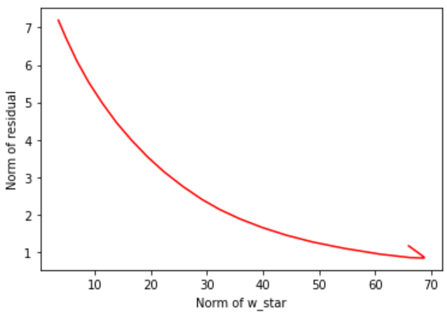
In [1]:
```python
def ista_solve_hot( A, d, la_array ):
    # ista_solve_hot: Iterative soft-thresholding for multiple values of
    # lambda with hot start for each case - the converged value for the previous
    # value of lambda is used as an initial condition for the current lambda.
    # this function solves the minimization problem
    # Minimize |Ax-d|_2^2 + lambda*|x|_1 (Lasso regression)
    # using iterative soft-thresholding.
    max_iter = 10**4
    tol = 10**(-3)
    tau = 1/np.linalg.norm(A,2)**2
    n = A.shape[1]
    w = np.zeros((n,1))
    num_lam = len(la_array)
    X = np.zeros((n, num_lam))
    for i, each_lambda in enumerate(la_array):
        for j in range(max_iter):
            z = w - tau*(A.T@(A@w-d))
            w_old = w
            w = np.sign(z) * np.clip(np.abs(z)-tau*each_lambda/2, 0, np.inf)
            X[:, i:i+1] = w
            if np.linalg.norm(w - w_old) < tol:
                break
    return X
```

1a)

In [2]:
```python
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
import pickle

X = loadmat("BreastCancer.mat")['X']
y = loadmat("BreastCancer.mat")['y']

X_100 = X[:100,:]
y_100 = y[:100,:]

lam = np.logspace(-8, np.log10(20), 100)

w_star = ista_solve_hot(X, y, lam)

print((X_100@w_star-y_100).shape)
coord1vals = []
coord2vals = []
for c in range(100):
    temp1 = np.linalg.norm(w_star[:,[c]], ord=1)
    coord1vals.append(temp1)
    temp2 = ((np.sum(((X_100@w_star[:,[c]]-y_100))**2))**0.5)
    #temp2 = (((np.sum((X_100@w_star-y_100)[:,[c]]))**2))**0.5
    coord2vals.append(temp2)

plt.plot(coord1vals, coord2vals, 'r')
plt.xlabel("Norm of w_star")
plt.ylabel("Norm of residual")
```

```
(100, 100)
```

Out[2]: Text(0, 0.5, 'Norm of residual')

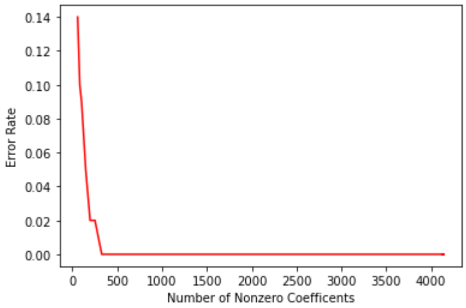


The lower l1 norm represents a lower lambda value. Small values of lambda increase the squared error. Thus we obtain such a graph where for high norm of w* we have a lower norm of residual.

1b)

In [3]:
```
def sparcity(w_arr):
    nonzeroCount = 0
    for i in w_arr:
        if i>10**-6:
            nonzeroCount += 1
    return nonzeroCount

nonzero_values = []
err = []

for i in range(100):
    nonzero_values.append(sparcity(w_star[:,i]))
    y_hat2 = np.sign(X_100@w_star[:,i:i+1])
    if np.all((y_hat2==0)):
        err.append(1)
        continue
    #err.append(error_rate(np.sign(X_100@w[:,i:i+1], y_train)))
    err.append(np.sum(np.abs(y_hat2-y_100))/2/100)

plt.plot(nonzero_values, err, 'r')
plt.xlabel("Number of Nonzero Coefficents")
plt.ylabel("Error Rate")
```

Out[3]:
```
Text(0, 0.5, 'Error Rate')
```



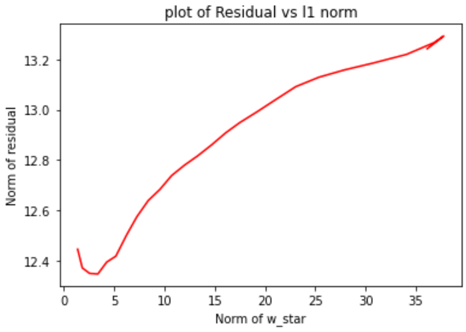
As the number of nonzero coefficients goes up, we get decreasing error rates. Intuitively, this makes perfect sense, as more and more data points are used in the calculation, and few a wasted as zero coefficients. Thus, the error rate tends to 0 at really high nonzero coefficients or low sparsity.

1c)

We repeat the same process with data from rows 101 to beyond.

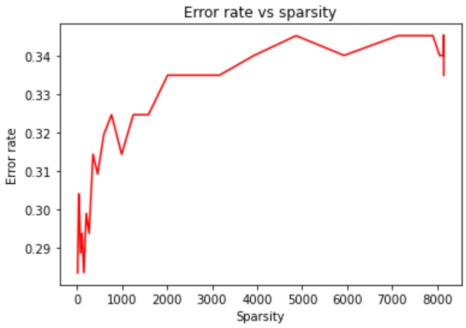
In [4]:
```
X_c = X[101:]
y_c = y[101:]
lambda_arr = np.logspace(-8,np.log10(20),100)
w_star = ista_solve_hot(X_100,y_100,lambda_arr)
coordx = []
coordy = []
for c in range(len(w_star[0])):
    w_temp = w_star[:, c:c+1]
    coordx.append(np.linalg.norm(w_temp,ord=1))
    coordy.append(np.linalg.norm(X_c@w_temp-y_c))
plt.plot(coordx,coordy, 'r')

plt.xlabel("Norm of w_star")
plt.ylabel("Norm of residual")

plt.title("plot of Residual vs l1 norm")
```

Out[4]:
```
Text(0.5, 1.0, 'plot of Residual vs l1 norm')
```



In [5]:
```
coordx = []
coordy = []
for i in range(len(w_star[0])):
    w = w_star[:, i:i+1]
    d_hat = np.sign(X_c@w)
    error_vec = [0 if m[0]==m[1] else 1 for m in np.hstack((d_hat, y_c))]
    error_rate = sum(error_vec)/len(error_vec)
    sparsity = 0
    for j in w:
        if j!=0:
            sparsity+=1
    coordx.append(sparsity)
    coordy.append(error_rate)
plt.plot(coordx, coordy, 'r')

plt.xlabel("Sparsity")
plt.ylabel("Error rate")
plt.title("Error rate vs sparsity")
```

Out[5]:
```
Text(0.5, 1.0, 'Error rate vs sparsity')
```



We see a completely different plot for the first plot, whereas in the first case there was an inverse relation between ||w*||1 and ||Aw∗ −d||2 in the second case the relation is directly proportional. The error rate wrt sparsity increases a lot initially in this case whereas it does increase as much initially, and instead falls in (b). Thus we conclude that the training set is not a good fit for the data. More features incorporated makes the error rate go up.

In [ ]:

In [ ]:

Q2)

In [1]:

```
def ista_solve_hot( A, d, la_array ):
    # ista_solve_hot: Iterative soft-thresholding for multiple values of
    # lambda with hot start for each case - the converged value for the previous
    # value of lambda is used as an initial condition for the current lambda.
    # this function solves the minimization problem
    # Minimize |Ax-d|_2^2 + lambda*|x|_1 (Lasso regression)
    # using iterative soft-thresholding.
    max_iter = 10**4
    tol = 10**(-3)
    tau = 1/np.linalg.norm(A,2)**2
    n = A.shape[1]
    w = np.zeros((n,1))
    num_lam = len(la_array)
    X = np.zeros((n, num_lam))
    for i, each_lambda in enumerate(la_array):
        for j in range(max_iter):
            z = w - tau*(A.T@(A@w-d))
            w_old = w
            w = np.sign(z) * np.clip(np.abs(z)-tau*each_lambda/2, 0, np.inf)
            X[:, i:i+1] = w
            if np.linalg.norm(w - w_old) < tol:
                break
    return X

def ridge( A, d, la_array ):
    n = A.shape[1]
    num_lam = len(la_array)
    X = np.zeros((n, num_lam))
    for i, each_lambda in enumerate(la_array):
        w = A.T@np.linalg.inv(A@A.T + each_lambda*np.eye(len(A@A.T)))@d
        X[:, i:i+1] = w
    return X
```

In [2]:
```python
## Breast Cancer LASSO Exploration
## Prepare workspace
from scipy.io import loadmat
import numpy as np
X = loadmat("BreastCancer.mat")['X']
y = loadmat("BreastCancer.mat")['y']

##  10-fold CV

# each row of setindices denotes the starting an ending index for one
# partition of the data: 5 sets of 30 samples and 5 sets of 29 samples
setindices = [[1,30],[31,60],[61,90],[91,120],[121,150],[151,179],[180,208],[209,237],[238,266],[267,295]]

# each row of holdoutindices denotes the partitions that are held out from
# the training set
holdoutindices = [[1,2],[2,3],[3,4],[4,5],[5,6],[7,8],[9,10],[10,1]]

cases = len(holdoutindices)

# be sure to initiate the quantities you want to measure before looping
# through the various training, validation, and test partitions

avg_error_rate_lasso = 0
avg_sqd_error_ridge = 0
avg_error_rate_ridge = 0
avg_sqd_error_lasso = 0
lam_vals = np.logspace(-8,np.log10(20),100)

# Loop over various cases
for j in range(cases):
    print("Iteration: ", j+1)
    # row indices of first validation set
    v1_ind = np.arange(setindices[holdoutindices[j][0]-1][0]-1,setindices[holdoutindices[j][0]-1][1])

    # row indices of second validation set
    v2_ind = np.arange(setindices[holdoutindices[j][1]-1][0]-1,setindices[holdoutindices[j][1]-1][1])

    # row indices of training set
    trn_ind = list(set(range(295))-set(v1_ind)-set(v2_ind))

    # define matrix of features and labels corresponding to first
    # validation set
    Av1 = X[v1_ind,:]
    bv1 = y[v1_ind]

    # define matrix of features and labels corresponding to second
    # validation set
    Av2 = X[v2_ind,:]
    bv2 = y[v2_ind]

    # define matrix of features and labels corresponding to the
    # training set
    At = X[trn_ind,:]
    bt = y[trn_ind]

    print(len(v1_ind), len(v2_ind), len(trn_ind))

# Use training data to learn classifier weights
    wmat_lasso = ista_solve_hot(At,bt,lam_vals)
    wmat_ridge = ridge(At,bt,lam_vals)
    lam_lasso = None
    w_opt_lasso = None
    min_error_lasso = -1
    lam_ridge = None
    w_opt_ridge= None
    min_error_ridge = -1

# Find best lambda value using the first validation set, then evaluate everything from there
    for i in range(len(wmat_lasso[0])):
        w_lasso = wmat_lasso[:, i:i+1]
        d_hat_lasso = np.sign(Av1@w_lasso)
        error_vec_lasso = [0 if m[1]==m[0] else 1 for m in np.hstack((d_hat_lasso, bv1))]
        error_rate_lasso = sum(error_vec_lasso)/len(error_vec_lasso)
        if min_error_lasso == -1 or error_rate_lasso<min_error_lasso:
            min_error_lasso = error_rate_lasso
            lam_lasso = lam_vals[i]
            w_opt_lasso = w_lasso
        w_ridge = wmat_ridge[:, i:i+1]
        d_hat_ridge = np.sign(Av1@w_ridge)
        error_vec_ridge = [0 if m[1]==m[0] else 1 for m in np.hstack((d_hat_ridge, bv1))]
        error_rate_ridge = sum(error_vec_ridge)/len(error_vec_ridge)
        if min_error_ridge == -1 or error_rate_ridge<min_error_ridge:
            min_error_ridge = error_rate_ridge
            lam_ridge = lam_vals[i]
            w_opt_ridge = w_ridge

# perform on second validation set, and accumulate performance metrics over all cases and their respective splittings
    print("Best lambda in LASSO: ", lam_lasso)
    print("Best lambda in ridge regression: ", lam_ridge)
    d_hat_lasso = np.sign(Av2@w_opt_lasso)
    error_vec_lasso = [0 if m[1]==m[0] else 1 for m in np.hstack((d_hat_lasso,bv2))]
    error_rate_lasso = sum(error_vec_lasso)/len(error_vec_lasso)
    avg_error_rate_lasso += error_rate_lasso
    squared_error_lasso = np.linalg.norm(d_hat_lasso-bv2)**2
    avg_sqd_error_lasso += squared_error_lasso
    print("Error rate of lasso: ", error_rate_lasso)
    print("Squared error of lasso: ", squared_error_lasso)
    print("avg error rate of lasso: ",  avg_error_rate_lasso)
    d_hat_ridge = np.sign(Av2@w_opt_ridge)
    error_vec_ridge = [0 if m[1]==m[0] else 1 for m in np.hstack((d_hat_ridge,bv2))]
    error_rate_ridge = sum(error_vec_ridge)/len(error_vec_ridge)
```

```python
    avg_error_rate_ridge += error_rate_ridge
    squared_error_ridge = np.linalg.norm(d_hat_ridge-bv2)**2
    avg_sqd_error_ridge += squared_error_ridge
    print("Error rate of ridge: ", error_rate_ridge)
    print("Squared error of ridge: ", squared_error_ridge)
    print("avg error rate of ridge: ",  avg_error_rate_ridge)
    print("\n\n")
avg_error_rate_ridge /= cases
avg_sqd_error_ridge /= cases
avg_error_rate_lasso /= cases
avg_sqd_error_lasso /= cases
print("Average error rate of lasso: ", avg_error_rate_lasso)
print("Average squared error of lasso: ", avg_sqd_error_lasso)
print("Average error rate of ridge: ", avg_error_rate_ridge)
print("Average squared error of ridge: ", avg_sqd_error_ridge)
```

```
Iteration:  1
30 30 235
Best lambda in LASSO:  3.5434932692979846
Best lambda in ridge regression:  1e-08
Error rate of lasso:  0.43333333333333335
Squared error of lasso:  51.99999999999999
avg error rate of lasso:  0.43333333333333335
Error rate of ridge:  0.43333333333333335
Squared error of ridge:  51.99999999999999
avg error rate of ridge:  0.43333333333333335



Iteration:  2
30 30 235
Best lambda in LASSO:  16.109430878355187
Best lambda in ridge regression:  1e-08
Error rate of lasso:  0.26666666666666666
Squared error of lasso:  32.00000000000001
avg error rate of lasso:  0.7
Error rate of ridge:  0.23333333333333334
Squared error of ridge:  28.000000000000004
avg error rate of ridge:  0.6666666666666667



Iteration:  3
30 30 235
Best lambda in LASSO:  3.5434932692979846
Best lambda in ridge regression:  1e-08
Error rate of lasso:  0.4666666666666667
Squared error of lasso:  56.0
avg error rate of lasso:  1.1666666666666665
Error rate of ridge:  0.43333333333333335
Squared error of ridge:  51.99999999999999
avg error rate of ridge:  1.1



Iteration:  4
30 30 235
Best lambda in LASSO:  6.780801107819854
Best lambda in ridge regression:  1e-08
Error rate of lasso:  0.13333333333333333
Squared error of lasso:  16.0
avg error rate of lasso:  1.2999999999999998
Error rate of ridge:  0.2
Squared error of ridge:  23.999999999999996
avg error rate of ridge:  1.3



Iteration:  5
30 29 236
Best lambda in LASSO:  1.4915314679609128
Best lambda in ridge regression:  1e-08
Error rate of lasso:  0.3103448275862069
Squared error of lasso:  36.0
avg error rate of lasso:  1.6103448275862067
Error rate of ridge:  0.3103448275862069
Squared error of ridge:  36.0
avg error rate of ridge:  1.610344827586207



Iteration:  6
29 29 237
Best lambda in LASSO:  16.109430878355187
Best lambda in ridge regression:  1e-08
Error rate of lasso:  0.13793103448275862
Squared error of lasso:  16.0
avg error rate of lasso:  1.7482758620689653
Error rate of ridge:  0.1724137931034483
Squared error of ridge:  20.000000000000004
avg error rate of ridge:  1.782758620689655



Iteration:  7
29 29 237
Best lambda in LASSO:  20.000000000000004
Best lambda in ridge regression:  1e-08
Error rate of lasso:  0.5172413793103449
Squared error of lasso:  60.00000000000001
avg error rate of lasso:  2.2655172413793103
```

```
Error rate of ridge:  0.4482758620689655
Squared error of ridge:  51.99999999999999
avg error rate of ridge:  2.231034482758621



Iteration:  8
29 30 236
Best lambda in LASSO:  3.955000701930882e-07
Best lambda in ridge regression:  1e-08
Error rate of lasso:  0.2
Squared error of lasso:  23.999999999999996
avg error rate of lasso:  2.4655172413793105
Error rate of ridge:  0.2
Squared error of ridge:  23.999999999999996
avg error rate of ridge:  2.431034482758621



Average error rate of lasso:  0.3081896551724138
Average squared error of lasso:  36.5
Average error rate of ridge:  0.3038793103448276
Average squared error of ridge:  36.0
```

In [ ]: