1.1

$$J(w) = \lambda \|w\|^2 - \sum_{i=1}^{n} (y_i \ln s_i + (1-y_i) \ln(1-s_i))$$

where $s_i = s(x_i \cdot w)$, $s(r) = 1/(1+e^{-r})$

$$\nabla_w J(w) = 2\lambda w - \sum_{i=1}^{n} \left(\frac{y_i}{s_i} \nabla s_i + \frac{1-y_i}{1-s_i} \nabla s_i\right)$$

$$s'(r) = \frac{d}{dr} \frac{1}{1+e^{-r}} = \frac{e^{-r}}{(1+e^{-r})^2} = s(r)(1-s(r))$$

$$\nabla_w J(w) = 2\lambda w - \sum_{i=1}^{n} \left(\frac{y_i}{s_i} - \frac{1-y_i}{1-s_i}\right) s_i (1-s_i) x_i$$

$$= -X^T (y - s(Xw)) + 2\lambda w$$

1.2

$$\nabla_w J(w) = -X^T(y - s(X \cdot w)) + 2\lambda w$$

$$\nabla_w^2 J(w) = 2\lambda I + X^T \Omega X$$

where Ω is the diagonal matrix of $s(i)$

13

$e \leftarrow$ solution to normal equations $(X^T S X + 2\lambda I)e$
$= X^T(y - s(Xw)) - 2\lambda w$

$w \leftarrow w + e$

1.4

$s^{(0)} = [0.95257413, 0.73105858, 0.73105858, 0.26894142]$
$w^{(1)} = [-1.25039512, 1.45276462, -1.53745277]$
$s^{(1)} = [0.9437917, 0.82784718, 0.47884061, 0.20831978]$
$w^{(2)} = [-0.87605518, 1.60823428, -2.16803196]$

2.1

$$J(w) = |Xw - y|^2 + \lambda \|w\|_1$$
$$= w^T X^T X w - 2y^T X w + y^T y + \lambda \|w\|_1$$

Since $X^T X = nI$, we get:

$$J(w) = w^T w + y^T y - 2y^T X w + \lambda \|w\|_1$$

Let $g(y) = y^T y$,

Given $J(w) = g(y) + \sum_{i=1}^{d} f(X_{*i}, w_i, y, \lambda)$

we get:

$$f(X_{*i}, w_i, y, \lambda) = J(w) - g(y)$$
$$= w^T w - 2y^T X w + \lambda \|w\|_1$$

Thus

$$J(w) = g(y) + f(X_{*i}, w_i, y, \lambda)$$

where $g(y) = y^T y$

$$f(X_{*i}, w_i, y, \lambda) = w^T w - 2y^T X w + \lambda \|w\|_1$$

$f'(X_{*i}, w_i, y, \lambda)$

$= 2w_i - 2y^T X_* + \lambda = 0$

$w_i^* = \frac{2y^T X_* - \lambda}{2}$ if $w_i^* > 0$

$w_i^* = \frac{2y^T X_* + \lambda}{2}$ if $w_i^* < 0$

2.4

when $w_i^* = 0$,

$$|2y^T x_*| \leq \lambda$$

2.5 $J(w) = |Xw - y|^2 + \lambda \|w\|_2$

$= w^T w + y^T y - 2y^T X w + \lambda w^2$

Thing

$J(w) = g(y) + f(X_{*i}, w_i, y, \lambda)$

where $g(y) = y^T y$

$f(X_{*i}, w_i, y, \lambda) = w^T w + \lambda w^2 - 2y^T X w$

$f'(X_{*i}, w_i, y, \lambda) = 2w_i + 2\lambda w_i - 2y^T X = 0$

$w_i^* = \frac{2y^T X}{2+2\lambda} = \frac{y^T X}{1+\lambda} = 0$

therefore $y^T X = 0$

3.1 Date

$$\frac{\partial |w|^4}{\partial w_i} = 4\left(\sum w_i^2\right) w_i$$

$$\nabla |w|^4 = 4 w^T w w$$

$$\nabla |Xw - y|^4 = 4 (Xw - y)^T (Xw - y) X^T (Xw - y)$$

3.2

Let $J(w) = |Xw-y|^4 + \lambda |w|^2$

$$\nabla J(w) = 4(Xw-y)^T (Xw-y) X^T (Xw-y) + 2\lambda w$$

Set $\nabla J(w) = 0$, we get

$$4(Xw^*-y)^T (Xw^*-y) X^T (Xw^*-y) + 2\lambda w^* = 0$$

$$w^* = -\frac{2}{\lambda} (Xw^*-y)^T (Xw^*-y)(Xw^*-y)^T X$$

So w^* can be written as $w^* = \sum_{i=1}^{n} a_i x_i$

where $a = -\frac{2}{\lambda} (Xw^*-y)^T (Xw^*-y)(Xw^*-y)^T$

Let $J(w) = \frac{1}{n} \sum_{i=1}^{n} L(w^T x_i, y_i) + \lambda \|w\|^2$

$$\nabla J(w) = \frac{1}{n} \sum_{i=1}^{n} L'(w^T x_i, y_i) x_i + 2\lambda w$$

Set $\nabla J(w) = 0$

$$\frac{1}{n} \sum_{i=1}^{n} L'(w^T x_i, y_i) x_i + 2\lambda w^* = 0$$

$$w^* = -\frac{1}{2n\lambda} L'(w^T x_i, y_i) x_i$$

$$= \sum_{i=1}^{n} \alpha_i x_i$$

Yes. optimal solution has the form $w^* = \sum_{i=1}^{n} \alpha_i x_i$
Because whether cost function is convex or not doesn't affect $\nabla J(w) = 0$.

4.9 Derive batch gradient descent update function for logistic regression with l2 regularization.

Find w that minimizes

$$J = -\sum_{i=1}^{n} \left(y_i \ln s(x_i \cdot w) + (1-y_i) \ln (1 - s(x_i \cdot w)) + \lambda \|w\|^2 \right)$$

let $s_i = s(x_i \cdot w)$. we get

$$J = -\sum_{i=1}^{n} \left(y_i \ln s_i + (1-y_i) \ln (1-s_i) + \lambda \|w\|^2 \right)$$

Take the gradient of J with respect to w

$$\nabla_w J = -\sum_{i=1}^{n} \left(\frac{y_i}{s_i} \nabla s_i + \frac{1-y_i}{1-s_i} \nabla s_i + 2\lambda w \right)$$

Since,

$$s'(r) = \frac{d}{dr} \frac{1}{1+e^{-r}} = \frac{e^{-r}}{(1+e^{-r})^2} = s(r)(1-s(r))$$

We get

$$\nabla_w J = -\sum_{i=1}^{n} \left(\frac{y_i}{s_i} - \frac{1-y_i}{1-s_i} \right) s_i (1-s_i) x_i + 2\lambda w$$

$$= -\sum_{i=1}^{n} \left(y_i (1-s_i) x_i - (1-y_i) s_i x_i + 2\lambda w \right)$$

$$= -\sum_{i=1}^{n} \left((y_i - s_i) x_i + 2\lambda w \right)$$

$$= -X^T (y - s(Xw)) + 2\lambda w$$

So the batch gradient descent update equation is:

$$w^{t+1} = w^t - \varepsilon \nabla_w J = w^t - \varepsilon \left[-X^T (y - s(Xw)) + 2\lambda w \right]$$

4. b Stochastic gradient descent update equation for logistic regression with l2 regularization

$$w^{t+1} = w^t - \varepsilon \nabla_{w,i} J = w^t - \varepsilon [-X_i^T (y_i - s(X_i w)) + 2\lambda w]$$

4.2

BGD converges slower but iterates much less times. SGD is faster but iterates much more times.

4.3

Comparing the two graphs, this strategy is not better than having a constant ϵ, since the convergence speed is slower than before.

4.4

Kaggle team name: JoKer
Accuracy: 0.99328
Description: BGD is better for this dataset, I chose 0.01 for learning rate and $\lambda = 1$ after validation.

5

The reason is that the feature is not linearly seperable.
To improve this, he can normalize the time stamp.

In [13]:

```
# Load data
import sys
if sys.version_info[0] < 3:
    raise Exception("Python 3 not detected.")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io
import random

wine = io.loadmat("data.mat")
print("\nloaded %s data!" % wine)
fields = "X", "y", "X_test"
for field in fields:
    print(field, wine[field].shape)

wine_X = wine['X']
wine_y = wine['y']
wine_test = wine['X_test']
```

```
loaded {'__header__': b'MATLAB 5.0 MAT-file Platform: posix, Created on: S
un Feb 26 19:25:34 2017', '__version__': '1.0', '__globals__': [], 'y': ar
ray([[0.],
       [1.],
       [0.],
       ...,
       [0.],
       [0.],
       [0.]]), 'X': array([[ 5.8  ,  0.555,  0.26 , ...,  0.46 ,  9.1  ,
0.5  ],
       [ 6.3  ,  0.36 ,  0.19 , ...,  0.52 , 12.7  ,  0.6  ],
       [ 6.4  ,  0.21 ,  0.5  , ...,  0.43 ,  8.8  ,  0.5  ],
       ...,
       [ 7.3  ,  0.2  ,  0.37 , ...,  0.49 , 10.9  ,  0.6  ],
       [ 8.5  ,  0.25 ,  0.27 , ...,  0.33 , 12.   ,  0.6  ],
       [ 6.6  ,  0.36 ,  0.21 , ...,  0.41 ,  9.9  ,  0.6  ]]), 'descripti
on': array(['fixed acidity       ', 'volatile acidity    ',
       'citric acid         ', 'residual sugar      ',
       'chlorides           ', 'free sulfur dioxide ',
       'total sulfur dioxide', 'density             ',
       'pH                  ', 'sulphates           ',
       'alcohol             ', 'quality             '], dtype='<U20'), 'X_
test': array([[ 8.2 ,  0.22,  0.49, ...,  0.33, 10.6 ,  0.6 ],
       [ 5.4 ,  0.74,  0.09, ...,  0.56, 11.6 ,  0.6 ],
       [ 6.9 ,  0.3 ,  0.21, ...,  0.48,  9.  ,  0.5 ],
       ...,
       [ 6.3 ,  0.28,  0.34, ...,  0.29, 12.1 ,  0.6 ],
       [ 7.8 ,  0.3 ,  0.74, ...,  0.52, 12.8 ,  0.6 ],
       [ 6.8 ,  0.32,  0.18, ...,  0.41,  8.9 ,  0.5 ]])} data!
X (6000, 12)
y (6000, 1)
X_test (497, 12)
```

In [14]:

```
# split data
train_label = np.concatenate((wine_X, wine_y), axis=1)
Wcolname = np.array([str(i) for i in np.arange(wine_X.shape[1] + 1)])
train_label_df = pd.DataFrame(data = train_label, columns = Wcolname)
train_label_df.sample(frac = 1)

train_label_df = np.array(train_label_df)
X_train, y_train = train_label_df[1000:,:-1],train_label_df[1000:,-1]
X_val, y_val = train_label_df[:1000,:-1],train_label_df[:1000,-1]

y_train = np.array(pd.DataFrame(y_train))
```

In [15]:

```
# normalization
X_train_n = (X_train - X_train.mean(axis = 0))/X_train.std(axis = 0)
X_val_n = (X_val - X_val.mean(axis = 0))/X_val.std(axis = 0)
```

In [16]:

```
# int type of label
y_train = np.array(pd.DataFrame(y_train)).astype(int)
y_val = np.array(pd.DataFrame(y_val)).astype(int)
```

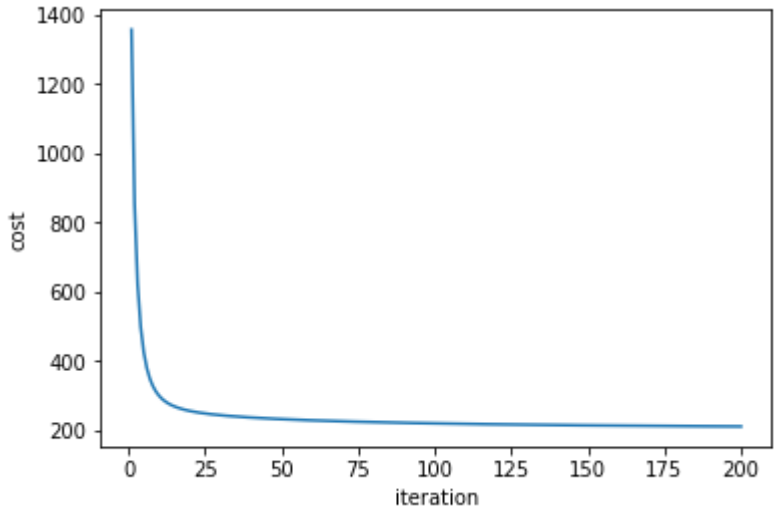
BGD

In [17]:

```
def BGD(X, y, a, lamb, max_iter):
    iter_cost = []
    X = np.concatenate((X,np.ones((X.shape[0],1))),axis = 1)
    w = np.random.randn(X.shape[1],1) # always be 13*1
    for i in range(max_iter):
        g = gradient(X,w,y,lamb)
        w = w - a * g  # always be 13*1
        iter_cost.append(loss(X,w,y,lamb)[0][0]) # single value
    return w,iter_cost

def gradient(X, w, y, lamb):
    return -X.T.dot(y - sigmoid(X, w)) + 2 * lamb * w # need 13 * 1

def sigmoid(X, w):
    return 1/(1 + np.exp(-X @ w)) # need to be 5000*1

def loss(X, w, y, lamb): # single value
    return -(y.T @ np.log(sigmoid(X, w) + np.spacing(1)) + (1 - y).T @ np. log(1 - sigm
oid(X, w) + np.spacing(1))) + lamb * (w.T @ w)
```

In [18]:

```
a = 1e-3
lamb = 1e0
max_iter = 200
w, iter_cost = BGD(X_train_n, y_train, a, lamb, max_iter);
fig = plt.figure()
plt.plot(range(1,max_iter+1), iter_cost)
plt.xlabel('iteration')
plt.ylabel('cost')
fig.savefig('BGD_loss_200iter.png')
```



In [19]:

```
# tune lambda
a = 1e-2
max_iter = 100
lambda_list= [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0, 1, 10]
```

In [20]:

```
def classifier(X_val,w):
    X_val = np.concatenate((X_val,np.ones((X_val.shape[0],1))),axis = 1)
    y_test = 1/(1+np.exp(-X_val @ w))
    y_test[y_test >= 0.5] = 1
    y_test[y_test < 0.5] =0
    y_test = y_test.astype(int)
    return y_test
```

In [21]:

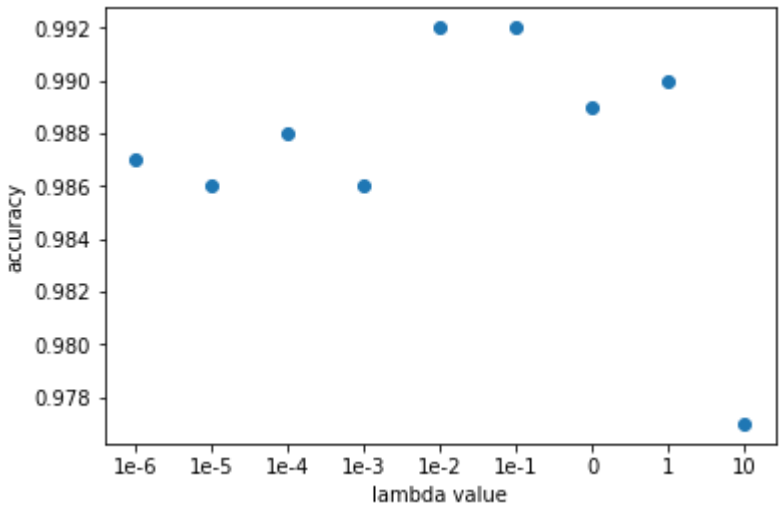
```
all_accuracy = []
for i in range(len(lambda_list)):
    w, iter_cost = BGD(X_train_n, y_train, a, lambda_list[i], max_iter)
    yp = classifier(X_val_n, w)
    all_accuracy.append(sum(yp == y_val) / 1000)
```

In [22]:

```
plt.figure()
plt.plot(range(len(lambda_list)), all_accuracy,'o')
plt.xlabel('lambda value')
plt.ylabel('accuracy')
plt.xticks(np.arange(9), ('1e-6', '1e-5', '1e-4', '1e-3', '1e-2', '1e-1', '0', '1', '1
0'))
```

Out[22]:

```
([<matplotlib.axis.XTick at 0x15e27d96f98>,
  <matplotlib.axis.XTick at 0x15e27d962b0>,
  <matplotlib.axis.XTick at 0x15e27d96710>,
  <matplotlib.axis.XTick at 0x15e27dc00b8>,
  <matplotlib.axis.XTick at 0x15e27dc0518>,
  <matplotlib.axis.XTick at 0x15e27dc09e8>,
  <matplotlib.axis.XTick at 0x15e27dc0eb8>,
  <matplotlib.axis.XTick at 0x15e27dc63c8>,
  <matplotlib.axis.XTick at 0x15e27dc6898>],
 <a list of 9 Text xticklabel objects>)
```



Kaggle

In [23]:

```
# lambda = 0
a = 1e-2
lamb = 1e0
max_iter = 5000
w, iter_cost = BGD(X_train_n, y_train, a, lamb, max_iter)
X_test_n = (wine_test - wine_test.mean(axis = 0))/wine_test.std(axis = 0)
yp = classifier(X_val_n, w)
sum(yp == y_val) / 1000
# plot(1:max_iter, iter_cost)
y_test = classifier(X_test_n, w)
```

In [24]:

```
y_test = y_test.flatten()
```

In [25]:

```
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission_wine.csv', index_label='Id')
results_to_csv(y_test)
```

In [10]:

```
# load data
import sys
if sys.version_info[0] < 3:
    raise Exception("Python 3 not detected.")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io
import random

wine = io.loadmat("data.mat")
print("\nloaded %s data!" % wine)
fields = "X", "y", "X_test"
for field in fields:
    print(field, wine[field].shape)

wine_X = wine['X']
wine_y = wine['y']
wine_test = wine['X_test']
```

```
loaded {'__header__': b'MATLAB 5.0 MAT-file Platform: posix, Created on: S
un Feb 26 19:25:34 2017', '__version__': '1.0', '__globals__': [], 'y': ar
ray([[0.],
       [1.],
       [0.],
       ...,
       [0.],
       [0.],
       [0.]]), 'X': array([[ 5.8  ,  0.555,  0.26 , ...,  0.46 ,  9.1  ,
0.5  ],
       [ 6.3  ,  0.36 ,  0.19 , ...,  0.52 , 12.7  ,  0.6  ],
       [ 6.4  ,  0.21 ,  0.5  , ...,  0.43 ,  8.8  ,  0.5  ],
       ...,
       [ 7.3  ,  0.2  ,  0.37 , ...,  0.49 , 10.9  ,  0.6  ],
       [ 8.5  ,  0.25 ,  0.27 , ...,  0.33 , 12.   ,  0.6  ],
       [ 6.6  ,  0.36 ,  0.21 , ...,  0.41 ,  9.9  ,  0.6  ]]), 'descripti
on': array(['fixed acidity       ', 'volatile acidity    ',
       'citric acid         ', 'residual sugar      ',
       'chlorides           ', 'free sulfur dioxide ',
       'total sulfur dioxide', 'density             ',
       'pH                  ', 'sulphates           ',
       'alcohol             ', 'quality             '], dtype='<U20'), 'X_
test': array([[ 8.2 ,  0.22,  0.49, ...,  0.33, 10.6 ,  0.6 ],
       [ 5.4 ,  0.74,  0.09, ...,  0.56, 11.6 ,  0.6 ],
       [ 6.9 ,  0.3 ,  0.21, ...,  0.48,  9.  ,  0.5 ],
       ...,
       [ 6.3 ,  0.28,  0.34, ...,  0.29, 12.1 ,  0.6 ],
       [ 7.8 ,  0.3 ,  0.74, ...,  0.52, 12.8 ,  0.6 ],
       [ 6.8 ,  0.32,  0.18, ...,  0.41,  8.9 ,  0.5 ]])} data!
X (6000, 12)
y (6000, 1)
X_test (497, 12)
```

In [11]:

```
# split data
train_label = np.concatenate((wine_X, wine_y), axis=1)
Wcolname = np.array([str(i) for i in np.arange(wine_X.shape[1] + 1)])
train_label_df = pd.DataFrame(data = train_label, columns = Wcolname)
train_label_df.sample(frac = 1)

train_label_df = np.array(train_label_df)
X_train, y_train = train_label_df[1000:,:-1],train_label_df[1000:,-1]
X_val, y_val = train_label_df[:1000,:-1],train_label_df[:1000,-1]

y_train = np.array(pd.DataFrame(y_train))
```

In [12]:

```
# normalization
X_train_n = (X_train - X_train.mean(axis = 0))/X_train.std(axis = 0)
X_val_n = (X_val - X_val.mean(axis = 0))/X_val.std(axis = 0)
```

In [13]:

```
# int type of label
y_train = np.array(pd.DataFrame(y_train)).astype(int)
y_val = np.array(pd.DataFrame(y_val)).astype(int)
```

In [14]:

```
def SGD(X, y, a, lamb, max_iter):
    iter_cost = []
    X = np.concatenate((X,np.ones((X.shape[0],1))),axis = 1)
    w = np.random.randn(X.shape[1],1) # always be 13*1
    for i in range(max_iter):
        if np.mod(i,1000) == 0:
            print('iter: '+ str(i))
        idex = np.ceil(random.random() * (X.shape[0]-1)).astype(int)
        Xi = X[idex,]
        yi = y[idex]
        s = 1/(1+np.exp(Xi*w))
        g = -Xi.T @ (yi-s) + 2*lamb*w
        w = w - a * g
        iter_cost.append(loss(X,w,y,lamb)[0][0])
    return w,iter_cost

def loss(X, w, y, lamb): # single value
    return -(y.T @ np.log(sigmoid(X, w) + np.spacing(1)) + (1 - y).T @ np. log(1 - sigm
oid(X, w) + np.spacing(1))) + lamb * (w.T @ w)

def sigmoid(X, w):
    return 1/(1 + np.exp(-X @ w)) # need to be 5000*1
```

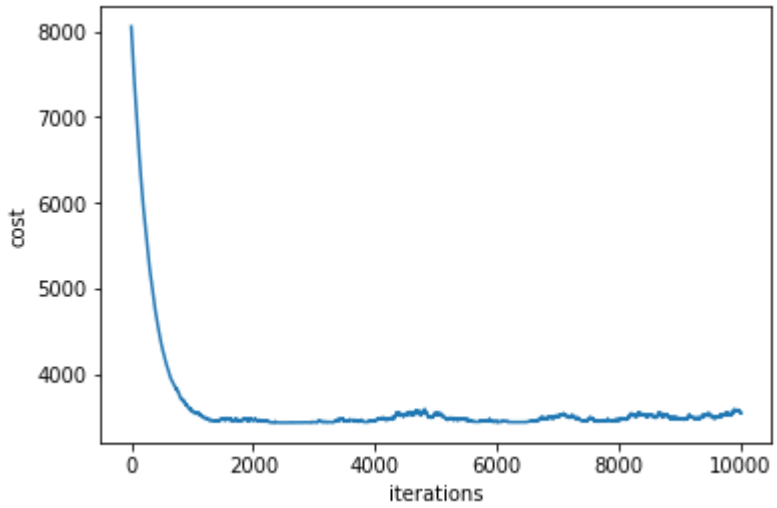
In [15]:

```
a = 1e-3
lamb = 1e0
max_iter = 10001
w, iter_cost = SGD(X_train_n, y_train, a, lamb, max_iter)
```

```
iter: 0
iter: 1000
iter: 2000
iter: 3000
iter: 4000
iter: 5000
iter: 6000
iter: 7000
iter: 8000
iter: 9000
iter: 10000
```

In [16]:

```
fig = plt.figure()
plt.plot(range(max_iter), iter_cost)
plt.xlabel('iterations')
plt.ylabel('cost')
fig.savefig('SGD_loss_10000iter.png')
```



In [17]:

```
def SGD_alpha(X, y, a, lamb, max_iter):
    iter_cost = []
    X = np.concatenate((X,np.ones((X.shape[0],1))),axis = 1)
    w = np.random.randn(X.shape[1],1) # always be 13*1
    for i in range(max_iter):
        if np.mod(i,1000) == 0:
            print('iter: '+ str(i))
        idex = np.ceil(random.random() * (X.shape[0]-1)).astype(int)
        Xi = X[idex,]
        yi = y[idex]
        s = 1/(1+np.exp(Xi*w))
        g = -Xi.T @ (yi-s) + 2*lamb*w
        w = w - a/(i+1) * g
        iter_cost.append(loss(X,w,y,lamb)[0][0])
    return w,iter_cost
```

In [18]:

```
a = 1e-3
lamb = 1e0
max_iter = 10001
w, iter_cost = SGD_alpha(X_train_n, y_train, a, lamb, max_iter)
```

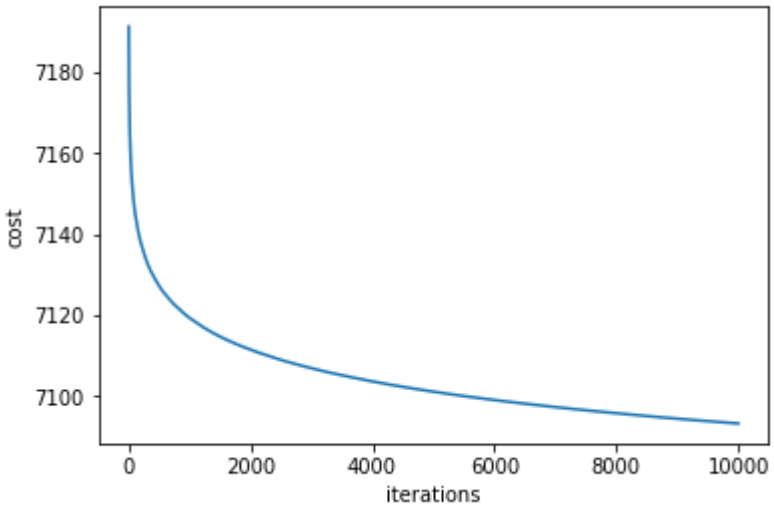
```
iter: 0
iter: 1000
iter: 2000
iter: 3000
iter: 4000
iter: 5000
iter: 6000
iter: 7000
iter: 8000
iter: 9000
iter: 10000
```

In [19]:

```
fig = plt.figure()
plt.plot(range(max_iter), iter_cost)
plt.xlabel('iterations')
plt.ylabel('cost')
fig.savefig('SGD_loss_10000iter_de_alpha.png')
```

Out[19]:

```
Text(0, 0.5, 'cost')
```



I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted.

Di Zhen