Sai Satya Charan . Malladi

charanm@umich.edu

EECS 545

HW-2

29, Sept, 2021

Q1)

(a) $X_1, \dots X_n \overset{iid}{\sim}$ Bernoulli(θ)

$$\text{pmf} = \begin{cases} \theta & x_i = 1 \\ 1-\theta & x_i = 0 \end{cases}$$

Let θ be the success probability

this can be written as $\underline{\theta^{x_i}(1-\theta)^{1-x_i}}$

Since x_i's are iid.

$$MLE = f(x|\theta) = \prod_{i=1}^{n} f(x_i; \theta)$$ where f is pmf/pdf

In case of bernoulli trail.

$$MLE = \prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1-x_i}$$

(maximum Likelihood estimate)

The log-like-lihood is given by.

$$= \log\left(\prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1-x_i}\right) \quad — (*)$$

from properties of logarithm, we can write $(*)$ as.

$$\Rightarrow \sum_{i=1}^{n} \log \theta^{x_i}(1-\theta)^{1-x_i}$$

$$\Rightarrow \sum_{i=1}^{n} \left[x_i \log^{\theta} + (1-x_i) \log^{1-\theta}\right]$$

$$\therefore \hat{\theta} = \arg\max \sum_{i=1}^{n} x_i \log \theta + (1-x_i)\log(1-\theta) \quad - (\#)$$

To maximise for θ take the derivative of expression $(\#)$ set it to 0

$$\frac{d}{d\theta}\left[\sum_{i=1}^{n} x_i \log\theta + (1-x_i)\log(1-\theta)\right] = 0$$

$$\Rightarrow \sum_{i=1}^{n}\left[\frac{d}{d\theta}(x_i)\log\theta + \frac{d}{d\theta}(1-x_i)\log(1-\theta)\right] = 0$$

$$\Rightarrow \sum_{i=1}^{n}\left[\frac{x_i}{\theta} - \frac{(1-x_i)}{(1-\theta)}\right] = 0$$

$$\Rightarrow \sum_{i=1}^{n}\left[\frac{x_i(1-\theta) - (1-x_i)\theta}{\underbrace{\theta(1-\theta)}_{\text{const}}}\right] = 0$$

$$\Rightarrow \sum_{i=1}^{n} x_i - \cancel{\theta x_i} - \theta + \cancel{x_i\theta} = 0$$

$$\Rightarrow \sum_{i=1}^{n}(x_i - \theta) = 0$$

$$\Rightarrow \sum_{i=1}^{n} x_i = \theta n$$

$$\boxed{\theta = \frac{\sum_{i=1}^{n} x_i}{n}}$$

$$\therefore \hat{\ell} = \sum_{i=1}^{n} x_i \log\left(\sum_{i=1}^{n} x_i/n\right) + (1-x_i)\log\left(1-\sum_{i=1}^{n} x_i/n\right)$$

maximum log-likelihood

1 (b) Hessian of log-likelihood.

$$\frac{d}{d\theta}\Bigg[\underbrace{\frac{d}{d\theta}\Big[\sum_{i=1}^{n} x_i \log\theta + (1-x_i)\log(1-\theta)\Big]}_{\text{computed in Q1) a)}}\Bigg]$$

$$\Rightarrow \frac{d}{d\theta}\left[\frac{\sum_{i=1}^{n} x_i}{\theta} - \frac{\sum_{i=1}^{n}(1-x_i)}{(1-\theta)}\right]$$

$$\Rightarrow -\left[\frac{\sum x_i}{\theta^2} + \frac{\sum(1-x_i)}{(1-\theta)^2}\right] \quad - (**)$$

$\Rightarrow$ Substitute $\theta = \hat{\theta} = \dfrac{\sum_{i=1}^{n} x_i}{n}$ in (**)

$$\Rightarrow -\left[\frac{\cancel{\sum x_i}}{\frac{(\sum x_i)^{\cancel{2}}}{n^2}} + \frac{(n-\sum x_i)}{\left[1-\left(\frac{\sum x_i}{n}\right)\right]^2}\right]$$

$$\Rightarrow -\left[\frac{n^2}{\sum_{i=1}^{n} x_i} + \frac{n^2}{(n-\sum_{i=1}^{n} x_i)}\right]$$

$$\Rightarrow -\left[\frac{n^2(n-\sum x_i) + n^2\sum x_i}{\sum_{i=1}^{n} x_i\,(n-\sum_{i=1}^{n} x_i)}\right]$$

$n = +ve$

$\sum x_i = +ve$

$[n-\sum x_i] = +ve$

$$\Rightarrow \frac{-n^3}{\sum_{i=1}^{n} x_i\,(n-\sum_{i=1}^{n} x_i)}$$

the hessian is always $-ve$ for $\theta = \hat{\theta}$

And hence $\hat{\theta} = \sum_{i=1}^{n} x_i / n$ is a maximizer of likelihood.

Q2) a) Intuitively P_{kj} = probability that j feature appears once in class k document.

$$P_{kj} = \frac{n_{kj} + \alpha}{n_k + \alpha d}$$

Apart from Naive Bayes assumption
We are assuming that all the words are equally likely

Ex: The bag of words might contain words like
say, [Porsche , Harley]

We intutively know that if ~~this~~ word Porsche occurs, it is supposed to be car document with high confidence, or if the word Harley occurs it is a Motor Cycle document.

But we are just counting the frequency of times the word occurs, than the weight associated with that words

So, we are assuming that all words have uniform influence in making the decision about classification.

Q2) b)

$$\hat{y} = \underset{k \in \{0,1\}}{\arg\max} \log\left(\hat{\pi}_k \prod_{j=1}^{d} (\hat{P}_{kj})^{x_j}\right)$$

$$= \underset{k \in \{0,1\}}{\arg\max} \log \hat{\pi}_k + \log \prod_{j=1}^{d} (\hat{P}_{kj})^{x_j}$$

$$= \underset{k \in \{0,1\}}{\arg\max} \log \hat{\pi}_k + \sum_{j=1}^{d} \log (\hat{P}_{kj})^{x_j}$$

$$\boxed{\hat{y} = \underset{k \in \{0,1\}}{\arg\max} \log \hat{\pi}_k + \sum_{j=1}^{d} x_j \log(\hat{P}_{kj})}$$

Q2) c) (i) $\log P_{kj}$ for each class $k=0,1$ & each feature $j=1,2\ldots d$ } code attached at the end of pdf.

(ii) $\log \pi_k$ for $k=\{0,1\}$

$\log \pi_0$ _ Class 0 : $\underline{-0.6965}$

$\log \pi_1$ _ Class 1 : $\underline{-0.6898}$

Q2) d) No. of missclassifications :- 100

$$\text{test error} = \frac{\text{No. of missclassifications}}{\text{Total test samples}} = \underline{\underline{12.59\%}}$$

Q2) e) majority class from training data = 1

$$\text{error with predicting majority class} = \frac{\text{No. of missclassifications}}{\text{Total test samples}} = \underline{\underline{49.874\%}}$$

Q3) a) $J(\theta) = -l(\theta) + \lambda\|\theta\|^2 \;\Big|\; l(\theta) = \sum_{i=1}^{n}\left[y_i \log\left(\frac{1}{1+e^{-\theta^T\tilde{x}_i}}\right) + (1-y_i)\log\left(\frac{e^{-\theta^T\tilde{x}_i}}{1+e^{-\theta^T\tilde{x}_i}}\right)\right]$

If ~~sign~~ label convention in logistic regression is changed from $y\in\{0,1\}$ to $y\in\{-1,1\}$ then

Let, $\eta(\hat{y})$ be the ~~posterior~~ conditional probability ~~that~~ for the label given by a logistic curve

$$P(y=1|\theta,x) \;\; \eta(y=1) = \frac{1}{1+e^{-\theta^T\tilde{x}_i}}$$ where $\theta = [b\; w_1\; w_2 \ldots w_d]^T$

$\tilde{x}_i = [1\; x_{i1} \ldots x_{id}]^T$

$$\eta(y=1) = \frac{1}{1+e^{-\theta^T \tilde{x}_i}} \quad -①$$

&

$$\eta(y=-1) = 1 - \eta(y=1)$$

$$\Rightarrow 1 - \frac{1}{1+e^{-\theta^T \tilde{x}_i}}$$

$$\Rightarrow \frac{\cancel{1} + e^{-\theta^T \tilde{x}_i} - \cancel{1}}{1+e^{-\theta^T \tilde{x}_i}}$$

$$\eta(y=-1) = \frac{e^{-\theta^T \tilde{x}_i}}{1+e^{-\theta^T \tilde{x}_i}} \Rightarrow \frac{1}{\frac{1}{e^{-\theta^T \tilde{x}_i}} + \frac{e^{-\theta^T \tilde{x}_i}}{e^{-\theta^T \tilde{x}_i}}} = \frac{1}{e^{\theta^T \tilde{x}_i} + 1}$$

$$\Rightarrow \eta(y=-1) = \frac{1}{1+e^{\theta^T \tilde{x}_i}} \quad -②$$

If we closely observe the structure of ① & ②

we can combine both and write them down as

$$\eta(y_i) = \frac{1}{1+e^{-y_i \theta^T \tilde{x}_i}} \qquad y_i \in \{-1, 1\}$$

&

$$\log\left(\prod_{i=1}^{n} \eta(y_i)\right) = \log \prod_{i=1}^{n} \left(1+e^{-y_i \theta^T \tilde{x}_i}\right)^{-1}$$

$\prod_{i=1}^{n} \eta(y_i) \rightarrow$ likelihood (assuming that x_i's are independent)

$$l(\theta) \Rightarrow -\sum_{i=1}^{n} \log\left(1+e^{-y_i \theta^T \tilde{x}_i}\right)$$

$$\boxed{-l(\theta) = \sum_{i=1}^{n} \log\left(1+e^{-y_i \theta^T \tilde{x}_i}\right)}$$

$\cancel{P(y_i = 1 | \theta, x)} = \frac{1}{\cancel{}}$

$-l(\theta) = \sum_{i=1}^{n} \log(1 + e^{-y_i \theta^T \tilde{x}_i})$ and $J = -l(\theta) + \lambda \|\theta\|^2$

$$\therefore J = \underbrace{\sum_{i=1}^{n} \log(1 + e^{-y_i \theta^T \tilde{x}_i})} + \lambda \|\theta\|^2$$

~~Let~~ $\cancel{\phi(y_i}$

Let $\phi(t) = \log(1 + e^{-t})$

$$\boxed{\therefore J = \sum_{i=1}^{n} \phi(y_i \theta^T \tilde{x}_i) + \lambda \|\theta\|^2}$$

Q3 b) $J(\theta) = \sum_{i=1}^{n} \phi(y_i \theta^T \tilde{x}_i) + \lambda \|\theta\|^2$

Let $t_i = y_i \theta^T \tilde{x}_i$, $\|\theta\|^2 = \underline{\theta^T \theta}$

Scalar

Using chain rule of matrix algebra.

$$\nabla(J(\theta)) = \underbrace{\sum_{i=1}^{n} \frac{\partial \phi(t_i)}{\partial t_i} \times \frac{\partial t_i}{\partial \theta}} + \underbrace{\frac{\partial}{\partial \theta}(\lambda \theta^T \theta)}$$

$\theta = [b \; w_1 \; w_2 \; w_3 \cdots w_d]^T$

$$\frac{\partial}{\partial \theta}(\lambda \theta^T \theta) = \frac{\partial}{\partial \theta} \lambda (b^2 + w_1^2 + w_2^2 + w_3^2 \cdots w_d^2)$$

$$\Rightarrow \lambda \begin{bmatrix} \frac{\partial}{\partial b}(b^2 + w_1^2 \cdots w_d^2) \\ \frac{\partial}{\partial w_1}(b^2 + w_1^2 + \cdots w_d^2) \\ \vdots \\ \frac{\partial}{\partial w_d}(b^2 + w_1^2 \cdots w_d^2) \end{bmatrix} = 2\lambda \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \underline{2\lambda\theta}$$

$\nabla(\lambda \theta^T \theta) = 2\lambda\theta$

$$\nabla(\phi(y_i\theta^T\tilde{x}_i)) \Rightarrow \frac{1}{1+e^{-y_i\theta^T\tilde{x}_i}} \times (-e^{(-y_i\theta^T\tilde{x}_i)}) \times \frac{\partial}{\partial\theta}(y_i\theta^T\tilde{x}_i)$$

$$\Rightarrow \frac{-e^{-y_i\theta^T\tilde{x}_i}}{1+e^{-y_i\theta^T\tilde{x}_i}} \frac{\partial}{\partial\theta}(\underbrace{y_i}_{\text{constant}}\theta^T\tilde{x}_i)$$

$$\Rightarrow \frac{-y_i e^{-y_i\theta^T\tilde{x}_i}}{1+e^{-y_i\theta^T\tilde{x}_i}} \begin{bmatrix} \frac{\partial}{\partial b}[b+w_1x_{i1}+w_2x_{i2}\cdots w_dx_{id}] \\ \frac{\partial}{\partial w_1}[b+w_1x_{i1}\cdots w_dx_{id}] \\ \vdots \\ \frac{\partial}{\partial w_d}[b+w_1x_{i1}+\cdots w_dx_{id}] \end{bmatrix}$$

$$\underline{\nabla(\phi(y_i\theta^T\tilde{x}_i)) = \frac{-y_ie^{-y_i\theta^T\tilde{x}_i}}{1+e^{-y_i\theta^T\tilde{x}_i}}\tilde{x}_i}$$

$$\therefore \nabla J(\theta) = \nabla\phi(y_i\theta^T\tilde{x}_i) + \nabla\lambda\theta^T\theta$$

$$\boxed{\nabla J(\theta) \Rightarrow \frac{-y_ie^{-y_i\theta^T\tilde{x}_i}}{1+e^{-y_i\theta^T\tilde{x}_i}}\tilde{x}_i + 2\lambda\theta}$$

This can be further simplified to

$$\boxed{\nabla J(\theta) = \frac{-y_i}{(1+e^{y_i\theta^T\tilde{x}_i})}\tilde{x}_i + 2\lambda\theta} \quad -(1)$$

Q3c) $\nabla^2(J(\theta)) = J((\nabla J(\theta))^T)$ +

Hessian — Jacobian — gradient of $J(\theta)$

Let $J \to$ be the jacobian

$$J\left(\frac{-y_i \vec{x}_i^T}{1+e^{y_i\theta^T\vec{x}_i}}\right) + J\left(2\lambda\,\theta^T\right)$$

1st term — 2nd term

$\Rightarrow$

Jacobian of 2nd term =
$$\begin{bmatrix} \frac{\partial}{\partial b}2\lambda b & \frac{\partial}{\partial b}2\lambda w_1 & \cdots & \frac{\partial}{\partial b}2\lambda w_d \\ \vdots & & & \\ \frac{\partial}{\partial w_d}2\lambda b & \frac{\partial}{\partial w_d}2\lambda w_1 & \cdots & \frac{\partial}{\partial w_d}2\lambda w_d \end{bmatrix}$$

$$= 2\lambda \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & & & 0 \\ \vdots & & & & & 0 \\ 0 & 0 & 0 & \cdots & & 1 \end{bmatrix}_{(d+1 \times d+1)}$$

$$J(2\lambda\theta^T) = 2\lambda I_{d+1 \times d+1}$$

Jacobian of 1st term.

$$= \begin{bmatrix} \frac{\partial}{\partial b}\frac{-y_i}{(1+e^{y_i\theta^T\vec{x}_i})} & \frac{\partial}{\partial b}\frac{-y_i}{1+e^{y_i\theta^T\vec{x}_i}}x_{i1} & \cdots & \frac{\partial}{\partial b}\frac{-y_i}{1+e^{y_i\theta^T\vec{x}_i}}x_{id} \\ \vdots & \vdots & & \vdots \\ \frac{\partial}{\partial w_d}\frac{-y_i}{1+e^{y_i\theta^T\vec{x}_i}} & \cdots & \cdots & \frac{\partial}{\partial w_d}\frac{-y_i}{1+e^{y_i\theta^T\vec{x}_i}}x_{id} \end{bmatrix}$$

$$= \frac{+y_i^2 e^{y_i\theta^T\vec{x}_i}}{(1+e^{y_i\theta^T\vec{x}_i})^2} \begin{bmatrix} 1\cdot 1 & 1\cdot x_{i1} & 1\cdot x_{i2} & \cdots & 1\cdot x_{id} \\ x_{i1}\cdot 1 & x_{i1}\cdot x_{i1} & x_{i1}\cdot x_{i2} & \cdots & x_{i1}\, x_{id} \\ \vdots & & & & \\ x_{id}\cdot 1 & x_{id}\cdot x_{i1} & & \cdots & x_{id}\cdot x_{id} \end{bmatrix}$$

Jacobian of ~~1st~~ term $\Rightarrow$ $\dfrac{y_i^2 e^{y_i\theta^T\tilde{x}_i}}{(1+e^{y_i\theta^T\tilde{x}_i})^2} \tilde{x}_i \tilde{x}_i^T$

$\therefore$

$$\boxed{\nabla^2(J(\theta)) = \frac{y_i^2 e^{y_i\theta^T\tilde{x}_i}}{(1+e^{y_i\theta^T\tilde{x}_i})^2} \tilde{x}_i \tilde{x}_i^T + 2\lambda I_{d+1\times d+1}}$$

Q3 d)

$$\nabla^2(J(\theta)) = \underbrace{\frac{y_i^2 e^{y_i\theta^T\tilde{x}_i}}{(1+e^{y_i\theta^T\tilde{x}_i})^2}}_{\text{is a +ve const}} \tilde{x}_i \tilde{x}_i^T + 2\lambda I_{d+1\times(d+1)}$$

replace it with Z

$$\Rightarrow Z(\tilde{x}_i \tilde{x}_i^T) + 2\lambda I_{d+1\times d+1}$$

$\rightarrow$ $x_i \tilde{x}_i^T$ is PSD

Proof

Let u be a random vector $u \in \mathbb{R}^{d+1}$

$$u^T x_i x_i^T u$$

$$\Rightarrow (x_i^T u)^T (x_i^T u) = \|x_i^T u\|^2 \geq 0$$

$\rightarrow$ To be a $\nabla^2(J(\theta))$ convex function we need $2\lambda I_{d+1\times d+1}$ to be PSD

$$\Rightarrow u^T(2\lambda I_{d+1\times d+1})u \geq 0 \Rightarrow 2\lambda \|u\|^2 \geq 0$$

$$\Rightarrow \lambda \geq 0$$

For strict convexity we need $2\lambda I_{d+1\times d+1}$ to be PD matrix.

$$\Rightarrow u^T(2\lambda I_{d+1\times d+1})u > 0 \Rightarrow 2\lambda \|u\|^2 > 0$$

$$\Rightarrow \lambda > 0$$

Q) 4b) I have choosen confidence to be the probability with which the ~~classifier~~ logistic regression misclassifies.

(ie) For example, let $\eta(y=1) = \dfrac{1}{1+e^{-\theta^T \vec{x}_i}}$

$\eta(y=-1) = \dfrac{1}{1+e^{-\theta^T \vec{x}_i}}$

For a data input

If the true label is 1

and logistic regression predicts (-1)

Then my confidence = $\eta(y=-1)$ for that data point

= amount of probability that the classifier ~~to~~ associates with the misclassification.

Q) 4 (a)

Test error = 3.4% → $\left(\dfrac{\text{Missclassified}}{\text{Total}}\right)$

Iterations to converge = 9

Value of Objective function = 456.64

Problem 4b

Figure 1: Top 20 misclassified images

HW2 Problem 2

In [1]:
```
# import relevant packages
import numpy as np
```

In [2]:
```
# load train and test data
train_x = np.load('hw2p2_data/hw2p2_train_x.npy')
train_y = np.load('hw2p2_data/hw2p2_train_y.npy')
test_x = np.load('hw2p2_data/hw2p2_test_x.npy')
test_y = np.load('hw2p2_data/hw2p2_test_y.npy')
```

In [3]:
```
# dimensions
n_train = train_x.shape[0] # No of train documents
n_test = test_x.shape[0]   # No of test documents
d = train_x.shape[1]       # dimension of feature vector

print('n_train: {} \nn_test: {} \nd: {}\n'.format(n_train, n_test, d))
```

```
n_train: 1192
n_test: 794
d: 1000
```

In [4]:
```
### Problem 2c - computation of log(p_kj) and log(pi_k) (k = 0 or 1)

## (i) esitmation of log(p_kj)
# computing n_k and n_kj
n_k = np.array([np.sum(train_x[train_y==k,:]) for k in range(2)])
n_kj = np.array([np.sum(train_x[train_y==k,:],axis = 0) for k in range(2)])

# computing p_kj
alpha = 1 # Laplace smoothing constant
p_kj = np.array([(n_kj[k,:] + alpha)/(n_k[k] + alpha*d) for k in range(2)])

# log(p_kj)
log_p_kj = np.log(p_kj)


## (ii) estimation of log(pi_k)
pi_k = np.array([np.sum(train_y==k)/n_train for k in range(2)])
log_pi_k = np.log(pi_k)

print('log-prior \nclass 0: {} \nclass 1: {}\n'.format(log_pi_k[0],log_pi_k[1]))
```

```
log-prior
class 0: -0.6965085282626502
class 1: -0.6897970936746632
```

In [6]:
```
### problem 2d - predction of classes for test data

predicted_y = np.zeros(n_test)   # predefine

for i in range(n_test):
    # computation of posterior for classes (k = 0,1) for each document (i = 0,1,2...)
    eta = np.array([log_pi_k[k] + np.sum(test_x[i,:]*log_p_kj[k,:])  for k in range(2)])

    # assign class that maximizes posterior
    predicted_y[i] = np.argmax(eta)

# error associated with naive-bayes classification
n_misclassified = np.sum(test_y != predicted_y)
test_error = n_misclassified/n_test*100

print('number of misclassified documents: {} \nerror in Naive Bayes classification: {} %\n'.format(n_misclassified,test_error))
```

```
number of misclassified documents: 100
error in Naive Bayes classification: 12.594458438287154 %
```

In [117…
```
### problem 2e - sanity check

# select the maximum ouccuring class in training data
majority_class_train = np.argmax([np.sum(train_y==0), np.sum(train_y==1)])

# error associated with dominant class classification
majority_class_error = np.sum(test_y != majority_class_train)/n_test*100

print('error in majority class classification: {} %\n'.format(majority_class_error))
```

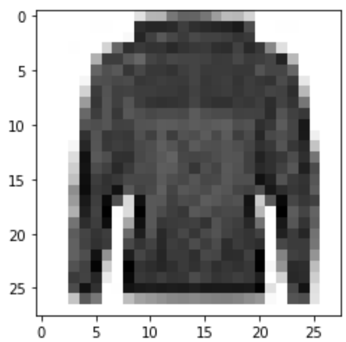
```
error in majority class classification: 49.87405541561713 %
```

HW2 Problem 4

In [2]:
```
# import relevant packages
import numpy as np
import matplotlib.pyplot as plt
import math
```

In [3]:
```
# load the data
x = np.load('hw2p4_data/fashion_mnist_images.npy')
y = np.load('hw2p4_data/fashion_mnist_labels.npy')
d, n = x.shape
```

In [4]:
```
# visualize
i = 0 #Index of the image to be visualized
plt.imshow(np.reshape(x[:,i], (int(np.sqrt(d)),int(np.sqrt(d)))), cmap="Greys")
plt.show()
```



In [5]:
```
# split the data into test and train
train_x = x[:,:5000]
train_y = y[0,:5000]
test_x = x[:,5000:]
test_y = y[0,5000:]
```

In [6]:
```
class logistic_regression:
    ## constructor
    def __init__(self,x,y,d,n):
        self.d = d        # number of features
        self.n = n        # number of examples
        self.x_tilde = np.vstack((np.ones(n),x)) # x augmented with a cloumn of ones
        self.y = y  # set of labels


    ## methods
    # objective function - log likelihood
    def objective(self,theta,reg_const):

        # for ease of coding unpack self
        n = self.n
        y = self.y
        x_tilde = self.x_tilde

        phi = np.sum([np.log(1+np.exp(-y[i]*np.dot(theta,x_tilde[:,i]))) for i in range(n)])
        return phi + reg_const*np.dot(theta,theta)


    # gradient - log likelihood
    def gradient(self,theta,reg_const):

        # for ease of coding unpack self
        n = self.n
        y = self.y
        x_tilde = self.x_tilde

        gradient_log_likelihood = 0
        for i in range(n):
            gradient_log_likelihood += -1/(1+np.exp(y[i]*np.dot(theta,x_tilde[:,i])))*y[i]*x_tilde[:,i]
        return gradient_log_likelihood + 2*reg_const*theta


    # hessian - log likelihood
    def hessian(self,theta,reg_const):

        # for ease of coding unpack self
        n = self.n
        y = self.y
        x_tilde = self.x_tilde

        hessian_log_likelihood = 0
        for i in range(self.n):
```

```
            hessian_log_likelihood += np.exp(y[i]*np.dot(theta,x_tilde[:,i]))/(1+np.exp(y[i]*np.dot(theta,x_tilde[:,i])))**2*y[i]**2*np
        return hessian_log_likelihood + 2*reg_const*np.eye(self.d+1)

    # newtons_method
    def newtons_method(self, theta_0,reg_const,tolerance):
        theta_t = theta_0 # initialize theta at step t to theta_0
        iteration = 0
        step_change = math.inf
        while(step_change > tolerance):

            # at step t
            gradient_t = self.gradient(theta_t,reg_const)
            hessian_t = self.hessian(theta_t,reg_const)
            objective_t = self.objective(theta_t,reg_const)

            # at step t+1
            theta_t1 = theta_t - np.matmul(np.linalg.inv(hessian_t),gradient_t)
            objective_t1 = self.objective(theta_t1,reg_const)

            step_change = np.absolute(objective_t1 - objective_t)/objective_t # update step_change
            theta_t = theta_t1                          # update theta
            iteration += 1                              # update iteration

        return theta_t, iteration, objective_t
```

In [7]:

```
### problem 4a - error, objective and interations

n_train = train_x.shape[1] # number of trainning examples

# declare a class object
prob4 = logistic_regression(train_x,train_y,d,n_train)

# inputs to the newtons method
theta_0 = np.zeros(d+1) # inital guess for optimization
reg_const = 1           # regularization constant
tolerance = 1e-6        # optimization tolerance

# run the optimzation (theta_hat - solution of the optimization)
theta_hat, iteration, objective = prob4.newtons_method(theta_0,reg_const,tolerance)
print('logistic regression results\niterations to converge: {} \nvalue of the objective function: {}'.format(iteration,objective))
```

```
logistic regression results
iterations to converge: 9
value of the objective function: 456.63896507162525
```

In [8]:

```
# prediction for test data
n_test = test_x.shape[1]                            # number of test examples
predicted_y = np.zeros(n_test)                      # predefine
test_x_tilde = np.vstack((np.ones(n_test),test_x)) # augment test_x
classes = np.array([-1,1])                          # classes (iteration purpose - for loop)
eta = np.zeros((2,n_test))

for i in range(n_test):
    # computation of posterior probability using sigmod function
    eta[:,i] = np.array([1/(1+np.exp(-j*np.dot(theta_hat,test_x_tilde[:,i]))) for j in classes])

    # assign class that maximizes posterior
    # this is equivalent to saying eta > 0.5 for any particular class
    predicted_y[i] = classes[np.argmax(eta[:,i])]

# error assoiated with prediction
test_error = np.sum(test_y != predicted_y)/n_test*100
print('error in classification: {} %\n'.format(test_error))
```

```
error in classification: 3.4000000000000004 %
```

In [11]:

```
### problem 4b - generate 20 images

# misclassified
misclassified = np.argwhere(test_y != predicted_y)

# confidence associated with mis-classifications
eta_misclassified = np.zeros(np.size(misclassified))
for i in range(np.size(misclassified)):
    eta_misclassified[i] = np.max(eta[:,misclassified[i]])

# arg sort to find the top 20 misclassifications in the test set
top20_misclassified = np.flip(misclassified[np.argsort(eta_misclassified)[-20:]])



# plot
fig = plt.figure(figsize=(12, 12))
rows = 4
columns = 5
for k in range(20):
    i = top20_misclassified[k]
    fig.add_subplot(rows, columns, k+1)
    plt.imshow(np.reshape(test_x[:,i], (int(np.sqrt(d)),int(np.sqrt(d)))), cmap="Greys")
```

```
    plt.axis('off')
    plt.title('true label : {} \n predicted label : {}'.format(test_y[i],predicted_y[i]),fontsize=10,color = 'b')
plt.show()
```