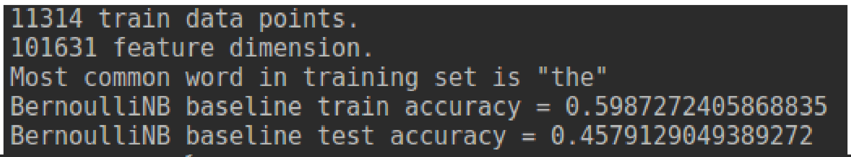
1. 20 Newsgroups predictions

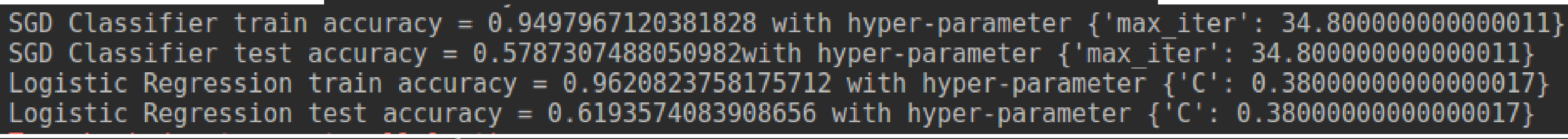
1.1 Data and algorithms:

Data: bag-of-words
used algorithms:
stochastic gradient descent classifier
logistic regression classifier
multinomial Naive-Bayes

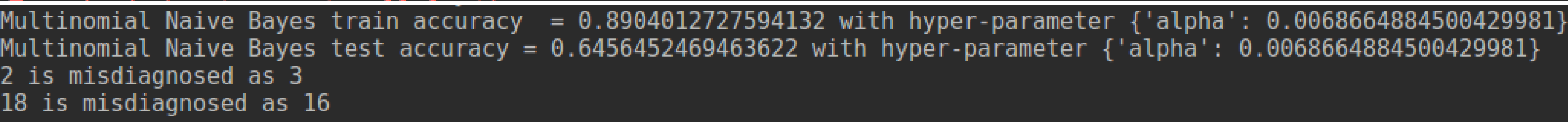
1.2Result:



```
11314 train data points.
101631 feature dimension.
Most common word in training set is "the"
BernoulliNB baseline train accuracy = 0.5987272405868835
BernoulliNB baseline test accuracy = 0.4579129049389272
```



```
SGD Classifier train accuracy = 0.9497967120381828 with hyper-parameter {'max_iter': 34.800000000000011}
SGD Classifier test accuracy = 0.5787307488050982with hyper-parameter {'max_iter': 34.800000000000011}
Logistic Regression train accuracy = 0.9620823758175712 with hyper-parameter {'C': 0.38000000000000017}
Logistic Regression test accuracy = 0.6193574083908656 with hyper-parameter {'C': 0.38000000000000017}
```



```
Multinomial Naive Bayes train accuracy  = 0.8904012727594132 with hyper-parameter {'alpha': 0.0068664884500429981}
Multinomial Naive Bayes test accuracy = 0.6456452469463622 with hyper-parameter {'alpha': 0.0068664884500429981}
2 is misdiagnosed as 3
18 is misdiagnosed as 16
```

Among three algorithms, Multinomial Naive-Bayes has best performance.

1.3hyper-parameter:

I used Randomized Search Cross validation to get the hyper-parameter, with kfold is 5.
max_iter in SGD classifier is 34.8.
C in logistic regression is 0.38
alpha in multinomial Naive-Bayes is 0.0068

1.4 the reason choosing these three algorithms:

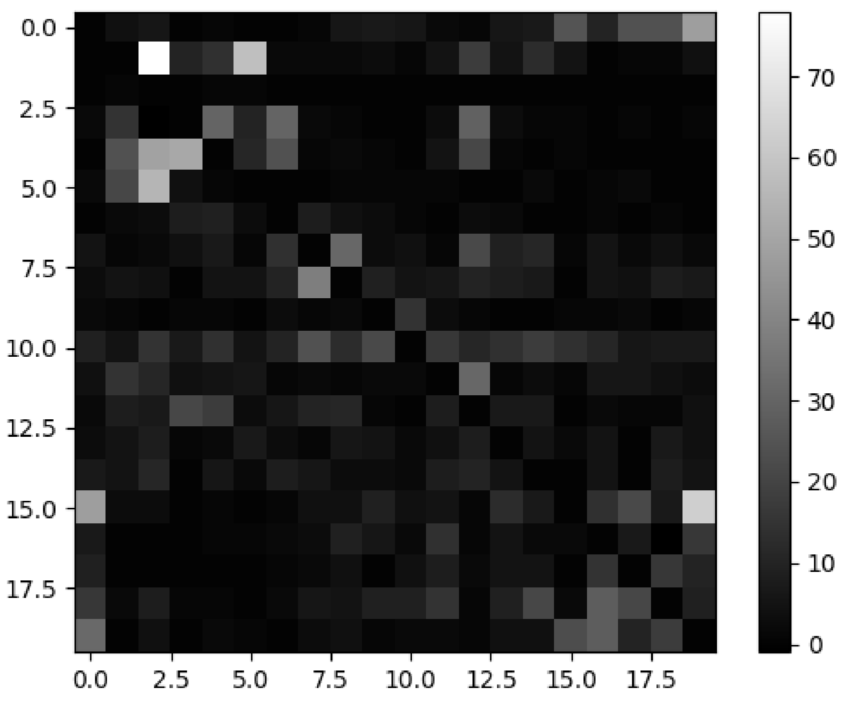
As for SGD classifier, it has advantage of efficiency and it can solve big set of data.

As for logistic regression classifier, it is easy to implement. Because the we have 20 features, logistic regression has a better performance than SVM.

As for Multinomial Naive-Bayes, the 20 features are diversification and discrete, so Multinomial Naive-Bayes has better performance than Bernoulli Naive-Bayes.

The result is as my thought all the three have better result than Bernoulli Naive-Bayes.

1.5 Because Multinomial Naive-Bayes has the best performance, I draw the confusion matrix of Multinomial Naive-Bayes and set the diagonal to 0 and get the two largest set.

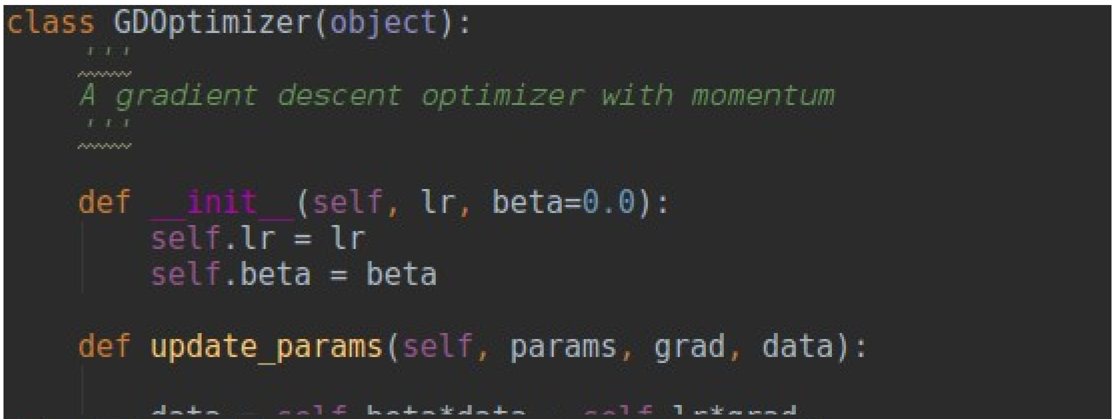


The two classes confused most is 2 is misdiagnosed as 3 and 18 is misdiagnosed as 16.

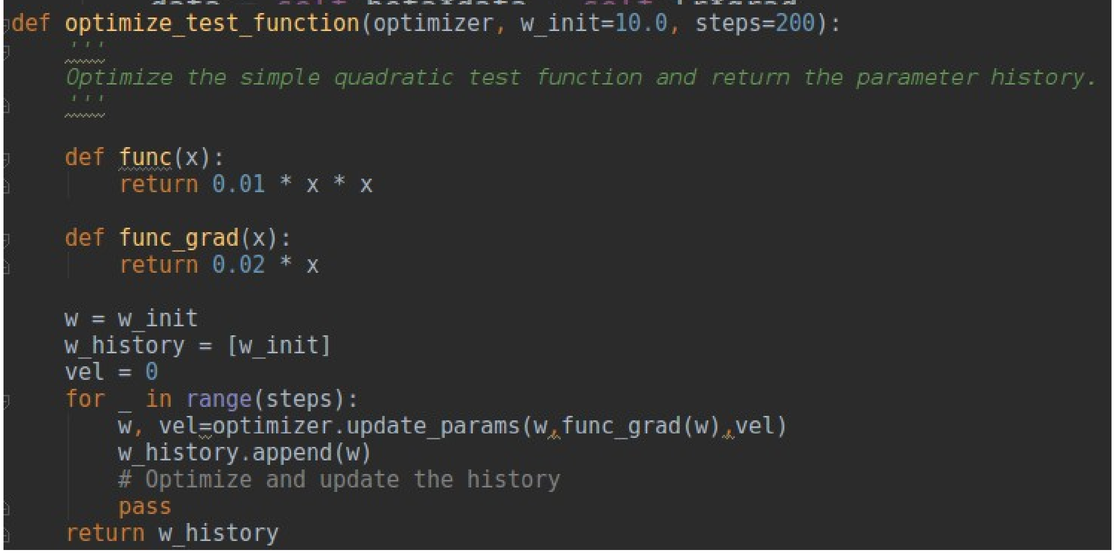
2 Training SVM with SGD:

2.1 SGD with Momentum

2.1.1



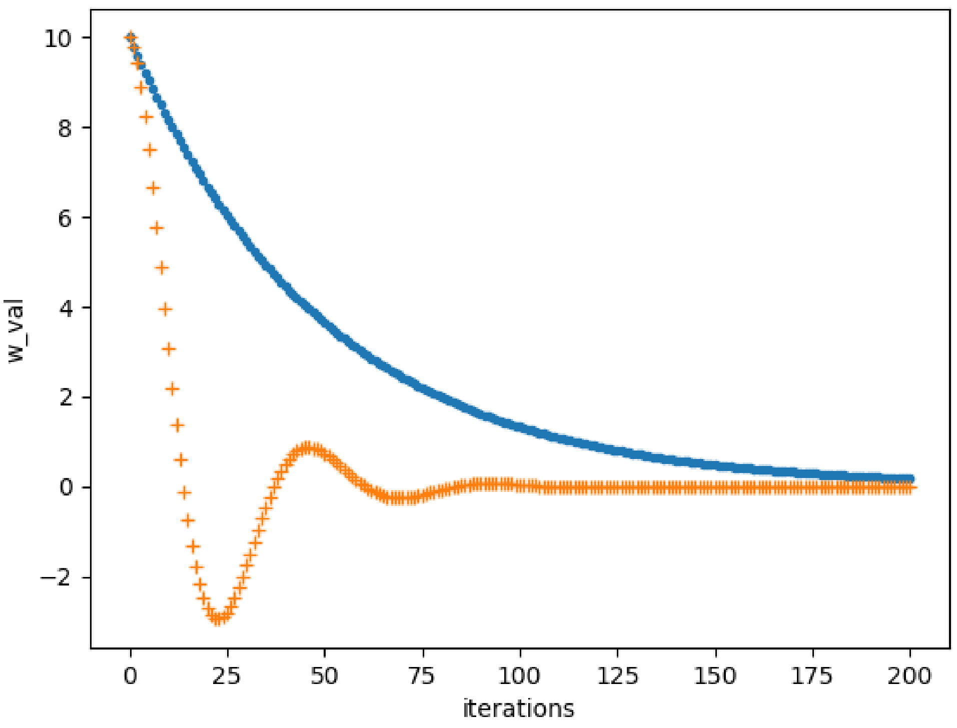
```
class GDOptimizer(object):
    '''
    A gradient descent optimizer with momentum
    '''

    def __init__(self, lr, beta=0.0):
        self.lr = lr
        self.beta = beta

    def update_params(self, params, grad, data):

        data = self.beta*data - self.lr*grad
```



```
def optimize_test_function(optimizer, w_init=10.0, steps=200):
    '''
    Optimize the simple quadratic test function and return the parameter history.
    '''

    def func(x):
        return 0.01 * x * x

    def func_grad(x):
        return 0.02 * x

    w = w_init
    w_history = [w_init]
    vel = 0
    for _ in range(steps):
        w, vel=optimizer.update_params(w,func_grad(w),vel)
        w_history.append(w)
        # Optimize and update the history
        pass
    return w_history
```

2.1.2

The blue one is beta =0, and the orange one is beta=0.9



10
8
6
4
2
0
−2
w_val
0
25
50
75
100
125
150
175
200
iterations

2.2 Training SVM:

The hinge-loss :

If $y*(X*w+b)<1$, $loss=1-y*(X*w+b)$

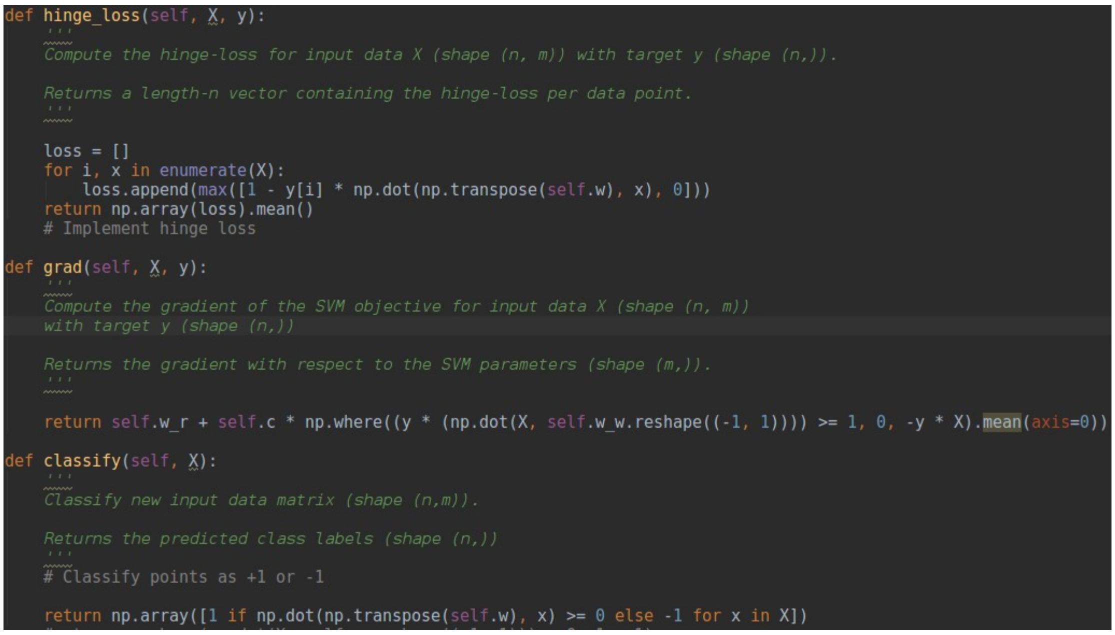
Else $loss=0$

The gradient of SVM

$$\frac{\partial J(w,b:\alpha)}{\partial w}=w-c*y*X \quad \text{If} \quad y*(X*w+b)<1$$

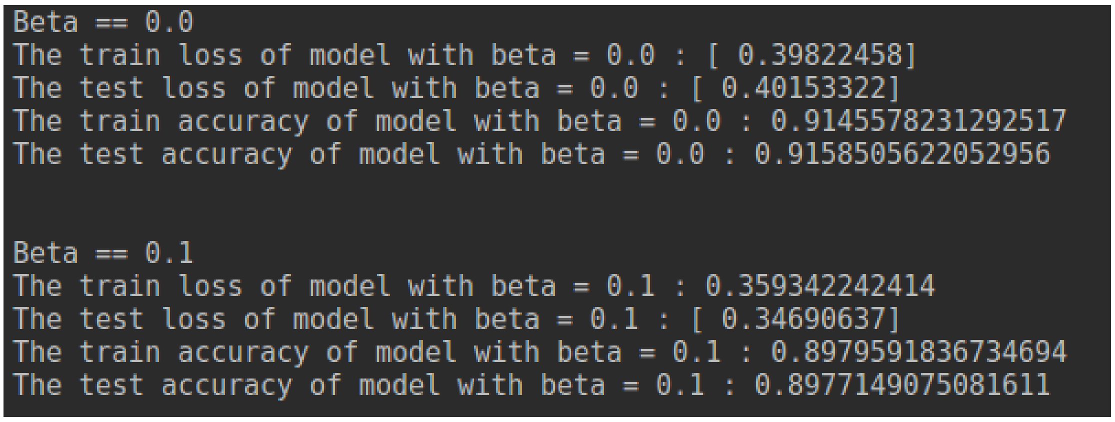
else $\frac{\partial J(w,b:\alpha)}{\partial w}=w$

Classify if $x*w\geq 0, y=1; x*w<0, y=-1;$



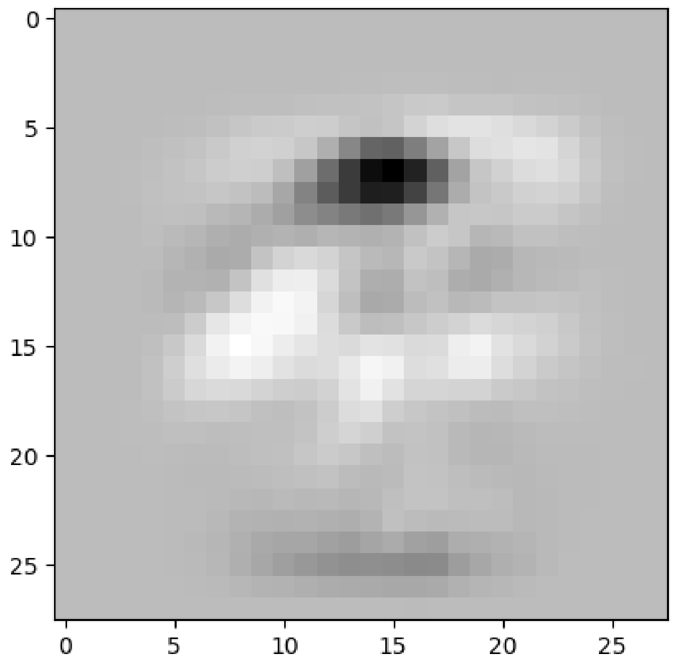
```python
def hinge_loss(self, X, y):
    '''
    Compute the hinge-loss for input data X (shape (n, m)) with target y (shape (n,)).

    Returns a length-n vector containing the hinge-loss per data point.
    '''

    loss = []
    for i, x in enumerate(X):
        loss.append(max([1 - y[i] * np.dot(np.transpose(self.w), x), 0]))
    return np.array(loss).mean()
    # Implement hinge loss

def grad(self, X, y):
    '''
    Compute the gradient of the SVM objective for input data X (shape (n, m))
    with target y (shape (n,))

    Returns the gradient with respect to the SVM parameters (shape (m,)).
    '''

    return self.w_r + self.c * np.where((y * (np.dot(X, self.w_w.reshape((-1, 1))))) >= 1, 0, -y * X).mean(axis=0))

def classify(self, X):
    '''
    Classify new input data matrix (shape (n,m)).

    Returns the predicted class labels (shape (n,))
    '''
    # Classify points as +1 or -1

    return np.array([1 if np.dot(np.transpose(self.w), x) >= 0 else -1 for x in X])
```

2.3 Apply on 4-vs-9 digits on MNIST:

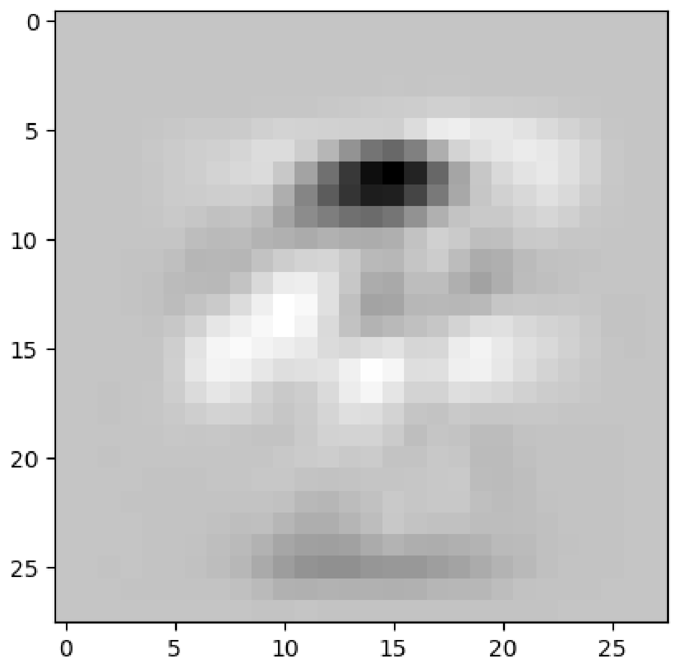


```
Beta == 0.0
The train loss of model with beta = 0.0 : [ 0.39822458]
The test loss of model with beta = 0.0 : [ 0.40153322]
The train accuracy of model with beta = 0.0 : 0.9145578231292517
The test accuracy of model with beta = 0.0 : 0.9158505622052956


Beta == 0.1
The train loss of model with beta = 0.1 : 0.359342242414
The test loss of model with beta = 0.1 : [ 0.34690637]
The train accuracy of model with beta = 0.1 : 0.8979591836734694
The test accuracy of model with beta = 0.1 : 0.8977149075081611
```

As for beta is 0:
The training loss is 0.398.
The test loss is 0.40.
The classification accuracy on the training set is 0.92.
The classification accuracy on the test set is 0.92
the image of w:



As for beta is 1:
The training loss is 0.359.
The test loss is 0.247.
The classification accuracy on the training set is 0.90.
The classification accuracy on the test set is 0.90
the image of w:



3 Kernels:

3.1

necessary : as for a random symmetric matrix K, decomposing the matrix into $K = Q^T \Lambda Q$ and $x^T Kx = (Qx)^T \Lambda Qx$.

Because Q is a orthogonal matrix, Qx is a linear transform of x, if $x^T Kx \geq 0$ we have the eigenvalues diagonal matrix $\Lambda \geq 0$. so K is positive semidefinite.

Sufficiency: if $x^T Kx < 0$, these must exist a eigenvalue in Λ is less than zero.

3.2

1. $k(x,y) = \alpha$ and $\alpha > 0$, we have matrix K has all element α ,

So we have as for any x, $x^T Kx \geq 0$. thus, $k(x,y) = \alpha$ is a kernel function.

2. $k(x,y) = f(x) * f(y)$

$k(x,y)$ Is symmetric matrix.

So, we have:

$$x^T Kx = \sum_{i,j=1}^{l} x_i x_j K_{ij} = \sum_{i,j=1}^{l} x_i x_j f(x_i) f(x_j) = (\sum_{i=1}^{l} x_i f(x_i))^2 \geq 0$$

$k(x,y)$ Is positive semidefinite.

Thus, $k(x,y)=f(x)*f(y)$ is a kernel faction.

3 because $a>0, b>0$, $x^T \alpha K x = \alpha x^T K x \geq 0, x^T b K x = b x^T K x \geq 0$

So, $a*k_1(x,y), b*k_2(x,y)$ are kernel faction.

And $a*k_1(x,y)+b*k_2(x,y)=\alpha x^T K x + x^T b K x \geq 0$,

Thus, it is a kernel.

4. $K(x,y)=K_1(x,y)/\sqrt{k_1(x,x)}\sqrt{k_1(y,y)}=\dfrac{\Phi(x)*\Phi(y)}{\|\Phi(y)\|\|\Phi(x)\|}$

If $K_1(x,y)$ is a kernel, then $aK_1(x,y), a\geq 0$ is also a kernel.

$a=\dfrac{1}{\sqrt{k_1(x,x)}\sqrt{k_1(y,y)}}$. Thus, $K(x,y)$ is a kernel faction.