Table 1: The samples collected from geo database used in this study.

|  |  |  |
| --- | --- | --- |
|  | Cancer | Normal |
| GSE29265 | 20 | 20 |
| GSE33630 | 49 | 45 |
| GSE29315 | 9 | 17 |
| GSE65074 | 38 | 0 |
| GSE60542 | 33 | 26 |
| GSE27155 | 51 | 21 |
| GSE129562 | 8 | 8 |
| GSE53157 | 7 | 3 |
| GSE3467 | 9 | 9 |

Figure S1 A: DEGs from train set (GSE29265, GSE33630, GSE29315 and GSE65074). B: Hierarchical clustering analysis of mRNAs of the train set, which were differentially expressed between tumor and normal tissues. C: Hierarchical clustering analysis of mRNAs of the test set (GSE60542, GSE27155, GSE129562, GSE53157 and GSE3467), which were differentially expressed between tumor and normal tissues.All microarray data was submitted to the GEO database (http://www.ncbi.nih.gov/geo). The raw data were downloaded as MINiML files. The extracted data were normalized and processed by log2 transformation. the microarray data were normalized using the preprocess Core package in R software (version 3.4.1). Probes were converted to gene symbols according to the platform annotation information of the normalized data. Probes with more than one gene were eliminated and the average value was calculated for genes corresponding to more than one probe. As an initial quality control step using variance stabilized counts with individual horse effect removed using the remove Batch Effect function of limma R package.

The Python code used in this study:

Code for automatic generation of gene list:

import pandas as pd

oldCSVFILE = "test000.csv"

newCSVFILE='bb.csv'

if \_\_name\_\_ == '\_\_main\_\_':

f = open("1.txt", "r")

indexnames = f.readlines()

csvFile = pd.read\_csv(oldCSVFILE,index\_col=0).T

df = pd.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)

df['Tag']='' # csvFile.\_stat\_axis.values.tolist()''

for i in indexnames:

indexname = i.replace("\n", "").upper()

if indexname in csvFile:

df[indexname] = csvFile[indexname]

else:

df[indexname] = ''

df['Tag']=csvFile.\_stat\_axis.values.tolist()

df.to\_csv(newCSVFILE, index=None)

Code for DNN training models:

import numpy as np

import pandas as pd

import torch

import torch.nn as nn

import torch.optim as optim

import torch.utils.data as Data

from matplotlib import pyplot as plt

torch.set\_default\_tensor\_type(torch.DoubleTensor)

torch.manual\_seed(1)

input\_size = 10

hidden\_1 = 1600

hidden\_2 = 1400

hidden\_3 = 500

lr = 0.001

BATCH\_SIZE = 20

train = pd.read\_csv('train.csv')

test = pd.read\_csv('test.csv')

def loadData(data):

feature\_size = data.iloc[1, 2:].shape[0]

x = torch.from\_numpy(data.iloc[:, 2:].values.astype('float64'))

y = np.array(data.iloc[:, 1].values)

y[y == 'patient'] = 1

y[y == 'normal'] = 0

y = np.array([np.eye(2)[i] for i in y])

y = torch.from\_numpy(y.astype('float64'))

data\_dataset = Data.TensorDataset(x, y)

return feature\_size, data\_dataset

input\_size, torch\_dataset = loadData(train)

loader = Data.DataLoader(

dataset=torch\_dataset,

batch\_size=BATCH\_SIZE,

shuffle=True,

num\_workers=4, )

class Net(nn.Module):

def \_\_init\_\_(self, ):

super(Net, self).\_\_init\_\_()

self.model = nn.Sequential(

nn.Linear(input\_size, hidden\_1),

nn.ReLU(),

nn.Linear(hidden\_1, hidden\_2),

nn.ReLU(),

nn.Linear(hidden\_2, hidden\_3),

nn.ReLU(),

nn.Linear(hidden\_3, 2),

# nn.Softmax(dim=1)

def forward(self, x):

out = self.model(x)

return out

if \_\_name\_\_ == '\_\_main\_\_':

model = Net()

criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr)

for epoch in range(10000):

for step, (batch\_x, batch\_y) in enumerate(loader):

y\_pred = model(batch\_x)

loss = criterion(y\_pred, batch\_y)

model.zero\_grad()

loss.backward()

optimizer.step()

if epoch % 5 == 0:

print(epoch, loss.item())

# torch.save(model, 'save/' + str(epoch) + 'model.pkl')

if epoch % 100 == 0:

torch.save(model, 'save/' + str(epoch) + 'model.pkl')

Code for DNN models predicting:

import torch

import numpy as np

import torch.nn as nn

import torch.optim as optim

import pandas as pd

import torch.utils.data as Data

from matplotlib import pyplot as plt

from matplotlib.pyplot import MultipleLocator

torch.set\_default\_tensor\_type(torch.DoubleTensor)

torch.manual\_seed(1)

num\_time\_steps = 50

output\_size = 1

input\_size = 10

hidden\_1 = 200

hidden\_2 = 100

hidden\_3 = 50

lr = 0.001

BATCH\_SIZE = 20

# train = pd.read\_csv('train2.csv')

test = pd.read\_csv('test.csv')

pklpath = r'save\1600model.pkl'

def loadData(data):

x = torch.from\_numpy(data.iloc[:, 2:].values.astype('float64'))

y = np.array(data.iloc[:, 1].values)

y[y == 'patient'] = 1

y[y == 'normal'] = 0

y = torch.from\_numpy(y.astype('float64'))

return x, y

class Net(nn.Module):

def \_\_init\_\_(self, ):

super(Net, self).\_\_init\_\_()

self.model = nn.Sequential(

nn.Linear(input\_size, hidden\_1),

nn.ReLU(),

nn.Linear(hidden\_1, hidden\_2),

nn.ReLU(),

nn.Linear(hidden\_2, hidden\_3),

nn.ReLU(),

nn.Linear(hidden\_3, 2),

nn.Softmax(dim=1) )

def forward(self, x):

out = self.model(x)

return out

if \_\_name\_\_ == '\_\_main\_\_':

model = torch.load(pklpath)

criterion = nn.MSELoss()

a = 0

x, y = loadData(test)

y\_pred = model(x)

y\_max, idx = model(x).max(1)

aresult = torch.stack([y, idx], dim=1)

num = len(idx)

loss\_ = (idx - y)

print("", y)

print("", idx)

print("", loss\_)

print("", np.nonzero(loss\_.numpy()))

# print(np.nonzero(a))

loss = np.nonzero(loss\_.numpy())

loss = len(loss[0])

print("", 1 - loss / num)

y\_s = y.numpy()

a00 = a01 = a10 = a11 = 0

for index in range(len(y\_s)):

if y\_s[index] == 0:

if loss\_[index] == 0:

a00 = a00 + 1

else:

a10 = a10 + 1

else:

if loss\_[index] == 0:

a11 = a11 + 1

else:

a01 = a01 + 1

print("predict\t", 0, '\t', 1)

print("label")

print(0, '\t\t', a00, a01)

print(1, '\t\t', a10, a11)

roc = []

x = []

y = []

xy=[]

y\_roc = torch.index\_select(y\_pred, 1, torch.tensor([1])).detach().numpy()

y\_roc2 = y\_roc.copy()

for i in np.arange(0, 1, 0.01):

y\_roc = y\_roc2.copy()

y\_roc[y\_roc >= i] = 1

y\_roc[y\_roc < i] = 0

TP = FN = FP = TN = 0

for index in range(len(y\_s)):

if y\_s[index] == 1:

if y\_roc[index] == 1:

TP = TP + 1

else:

FN = FN + 1

else:

if y\_roc[index] == 1:

FP = FP + 1

else:

TN = TN + 1

# print(TP, TN, FP, TN)

TRP = 1.0 \* TP / (TP + FN)

FPR = 1.0 \* FP / (FP + TN)

x.append(FPR)

y.append(TRP)

xy.append([FPR, TRP])

data = np.array(xy)

xy = data[np.argsort(data[:, 0])].tolist()

plt.xlim(0, 1)

plt.ylim(0, 1)

ax = plt.gca()

ax.xaxis.set\_major\_locator(MultipleLocator(0.1))

ax.yaxis.set\_major\_locator(MultipleLocator(0.1))

plt.plot(x, y, label='linear')

print(xy)

auc = 0.

prev\_x = 0

for x1, y1 in xy:

if x1 != prev\_x:

auc += (x1 - prev\_x) \* y1

prev\_x = x1

print("AUC" + str(auc))

plt.title("ROC")

plt.xlabel("1-Specificity")

plt.ylabel("Sensitivity")

plt.show()

Code for CNN training models:

import numpy as np

import pandas as pd

import torch

import torch.nn as nn

import torch.optim as optim

import torch.utils.data as Data

import math

from matplotlib import pyplot as plt

torch.set\_default\_tensor\_type(torch.DoubleTensor)

torch.manual\_seed(1)

input\_size = 10

in\_channels = 1;

BATCH\_SIZE = 1

lr = 0.001

Conv1d\_out\_ch = 64

Conv1d\_kernel\_size = 5

Conv1d\_pool\_kernel\_size = 2

Conv1d\_out\_stride = 1

Conv1d\_out\_padding = (Conv1d\_kernel\_size - 1) // 2

Conv2d\_out\_ch = 256

Conv2d\_kernel\_size = 5

Conv2d\_pool\_kernel\_size = 2

Conv2d\_out\_stride = 1

Conv2d\_out\_padding = (Conv2d\_kernel\_size - 1) // 2

print\_epoch = 5

save\_epoch = 100

epoch\_size = 10000

lin\_input\_size = 0

train = pd.read\_csv('train.csv')

def loadData(data):

feature\_size = data.iloc[1, 2:].shape[0]

x = torch.from\_numpy(data.iloc[:, 2:].values.astype('float64'))

y = np.array(data.iloc[:, 1].values)

y[y == 'patient'] = 1

y[y == 'normal'] = 0

y = np.array([np.eye(2)[i] for i in y])

y = torch.from\_numpy(y.astype('float64'))

data\_dataset = Data.TensorDataset(x, y)

input\_size, torch\_dataset = loadData(train)

lin\_input\_size = int(input\_size // Conv1d\_pool\_kernel\_size // Conv2d\_pool\_kernel\_size \* Conv2d\_out\_ch)

loader = Data.DataLoader(

dataset=torch\_dataset,

batch\_size=BATCH\_SIZE,

shuffle=True,

num\_workers=4, )

class Net(nn.Module):

def \_\_init\_\_(self, ):

super(Net, self).\_\_init\_\_()

self.conv1 = nn.Sequential(

nn.Conv1d(

in\_channels=in\_channels,

out\_channels=Conv1d\_out\_ch,

kernel\_size=Conv1d\_kernel\_size,

stride=Conv1d\_out\_stride,

padding=Conv1d\_out\_padding, ),

nn.ReLU(),

nn.MaxPool1d(kernel\_size=Conv1d\_pool\_kernel\_size)

)

self.conv2 = nn.Sequential(

nn.Conv1d(

in\_channels=Conv1d\_out\_ch,

out\_channels=Conv2d\_out\_ch,

kernel\_size=Conv2d\_kernel\_size,

stride=Conv2d\_out\_stride,

padding=Conv2d\_out\_padding,

),

nn.ReLU(),

nn.MaxPool1d(kernel\_size=Conv2d\_pool\_kernel\_size)

)

self.out = nn.Linear(lin\_input\_size, 2)

def forward(self, x):

x = self.conv1(x)

x = self.conv2(x)

x = x.view(x.size(0), -1)

out = self.out(x)

return out

if \_\_name\_\_ == '\_\_main\_\_':

model = Net()

criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr)

for epoch in range(epoch\_size):

for step, (batch\_x, batch\_y) in enumerate(loader):

y\_pred = model(batch\_x.unsqueeze(0))

loss = criterion(y\_pred, batch\_y)

model.zero\_grad()

loss.backward()

optimizer.step()

if epoch % print\_epoch == 0:

print(epoch, loss.item())

# torch.save(model, 'save/' + str(epoch) + 'model.pkl')

if epoch % save\_epoch == 0:

torch.save(model, 'save/' + str(epoch) + 'model.pkl')

Code for DNN models predicting:

import torch

import numpy as np

import torch.nn as nn

from torch.nn import functional as F

import torch.optim as optim

import pandas as pd

import torch.utils.data as Data

from matplotlib import pyplot as plt

from matplotlib.pyplot import MultipleLocator

from functools import cmp\_to\_key

torch.set\_default\_tensor\_type(torch.DoubleTensor)

torch.manual\_seed(1)

num\_time\_steps = 50

output\_size = 1

input\_size = 10

hidden\_1 = 200

hidden\_2 = 100

hidden\_3 = 50

lr = 0.001

BATCH\_SIZE = 20

# train = pd.read\_csv('train2.csv')

test = pd.read\_csv('test.csv')

pklpath = r'save\100model.pkl'

def loadData(data):

x = torch.from\_numpy(data.iloc[:, 2:].values.astype('float64'))

y = np.array(data.iloc[:, 1].values)

y[y == 'patient'] = 1

y[y == 'normal'] = 0

y = torch.from\_numpy(y.astype('float64'))

return x, y

class Net(nn.Module):

def \_\_init\_\_(self, ):

super(Net, self).\_\_init\_\_()

self.conv1 = nn.Sequential(

nn.Conv1d(

in\_channels=1,

out\_channels=64,

kernel\_size=11,

stride=1,

padding=5,

),

nn.ReLU(),

nn.MaxPool1d(kernel\_size=2)

)

self.conv2 = nn.Sequential(

nn.Conv1d(

in\_channels=64,

out\_channels=256,

kernel\_size=11,

stride=1,

padding=5,

),

nn.ReLU(),

nn.MaxPool1d(kernel\_size=2)

)

self.out1 = nn.Sequential(

nn.Linear(256 / 2 / 2 \* input\_size, 2),

nn.Softmax(dim=1)

)

def forward(self, x):

x = self.conv1(x)

x = self.conv2(x)

x = x.view(x.size(0), -1)

out = self.out(x)

return out

if \_\_name\_\_ == '\_\_main\_\_':

model = torch.load(pklpath)

criterion = nn.MSELoss()

a = 0

x, y = loadData(test)

y\_pred = []

for i in range(x.shape[0]):

y\_pred\_ = model(x[i].unsqueeze(0).unsqueeze(0))

y\_pred.append(y\_pred\_.detach().numpy().tolist()[0])

y\_pred = torch.Tensor(y\_pred)

# print(y\_pred)

y\_pred = F.softmax(y\_pred, dim=1)

# print(y\_pred)

y\_max, idx = y\_pred.max(1)

print(idx)

aresult = torch.stack([y, idx], dim=1)

num = len(idx)

loss\_ = (idx - y)

print("", y)

print("", idx)

print("", loss\_)

print("", np.nonzero(loss\_.numpy()))

# print(np.nonzero(a))

loss = np.nonzero(loss\_.numpy())

loss = len(loss[0])

print("", 1 - loss / num)

y\_s = y.numpy()

a00 = a01 = a10 = a11 = 0

for index in range(len(y\_s)):

if y\_s[index] == 0:

if loss\_[index] == 0:

a00 = a00 + 1

else:

a10 = a10 + 1

else:

if loss\_[index] == 0:

a11 = a11 + 1

else:

a01 = a01 + 1

print("predict\t", 0, '\t', 1)

print("label")

print(0, '\t\t', a00, a01)

print(1, '\t\t', a10, a11)

roc = []

x = []

y = []

y\_roc = torch.index\_select(y\_pred, 1, torch.tensor([1])).detach().numpy()

y\_roc2 = y\_roc.copy()

xy = []

for i in np.arange(0, 1, 0.01):

y\_roc = y\_roc2.copy()

y\_roc[y\_roc >= i] = 1

y\_roc[y\_roc < i] = 0

TP = FN = FP = TN = 0

for index in range(len(y\_s)):

if y\_s[index] == 1:

if y\_roc[index] == 1:

TP = TP + 1

else:

FN = FN + 1

else:

if y\_roc[index] == 1:

FP = FP + 1

else:

TN = TN + 1

TRP = 1.0 \* TP / (TP + FN)

FPR = 1.0 \* FP / (FP + TN)

x.append(FPR)

y.append(TRP)

xy.append([FPR, TRP])

data = np.array(xy)

xy = data[np.argsort(data[:, 0])].tolist()

plt.xlim(0, 1)

plt.ylim(0, 1)

ax = plt.gca()

ax.xaxis.set\_major\_locator(MultipleLocator(0.1))

ax.yaxis.set\_major\_locator(MultipleLocator(0.1))

plt.plot(x, y, label='linear')

print(xy)

auc = 0.

prev\_x = 0

for x1, y1 in xy:

if x1 != prev\_x:

auc += (x1 - prev\_x) \* y1

prev\_x = x1

print("AUC" + str(auc))

plt.title("ROC")

plt.xlabel("1-Specificity")

plt.ylabel("Sensitivity")

plt.show()

One of the confusion matrixes of prediction using DNN model from m6a genes list:

predict 0 1

label

0 0 0

1. 67 108

Table S2 genes list used as input features in the research (because the table was large, we uploaded the excel file separately).

Table S3 Parameters of DNN artificial neural network models training.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | hidden layers | hidden 1 | hidden 2 | hidden 3 | learning rate | in put size | batch size |
| DEGs list | 3 | 400 | 200 | 100 | 0.001 | 10 | 20 |
| all genes list | 1 | 3600 | - | - | 0.0001 | 20 | 20 |
| DDR Genes list | 3 | 400 | 200 | 100 | 0.001 | 10 | 20 |
| anaerobic energy metabolism genes list | 3 | 2000 | 1000 | 500 | 0.001 | 10 | 20 |
| lipid metabolism related genes list | 3 | 1000 | 500 | 250 | 0.001 | 10 | 20 |
| EMT genes list | 3 | 2000 | 1000 | 500 | 0.001 | 10 | 20 |
| hypoxia-related genes list | 3 | 100 | 50 | 25 | 0.001 | 10 | 20 |
| IS genes list | 3 | 4000 | 2000 | 1000 | 0.001 | 20 | 20 |
| m6a genes list | - | - | - | - | - | - | - |
| ferroptosis related genes list | 3 | 40 | 20 | 10 | 0.0001 | 10 | 20 |

Table S4 Parameters of CNN artificial neural network models training.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | convolution layer | Conv 1d out | Conv 2d out | Conv 1d kernel size | Conv 1d pool kernel size | Conv 1d kernel size | Conv 1d pool kernel size |
| DEGs list | 2 | 64 | 256 | 5 | 2 | 0.0001 | 10 |
| all genes list | 2 | 256 | 1024 | 10 | 2 | 0.00001 | 20 |
| DDR Genes list | 2 | 128 | 512 | 5 | 2 | 0.0001 | 10 |
| anaerobic energy metabolism genes list | 2 | 128 | 512 | 5 | 2 | 0.0001 | 10 |
| lipid metabolism related genes list | 2 | 256 | 1024 | 10 | 2 | 0.0001 | 10 |
| EMT genes list | 2 | 128 | 512 | 5 | 2 | 0.0001 | 10 |
| hypoxia-related genes list | 2 | 16 | 64 | 5 | 2 | 0.0001 | 10 |
| IS genes list | 2 | 128 | 512 | 5 | 2 | 0.0001 | 20 |
| m6a genes list | 2 | 128 | 512 | 5 | 2 | 0.0001 | 10 |
| ferroptosis related genes list | 2 | 64 | 256 | 5 | 2 | 0.00001 | 10 |