} = \left[(X^T X + \lambda I)^{-1} X^T \vec{\mathrm{t}} \right]$$

Loading Data

In [912]:
```
X = np.vstack(X_original[:,0])
ones = np.vstack(np.ones(X.shape))
X = np.hstack((ones,X))
target = np.vstack(X_original[:,1])
print("Shape of X: {} \nShape of target: {}".format(X.shape, target.shap
e))
```

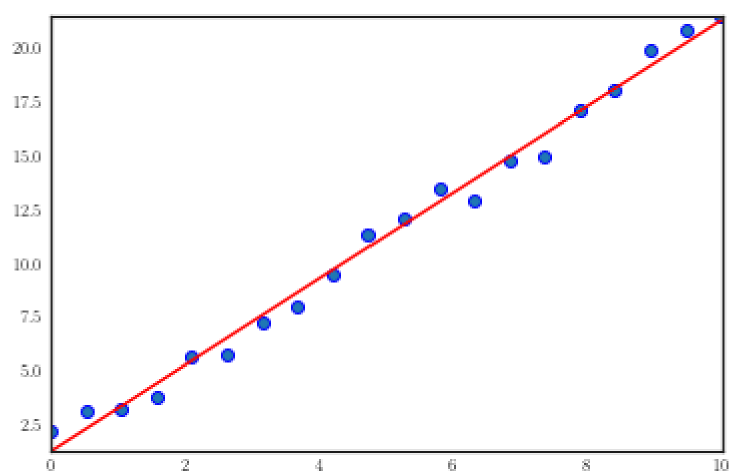
```
Shape of X: (20, 2)
Shape of target: (20, 1)
```

In [913]:
```
def prediction(X, beta):
    result = np.dot(X, beta)
    return result
```

Non-Regularized Normal Equation

In [914]:
```
def Vectorized_closed_form(X, target):

    target = np.mat(target)
    left_matrix = np.linalg.inv(np.dot(X.T, X))
    right_matrix = np.dot(X.T, target)
    beta = np.dot(left_matrix,right_matrix)
    print("Our Non-regularized beta is: \n{}".format(beta))

    return beta

beta_1 = Vectorized_closed_form(XX, target)
print("Shape of returned predict array",prediction(X, beta_2).shape)
print("Non-Regularized Normal Equation yields following Regression")
plt.figure()
plt.scatter(X_original[:,0], target)
plt.plot(X_original[:,0], prediction(X, beta_1), color = 'red')
plt.show()
```

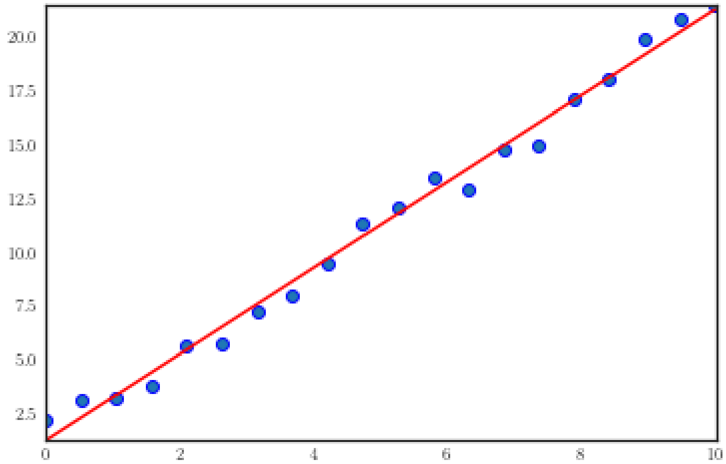
```
Our Non-regularized beta is:
[[1.2375465 ]
 [2.00737626]]
Shape of returned predict array (20, 1)
Non-Regularized Normal Equation yields following Regression
```



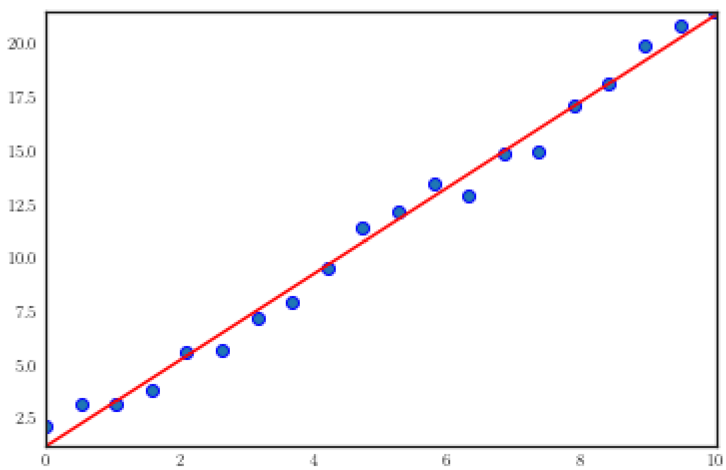
Regularized Normal Equation with multiple Lambda λ Values

In [915]:
```
def Regularized_Vectorized_closed_form(X, target,lambda0):
#     lambda0 = 1

    target = np.mat(target)
    left_matrix = np.linalg.inv(np.dot(X.T, X) + np.dot(lambda0,np.ident
ity(target.shape[1])))
    right_matrix = np.dot(X.T, target)
    beta = np.dot(left_matrix,right_matrix)
    print("Our Regularized beta with Lambda value {} is: \n{}".format(la
mbda0, beta))

    return beta

lambda0 = [0.01,0.1,1,10]
for i in lambda0:
    beta_2 = Regularized_Vectorized_closed_form(X, target,i)
    print("Shape of returned predict array",prediction(X, beta_2).shape)
    print("Regularized Normal Equation with Lambda value {} yields follo
wing Regression".format(i))
    plt.figure()
    plt.scatter(X_original[:,0], target)
    plt.plot(X_original[:,0], prediction(X, beta_2), color = 'red')
    plt.show()
```

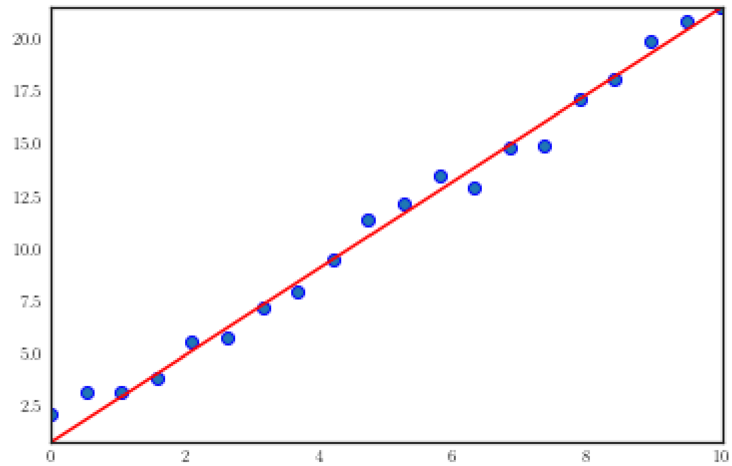
```
Our Regularized beta with Lambda value 0.01 is:
[[1.23240801]
 [2.00807991]]
Shape of returned predict array (20, 1)
Regularized Normal Equation with Lambda value 0.01 yields following Reg
ression
```



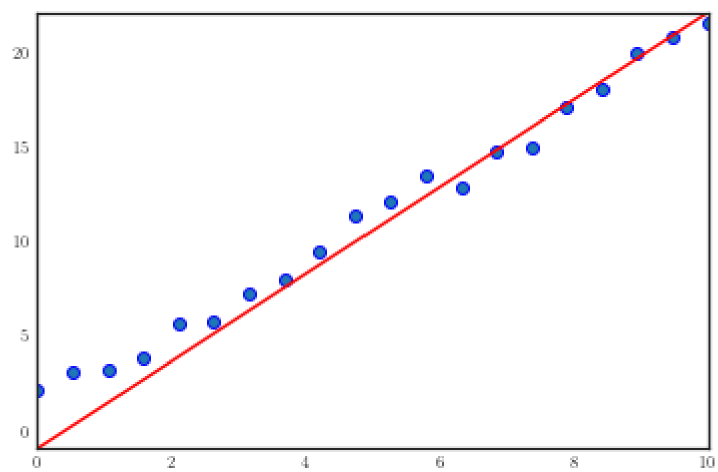

```
Our Regularized beta with Lambda value 0.1 is:
[[1.18678599]
 [2.01432725]]
Shape of returned predict array (20, 1)
Regularized Normal Equation with Lambda value 0.1 yields following Regr
ession
```




```
Our Regularized beta with Lambda value 1 is:
[[0.78493727]
 [2.06935518]]
Shape of returned predict array (20, 1)
Regularized Normal Equation with Lambda value 1 yields following Regres
sion
```



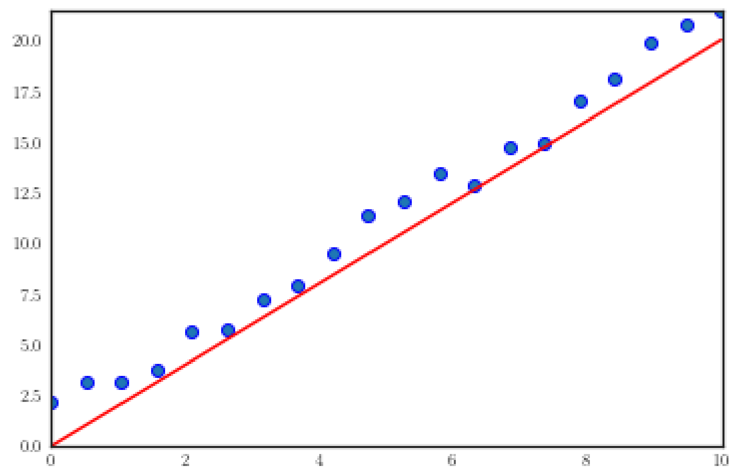
```
Our Regularized beta with Lambda value 10 is:
[[-0.93486862]
 [ 2.30486013]]
Shape of returned predict array (20, 1)
Regularized Normal Equation with Lambda value 10 yields following Regre
ssion
```



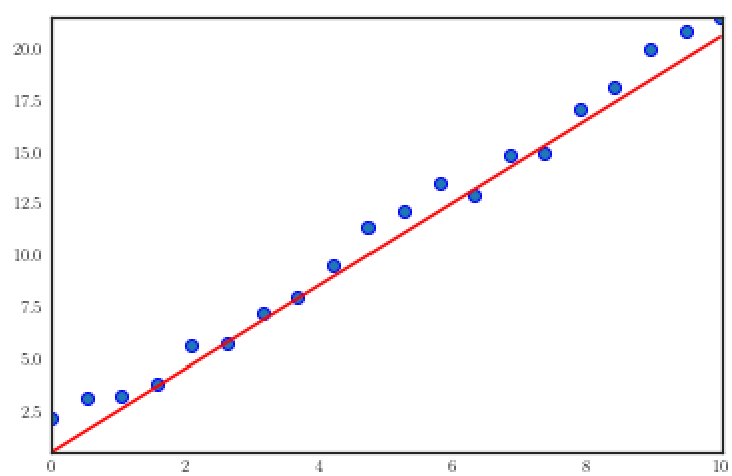
Verification from Scikit Learn Model

In [916]:
```
from sklearn.linear_model import Ridge

def Scikit_Ridge_Linear_Regression(X_original, X, target):
    bias_list = [0,0.5, 1]

    for bias in bias_list:
        # ==================================Building Model==============
================================================
        model = Ridge(alpha = 0.1)
        fit = model.fit(X,target)
        ridgeCoefs = model.coef_
        predict = model.predict(ridgeCoefs)
        y_hat = np.dot(X, ridgeCoefs.T)

        print("Our Fit Model \n",fit)
        print("Our Coefficients with (current) Bias value '{}' are: \n{}
".format(bias, ridgeCoefs+bias))
        print("predcit from scikit model: ",predict)
        print("Following is the Scikit Normal Equation/Ridge Linear Regr
ession with Bias value '{}'".format(bias))

        # ==================================Plot Graph==================
============================================
        plt.figure()
        plt.scatter(X_original[:,0], target)
        plt.plot(X_original[:,0], y_hat+bias, color = 'red')
        plt.show()

Scikit_Ridge_Linear_Regression(X_original, X, target)
```

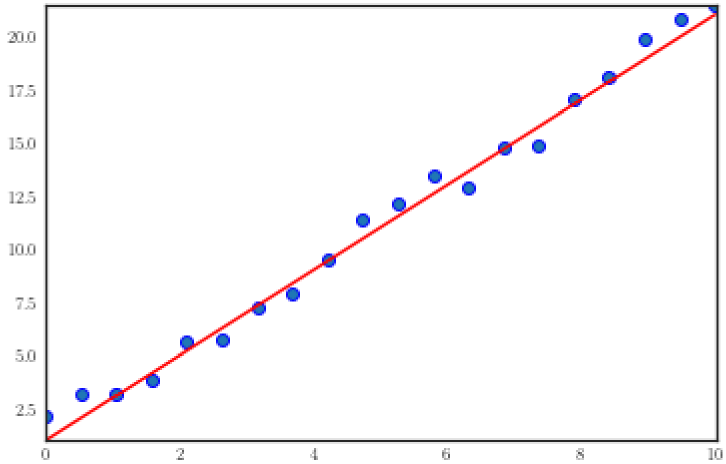
```
Our Fit Model
 Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
Our Coefficients with (current) Bias value '0' are:
[[0.         2.00628713]]
predcit from scikit model:  [[5.26818018]]
Following is the Scikit Normal Equation/Ridge Linear Regression with Bi
as value '0'
```




```
Our Fit Model
 Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
Our Coefficients with (current) Bias value '0.5' are:
[[0.5        2.50628713]]
predcit from scikit model:  [[5.26818018]]
Following is the Scikit Normal Equation/Ridge Linear Regression with Bi
as value '0.5'
```




```
Our Fit Model
 Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
Our Coefficients with (current) Bias value '1' are:
[[1.         3.00628713]]
predcit from scikit model:  [[5.26818018]]
Following is the Scikit Normal Equation/Ridge Linear Regression with Bi
as value '1'
```



Questions 2.3

2.3. **Kernel Ridge regression.** Given the 'Normal Equations' solution to the regularized regression model, we now want to turn the regression model into a formulation over kernels.

2.3.1. Start by showing that this solution can read as

$$\beta = \mathbf{X}^T(\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}$$

where $\mathbf{K}$ is the kernel matrix defined from the scalar product of the prototypes, i.e. $\mathbf{K}_{i,j} = \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)})^T(\mathbf{x}^{(j)})$.

Solution 2.3.1

1. Substitute K into the Equation:

$$\mathbf{K} = \mathbf{X}\mathbf{X}^T$$

Extra Work: (Optional) Proof

1. Our Normal Equation is:
$$\overrightarrow{\beta} = (X^T X)^{-1} X^T \overrightarrow{\mathrm{t}}$$
2. Suppose $(X^T X)^{-1}$ exists, then let $\widehat{\beta}_{ML}$ be:
$$\widehat{\beta}_{ML} = (X^T X)^{-1} X^T \overrightarrow{\mathrm{t}}$$
$$\therefore (X^T X)(X^T X)^{-1}(X^T X)^{-1} X^T \overrightarrow{\mathrm{t}}$$
$$\therefore (X^T X)(X^T X)^{-2} X^T \overrightarrow{\mathrm{t}}$$
$$\widehat{\beta}_{ML} \simeq X^T \alpha$$

where $\alpha = X(X^T X)^{-2} X^T \overrightarrow{\mathrm{t}}$

1. Get **Gram Matrix** if we want to predict the y values from X values:
$$X\widehat{\beta}_{ML} = XX^T \alpha = K\alpha$$

1. Let our Ridge Regularized Normal Equation be $\widehat{\beta}_{MAP}$:
$$\widehat{\beta}_{MAP} = (X^T X + \lambda I)^{-1} X^T \overrightarrow{\mathrm{t}}$$
$$(X^T X + \lambda I)\widehat{\beta}_{MAP} = X^T \overrightarrow{\mathrm{t}}$$
$$X^T X\widehat{\beta}_{MAP} + \lambda\widehat{\beta}_{MAP} = X^T \overrightarrow{\mathrm{t}}$$
$$\lambda\widehat{\beta}_{MAP} = X^T \left(\overrightarrow{\mathrm{t}} - X\widehat{\beta}_{MAP}\right)$$
$$\widehat{\beta}_{MAP} = \lambda^{-1} X^T \left(\overrightarrow{\mathrm{t}} - X\widehat{\beta}_{MAP}\right)$$
$$\widehat{\beta}_{MAP} = X^T \alpha$$

where $\alpha = \lambda^{-1} \left(\overrightarrow{\mathrm{t}} - X\widehat{\beta}_{MAP}\right)$

1. Solve for α, use Gram Matrix equation and substitute the equation:
$$\lambda\alpha = \overrightarrow{\mathrm{t}} - X\widehat{\beta}_{MAP}$$
$$\lambda\alpha = \overrightarrow{\mathrm{t}} - XX^T\alpha$$
$$\left(XX^T + \lambda\mathbf{I}_N\right)\alpha = \overrightarrow{\mathrm{t}}$$
$$\alpha = \left(XX^T + \lambda\mathbf{I}_N\right)^{-1} \overrightarrow{\mathrm{t}}$$
Substitute XX^T for K:
$$\alpha = (K + \lambda\mathbf{I}_N)^{-1} \overrightarrow{\mathrm{t}}$$
2. Substitude $Equation\ 55$ into $Equation\ 50$
$$\beta = X^T (K + \lambda\mathbf{I}_N)^{-1} \overrightarrow{\mathrm{t}}$$

Question 2.3.2.

Given this, the classifier can read as $f(\mathbf{x}) = \beta^T\mathbf{x} = \sum_{i=1}^N \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$. What are the α in this case?

Solution 2.3.2

α in this case are $weights$

Question 2.3.3.

We will apply this idea to text data. Using kernels with text data is interesting because it is usually easier to compare documents than to find appropriate features to represent those documents. The file 'headlines_train.txt' contains a few headlines, some of them being about finance, others being about weather forecasting. Use the first group of lines below to load those lines and their associated targets (1/0).

Solution 2.3.3

In [917]:
```
# Start by loading the file using the lines below
import numpy as np

def load_text_train_data():
    f = open('headlines_train.txt', "r")
    lines = f.readlines()
    f.close()

    sentences = ['Start']
    target = [0]

    for l in np.arange(len(lines)-2):
        if l%2 == 0:
            lines_tmp = lines[l]
            lines_tmp = lines_tmp[:-1]
            sentences.append(lines_tmp)
            if lines_tmp[-1] == ' ':
                target.append(float(lines_tmp[-2]))
            else:
                target.append(float(lines_tmp[-1]))

    sentences = sentences[1:]
    target = target[1:]
    print("Example of Sentence: {} \
          \n\nExamples of Target: {} ".format(sentences[4], target[:10
]))
    return sentences,target

sentences, target = load_text_train_data()
```

```
Example of Sentence: TEMPERATURES have soared this week, with sunny ski
es and clear days for much of the UK. However, this will soon change as
the forecast makes for a much colder weekend.0

Examples of Target: [1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

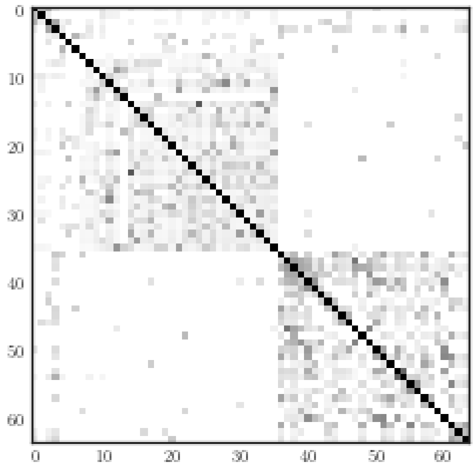
Question 2.3.4.

Now use the lines below to define the kernel. The kernel is basically built by generating a TF-IDF vector for each sentence and comparing those sentences through a cosine similarity measure. the variable 'kernel' the kernel matrix, i.e. $\kappa(i, j) = \frac{\phi_i^T \phi_j}{\|\phi_i\| \|\phi_j\|}$ where the ϕ_i encodes the tf-idf vectors. Use the lines below to compute the kernel matrix.

Solution 2.3.4

In [918]:
```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics.pairwise import pairwise_kernels
import matplotlib.pyplot as plt


model = TfidfVectorizer(max_features=100, stop_words='english',
                                 decode_error='ignore')

TF_IDF = model.fit_transform(sentences)
feature_names = model.get_feature_names()
kernel = cosine_similarity(TF_IDF)


print("Our Model \n {}".format(model))
print("\n")
print("TF-IDF Shape: {}".format(TF_IDF.shape))
print("TF-IDF Example: \n{}".format(TF_IDF[5]))
print("\nFeature Names: \n {}".format(feature_names))
print("\n")
print("Shape of Kernel Matrix (an array of shape (X,Y)): {}\
     \n \nA example of Kernel Matrix Value (15): \n {}".format(kernel.sh
ape, kernel[15]))

plt.imshow(kernel)
plt.show()
```

```
Our Model
 TfidfVectorizer(analyzer='word', binary=False, decode_error='ignore',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_featur
es=100,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=N
one,
                smooth_idf=True, stop_words='english', strip_accents=No
ne,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use_idf=True, vocabulary=None)


TF-IDF Shape: (64, 100)
TF-IDF Example:
  (0, 4)        0.5886777990045652
  (0, 45)       0.5540008103949683
  (0, 44)       0.5886777990045652

Feature Names:
 ['50s', '60s', 'act', 'additional', 'afternoon', 'ahead', 'areas', 'as
ian', 'associated', 'backstop', 'banks', 'beginning', 'best', 'bring',
'cash', 'cloud', 'come', 'conditions', 'coronavirus', 'crisis', 'day',
'days', 'dump', 'early', 'economic', 'expected', 'fall', 'fear', 'fed',
'federal', 'financial', 'flash', 'flooding', 'followed', 'forecast', 'f
orecasters', 'friday', 'funds', 'gauge', 'heavy', 'high', 'hit', 'hous
e', 'investors', 'late', 'likely', 'market', 'markets', 'markets1', 'mo
ney', 'morning', 'municipal', 'national', 'new', 'night', 'package', 'p
andemic', 'passed', 'plunge', 'possible', 'prime', 'puget', 'rain', 'ra
infall', 'rains', 'representatives', 'reserve', 'said', 'say', 'servic
e', 'severe', 'showers', 'shows', 'skies', 'sound', 'south', 'stock',
'stocks', 'storm', 'storms', 'street', 'strong', 'temperatures', 'ter
m', 'threat', 'thunderstorm', 'thunderstorms', 'thursday', 'treasury',
'trillion', 'tuesday', 'turbulence', 'vix', 'wall', 'way', 'weather',
'week', 'weekend', 'wind', 'winds']


Shape of Kernel Matrix (an array of shape (X,Y)): (64, 64)

A example of Kernel Matrix Value (15):
 [0.         0.         0.         0.         0.         0.
 0.         0.         0.25138966 0.13817878 0.         0.06829762
 0.06608462 0.         0.08675784 1.         0.252001   0.27728762
 0.         0.         0.06457477 0.06745497 0.09723764 0.32477066
 0.11848253 0.0510486  0.         0.39888888 0.06315905 0.29708278
 0.05490185 0.21759867 0.0597897  0.08531799 0.47439759 0.05212403
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
```


0
10
20
30
40
50
60
0
10
20
30
40
50
60

In [919]:
```
stop_word_list = model.get_stop_words()
print("Stop word List Example: \n{}\n".format(stop_word_list))
```

```
Stop word List Example:
frozenset({'where', 'whereby', 'who', 'bottom', 'co', 'after', 'whereve
r', 'being', 'made', 'into', 'nevertheless', 'thereby', 'almost', 'et
c', 'be', 'their', 'sixty', 'seeming', 'own', 'it', 'full', 'here', 'fr
ont', 'someone', 'several', 'down', 'cry', 'inc', 'many', 'nor', 'sid
e', 'four', 'now', 'himself', 'indeed', 'anywhere', 'themselves', 'anyo
ne', 'thereafter', 'fill', 'its', 'put', 'nobody', 'per', 'such', 'see
m', 'nowhere', 'for', 'are', 'myself', 'became', 'noone', 'three', 'eig
ht', 'elsewhere', 'hence', 'amount', 'mill', 'through', 'has', 'will',
'our', 'among', 'back', 'thus', 'she', 'he', 'might', 'whereupon', 'her
eafter', 'last', 'in', 'alone', 'interest', 'found', 'them', 'can', 'ra
ther', 'beforehand', 'seemed', 'both', 'only', 'everywhere', 'must', 't
hen', 'twenty', 'within', 'which', 'again', 'whereafter', 'enough', 've
ry', 'by', 'take', 'less', 'go', 'call', 'anything', 'was', 'amongst',
'next', 'some', 'anyhow', 'whereas', 'how', 'or', 'most', 'though', 'd
e', 'i', 'herself', 'thereupon', 'because', 'against', 'former', 'fift
y', 'first', 'up', 'than', 'never', 'either', 'during', 'serious', 'the
refore', 'detail', 'at', 'beside', 'ever', 'six', 'until', 'forty', 'be
yond', 'twelve', 'whither', 'seems', 'were', 'hasnt', 'hereupon', 'your
self', 'hundred', 'herein', 'whenever', 'each', 'however', 'between',
'whose', 'give', 'once', 'not', 'find', 'see', 'couldnt', 'nothing', 'm
uch', 'above', 'that', 'bill', 'done', 'empty', 'otherwise', 'ie', 'alr
eady', 'another', 'somewhere', 'itself', 'but', 'third', 'thin', 'besid
es', 'over', 'thence', 'ours', 'ten', 'amoungst', 'with', 'via', 'els
e', 'your', 'would', 'if', 'cant', 'along', 'hers', 'every', 'out', 'pa
rt', 'something', 'cannot', 'before', 're', 'around', 'may', 'more', 'n
amely', 'yet', 'since', 'whether', 'yours', 'and', 'also', 'five', 'sho
uld', 'of', 'nine', 'yourselves', 'anyway', 'others', 'sometime', 'thi
s', 'had', 'toward', 'becomes', 'to', 'further', 'us', 'afterwards', 'u
n', 'any', 'sincere', 'system', 'get', 'one', 'whoever', 'same', 'wha
t', 'could', 'under', 'my', 'ourselves', 'ltd', 'on', 'describe', 'ont
o', 'his', 'no', 'often', 'even', 'latter', 'wherein', 'few', 'latterl
y', 'upon', 'without', 'they', 'somehow', 'everything', 'hereby', 'alwa
ys', 'off', 'these', 'move', 'behind', 'when', 'please', 'is', 'too',
'neither', 'together', 'therein', 'an', 'we', 'her', 'all', 'eleven',
'so', 'top', 'due', 'been', 'become', 'name', 'those', 'whence', 'two',
'show', 'becoming', 'thru', 'across', 'why', 'thick', 'perhaps', 'fir
e', 'keep', 'eg', 'mostly', 'mine', 'whatever', 'although', 'formerly',
'about', 'least', 'throughout', 'none', 'towards', 'am', 'me', 'the',
'except', 'meanwhile', 'do', 'a', 'whom', 'you', 'con', 'moreover', 'wh
ile', 'everyone', 'him', 'still', 'well', 'whole', 'as', 'fifteen', 'be
low', 'have', 'sometimes', 'from', 'other', 'there'})
```

Question 2.3.4.

Once you have the kernel matrix, compute the weights α of the classifier $y(\mathbf{x}) = \sum_{i \in D} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$.

Solution 2.3.4

$$\beta = \mathbf{X}^T(\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\mathbf{t}$$

$\mathbf{K}_{i,j} = \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)})^T(\mathbf{x}^{(j)})$

$\mathbf{K} = \mathbf{X}\mathbf{X}^T$

In [920]:
```
# compute the alpha weights
def alpha_weights(X,kernel, target):

    lambda0 = 0.1
    K = np.dot(X,X.T)

    center = np.linalg.inv(kernel + lambda0*np.identity(X.shape[0]))

#     beta = X.T @ center @ target
    beta = center @ target

    print("Shape of weights: ",beta.shape, "\n")
    return beta

weights = alpha_weights(TF_IDF, kernel,target)
```

```
Shape of weights:  (64,)
```

Question 2.3.5.

Now that you have the weights, we want to apply the classifier to a few new headlines. Those headlines are stored in the file 'headlines*test.txt'. Use the lines below to load those sentences and compute their TF-IDF representation. the classifier $y(\mathbf{x}) = \sum*{i\in \mathcal{D}}\alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$

Solution 2.3.5

In [921]:

```
# Start by loading the file using the lines below
import numpy as np
def load_data_text_test():
    f = open('headlines_test.txt', "r")
    lines = f.readlines()
    f.close()

    sentences_test = ['Start']
    for l in np.arange(len(lines)):

        if l%2 == 0:
            lines_tmp = lines[l]
            lines_tmp = lines_tmp[:-1]
            sentences_test.append(lines_tmp)


    sentences_test = sentences_test[1:]

    print("Example of Test Sentence: \n{}\n".format(sentences_test[3]))
    return sentences_test
sentences_test = load_data_text_test()
```

```
Example of Test Sentence:
Money market mutual funds and exchange traded funds accounted for $286b
n of inflows for the period
```

In [922]:
```
'''Compute Test_F and print Relevent Information'''
test_F = np.hstack((tfidf_test.todense(), np.zeros((rows, 100-np.shape(t
fidf_test.todense())[1]))))

print("Our Model_test \n {}".format(model))
print("\n")
print("TF-IDF_test Shape: {}".format(tfidf_test.shape))
print("TF-IDF_test Example: \n{}".format(tfidf_test[2]))
print("\n")
print("Shape of Kernel_test Matrix (an array of shape (X,Y)): {}\
     \n \nA example of Kernel_test Matrix Value (2): \n {}".format(kerne
l_test.shape, kernel_test[2]))
print("\nShape of test_F: {}".format(test_F.shape))
```

```
Our Model_test
 TfidfVectorizer(analyzer='word', binary=False, decode_error='ignore',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_featur
es=100,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=N
one,
                smooth_idf=True, stop_words='english', strip_accents=No
ne,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use_idf=True, vocabulary=None)


TF-IDF_test Shape: (4, 100)
TF-IDF_test Example:
  (0, 86)	0.5976365372261107
  (0, 62)	0.30548373934724476
  (0, 45)	0.5227334676700163
  (0, 39)	0.29272514048374804
  (0, 36)	0.43654549393305875


Shape of Kernel_test Matrix (an array of shape (X,Y)): (4, 64)

A example of Kernel_test Matrix Value (2):
 [0.11856109 0.         0.11015458 0.         0.         0.28959476
 0.         0.07688156 0.0686766  0.13647488 0.         0.14091918
 0.06526972 0.25274064 0.64283057 0.08675784 0.14381919 0.15825049
 0.22585187 0.20542024 0.13323777 0.21479452 0.20063141 0.14565543
 0.2444662  0.10532909 0.24421569 0.1089716  0.1303167  0.13323777
 0.23387677 0.12418547 0.31167683 0.44475285 0.18939264 0.43663059
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]

Shape of test_F: (4, 100)
```

Question 2.3.6.

Once you have the tf-idf representations stored in the matrix test_F (size 4 by 100 features) the value $\kappa(\mathbf{x}, \mathbf{x}_i)$ that you need to get the final classifier $y(\mathbf{x}) = \sum_{i\in\mathcal{D}} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$ and hence the target of the new sentences, you need to compute the cosine similarity of the new "test" tf-idf vectors with the "training" tf-idf vectors which you computed earlier. each of those cosine similarities will give you an entry in $\kappa(\mathbf{x}, \mathbf{x}_i)$ (here $\mathbf{x}$ denotes any of the fixed test sentences). once you have those similarities, compute the target from your α values as $t(\mathbf{x}) = \sum_{i\in\text{train}} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$. print those targets below.

Solution 2.3.6

In [923]:
```
tfidf_test = model.transform(sentences_test)
'''Kernel Test Documents'''
kernel_test = cosine_similarity(test_F,TF_IDF)

'''Non-binary Target Values'''
final_target = np.dot(weights,kernel_test.T)

target_test_final = []
for tar in final_target:
    if tar >= 0.5:
        tar = 1
        target_test_final.append(tar)

    else:
        tar = 0
        target_test_final.append(tar)

print("Shape of Kernel for Test Documents: {}\n".format(kernel_test.shap
e))
print("These are non-binary Target values {} before converting them \nin
to binary numbers, 0's and 1's \n".format(final_target))

print("\tFinal Targets for Test Documents are {}; each value for each Do
cument (sentence).\n".format(target_test_final))
print("\033[1m"+"\t\tIn our case, 0 = Weather/Climate | 1 = Finance/Buis
ness"+"\033[0m")

identity_label = ["Climate", "Finance","Climate", "Finance"]
for tense, label, identity in zip(sentences_test,target_test_final,ident
ity_label):
    print("\nOur Document (Sentence) is: \n{}. \n\tand\
 its target is {} which is {} in our case".format(tense, label,identity
))
```

```
Shape of Kernel for Test Documents: (4, 64)

These are non-binary Target values [-0.06407084  0.84859565  0.00165005
0.6700912 ] before converting them
into binary numbers, 0's and 1's

        Final Targets for Test Documents are [0, 1, 0, 1]; each value f
or each Document (sentence).

                In our case, 0 = Weather/Climate | 1 = Finance/Buisness

Our Document (Sentence) is:
Threat of severe thunderstorms and heavy rain will return to the South
 early week, as another low pressure system tracks across the region..
        and its target is 0 which is Climate in our case

Our Document (Sentence) is:
Investors put a record amount of cash into money market funds in the we
ek.
        and its target is 1 which is Finance in our case

Our Document (Sentence) is:
Heavy snow or rain and thunderstorms with hail are likely over Jammu &
 Kashmir and Ladakh on Friday.
        and its target is 0 which is Climate in our case

Our Document (Sentence) is:
Money market mutual funds and exchange traded funds accounted for $286b
n of inflows for the period.
        and its target is 1 which is Finance in our case
```

Please reach out if anything is unclear

PDF of this file is attached

END OF CODE