```python
from IPython import display
import numpy as np

df = pd.read_csv('game1_train_data_numeric.csv')
print("dataset size: %d" % len(df))
print("dataset columns size: %d" % len(df.columns))
df = df.dropna(subset=["rtype_u"])
df = df.fillna(method="backfill")
df = df.dropna()
df = (df - df.min()) / (df.max() - df.min())

display.display(df.head())

y = df['rtype_u']
x = df.drop(columns='rtype_u')
x = x.drop(columns='d')

print("dataset size: %d" % len(df))
print("dataset columns size: %d" % len(df.columns))
```

```
dataset size: 8314
dataset columns size: 110
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Unnamed: 0	PROVINCE	urban	age	gender	l1_3_1	party	army	sec_lan	f_b_hukou	...	fedu	medu
0	0.000000	0.611111	0.0	0.169014	0.0	0.618183	0.0	0.0	0.0	0.0	...	0.00	0.0
1	0.000121	0.462963	1.0	0.492958	0.0	0.472729	0.0	0.0	0.0	0.0	...	0.00	0.0
2	0.000241	0.425926	0.0	0.422535	1.0	0.436366	0.0	0.0	0.0	0.0	...	0.00	0.0
3	0.000362	0.574074	0.0	0.619718	1.0	0.581820	0.0	0.0	0.0	0.0	...	0.25	0.0
4	0.000483	0.722222	1.0	0.549296	0.0	0.727274	0.0	0.0	0.0	0.0	...	0.00	0.0

5 rows × 110 columns

```
dataset size: 8285
dataset columns size: 110
```

```python
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

x_train, _, y_train, _ = train_test_split(x, y, test_size=0.1, random_state=42)

dft = pd.read_csv('game1_test_data_numeric.csv')
print(len(df))
print(len(df.columns))
dft = dft.dropna(subset=["rtype_u"])
dft = dft.fillna(method="backfill")
dft = dft.dropna()
dft = (dft - dft.min()) / (dft.max() - dft.min())

y_test = dft['rtype_u']
x_test = dft.drop(columns='rtype_u')
x_test = x_test.drop(columns='d')
display.display(y_test.head())
display.display(x_test.head())
print(len(dft))
print(len(dft.columns))
```

```
def show_results(clf):
    # global x_train, x_test, y_train, y_test
    print("R^2 on Train: %s" % clf.score(x_train, y_train));
    print("Accuracy On Train: %s" % accuracy_score(y_train, clf.predict(x_train)));
    print("R^2 on Test: %s" % clf.score(x_test, y_test));
    print("Accuracy On Test: %s" % accuracy_score(y_test, clf.predict(x_test)));
```

```
8285
110
```

```
0    1.0
1    1.0
2    0.0
3    1.0
4    1.0
Name: rtype_u, dtype: float64
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Unnamed: 0	PROVINCE	urban	age	gender	I1_3_1	party	army	sec_lan	f_b_hukou	...	edu	fedu	medu	jobmean	dinr
0	0.000000	0.592593	0.0	0.661972	0.0	0.464791	0.0	0.0	0.0	0.00	...	0.0	0.0	0.0	0.584615	0.16
1	0.000121	0.425926	0.0	0.436620	0.0	0.338031	0.0	0.0	0.0	0.00	...	0.0	0.0	0.0	0.584615	0.16
2	0.000241	0.740741	0.0	0.492958	1.0	0.577467	0.0	0.0	0.0	0.00	...	0.0	0.0	0.0	0.584615	0.16
3	0.000362	0.407407	1.0	0.253521	1.0	0.577467	0.0	0.0	0.0	0.00	...	0.0	0.0	0.0	0.505816	0.67
4	0.000483	0.592593	1.0	0.619718	0.0	0.464791	0.0	0.0	0.0	0.25	...	0.0	0.0	0.0	0.668630	0.22

5 rows × 108 columns

```
8284
110
```

逻辑回归

- Lasso
- LassoCV
- Logistic
- LogisticCV

```
data = []
import sklearn.linear_model as lm

print("LassoLogistic====================")
clf = lm.LogisticRegression(penalty='l1', solver='liblinear', multi_class='ovr')
clf.fit(x_train, y_train)
show_results(clf)
data.append(["LassoLogistic", accuracy_score(y_train, clf.predict(x_train)), accuracy_score(y_test, clf.predict(x_test)), accuracy_score(y, clf.predict(x))])

print("LassoLogisticCV====================")
clf = lm.LogisticRegressionCV(cv=5, penalty='l1', solver='liblinear', multi_class='ovr')
clf.fit(x_train, y_train)
show_results(clf)
data.append(["LassoLogisticCV", accuracy_score(y_train, clf.predict(x_train)), accuracy_score(y_test, clf.predict(x_test)), accuracy_score(y, clf.predict(x))])

print("Logistic====================")
clf = lm.LogisticRegression(multi_class='ovr', max_iter=1000)
clf.fit(x_train, y_train)
show_results(clf)
data.append(["Logistic", accuracy_score(y_train, clf.predict(x_train)), accuracy_score(y_test, clf.predict(x_test)), accuracy_score(y, clf.predict(x))])
```

```
print("LogisticCV====================")
clf = lm.LogisticRegressionCV(cv=5, multi_class='ovr', max_iter=1000)
clf.fit(x_train, y_train)
show_results(clf)
data.append(["LogisticCV", accuracy_score(y_train, clf.predict(x_train)), accuracy_score(y_test, clf.predict(x_test)), accuracy_score(y, clf.predict(x))])

pd.DataFrame(data, columns=['Method', 'Accuraccy on Train', 'Accuraccy on Test', 'Accuraccy on All'])
```

```
LassoLogistic====================
R^2 on Train: 0.6720761802575107
Accuracy On Train: 0.6720761802575107
R^2 on Test: 0.6542732979237084
Accuracy On Test: 0.6542732979237084
LassoLogisticCV====================
R^2 on Train: 0.6703326180257511
Accuracy On Train: 0.6703326180257511
R^2 on Test: 0.6533075808788025
Accuracy On Test: 0.6533075808788025
Logistic====================
R^2 on Train: 0.6731491416309013
Accuracy On Train: 0.6731491416309013
R^2 on Test: 0.6542732979237084
Accuracy On Test: 0.6542732979237084
LogisticCV====================
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to s
  FutureWarning)
```

```
R^2 on Train: 0.6716738197424893
Accuracy On Train: 0.6716738197424893
R^2 on Test: 0.6548768710767745
Accuracy On Test: 0.6548768710767745
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Method	Accuraccy on Train	Accuraccy on Test	Accuraccy on All
0	LassoLogistic	0.672076	0.654273	0.667109
1	LassoLogisticCV	0.670333	0.653308	0.664695
2	Logistic	0.673149	0.654273	0.667713
3	LogisticCV	0.671674	0.654877	0.666506

集成学习

- Random Forest
- Gradient Boosting
- Ada Boost
- Bagging
- KNeighbors

```
data = []
from sklearn.ensemble import RandomForestClassifier

print("Random Forest====================")
clf = RandomForestClassifier(max_features=100, n_estimators=100, max_depth=100)
clf.fit(x_train, y_train)
show_results(clf)
data.append(["Random Forest", accuracy_score(y_train, clf.predict(x_train)), accuracy_score(y_test, clf.predict(x_test)), accuracy_score(y, clf.predict(x))])

from sklearn.ensemble import GradientBoostingClassifier
```

```
print("GradientBoosting====================")
clf = GradientBoostingClassifier()
clf.fit(x_train, y_train)
show_results(clf)
data.append(["GradientBoosting", accuracy_score(y_train, clf.predict(x_train)), accuracy_score(y_test, clf.predict(x_test)), accuracy_score(y, clf.predict(x))])

from sklearn.ensemble import AdaBoostClassifier

print("AdaBoost====================")
clf = AdaBoostClassifier()
clf.fit(x_train, y_train)
show_results(clf)
from sklearn.ensemble import BaggingClassifier

print("Bagging====================")
clf = BaggingClassifier()
clf.fit(x_train, y_train)
show_results(clf)
data.append(["BaggingClassifier", accuracy_score(y_train, clf.predict(x_train)), accuracy_score(y_test, clf.predict(x_test)), accuracy_score(y, clf.predict(x))])

from sklearn.neighbors import KNeighborsClassifier

print("KNeighbors====================")
clf = KNeighborsClassifier()
clf.fit(x_train, y_train)
show_results(clf)
data.append(["KNeighbors", accuracy_score(y_train, clf.predict(x_train)), accuracy_score(y_test, clf.predict(x_test)), accuracy_score(y, clf.predict(x))])

pd.DataFrame(data, columns=['Method', 'Accuraccy on Train', 'Accuraccy on Test', 'Accuraccy on All'])
```

```
Random Forest====================
[['Random Forest', 0.9998658798283262, 0.6693626267503622, 0.9670488835244417]]
GradientBoosting====================
0.7251659625829813
accuraccy:0.6751569290
AdaBoost====================
0.6828002414001207
accuraccy:0.6641718976
Bagging====================
0.9529269764634882
accuraccy:0.6263882183
KNeighbors====================
0.7255280627640314
accuraccy:0.6151617576
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Method	Accuraccy on Train	Accuraccy on Test	Accuraccy on All
0	Random Forest	0.999866	0.669363	0.967049
1	GradientBoosting	0.731760	0.675157	0.725166
2	AdaBoost	0.686025	0.664172	0.682800
3	BaggingClassifier	0.987393	0.626388	0.952927
4	KNeighbors	0.739807	0.615162	0.725528

支持向量机 `SVM`

```
data = []
from sklearn import svm

clf = svm.SVC(gamma='scale')
clf.fit(x, y)
data.append(["SVM", accuracy_score(y_train, clf.predict(x_train)), accuracy_score(y_test, clf.predict(x_test)), accuracy_score(y, clf.predict(x))])

pd.DataFrame(data, columns=['Method', 'Accuraccy on Train', 'Accuraccy on Test', 'Accuraccy on All'])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Method	Accuraccy on Train	Accuraccy on Test	Accuraccy on All
0	SVM	0.726797	0.652342	0.723355

神经网络 (DNN)

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms


class DNN(nn.Module):
    def __init__(self):
        super(DNN, self).__init__()
        self.fc1 = nn.Linear(108, 200)
        self.fc2 = nn.Linear(200, 100)
        self.fc3 = nn.Linear(100, 2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.log_softmax(self.fc3(x))
        return x

def train(model, device, train_loader, optimizer, epoch):
    model.train()
    train_loss = 0
    correct = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        # print(output, target)
#         regularization_loss = 0
#         for param in model.parameters():
#             regularization_loss += torch.sum(abs(param))
        loss = F.nll_loss(output, target) #+ regularization_loss * 0.001

        loss.backward()
        optimizer.step()

        pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
        correct += pred.eq(target.view_as(pred)).sum().item()
        train_loss += loss.item()

#         if batch_idx % 10 == 0:
#             print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
#                 epoch, batch_idx * len(data), len(train_loader.dataset),
#                 100. * batch_idx / len(train_loader), loss.item()))

    train_loss /= len(train_loader.dataset)

    print('Train set: Average loss: {:.4f}, Accuracy: {}/{} ({:.5f}%)'.format(
        train_loss, correct, len(train_loader.dataset),
        100. * correct / len(train_loader.dataset)))
```

```
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item() # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.5f}%)'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

def main():
    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")

    batch_size = 32
    test_batch_size = 100

    x_train_tensor = torch.tensor(x_train.to_numpy(), dtype=torch.float64)
    y_train_tensor = torch.tensor([y for y in y_train.to_numpy()], dtype=torch.long)
    train_dataset = torch.utils.data.TensorDataset(x_train_tensor, y_train_tensor)
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

    x_test_tensor = torch.tensor(x_test.to_numpy(), dtype=torch.float64)
    y_test_tensor = torch.tensor([y for y in y_test.to_numpy()], dtype=torch.long)
    test_dataset = torch.utils.data.TensorDataset(x_test_tensor, y_test_tensor)
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=test_batch_size, shuffle=True)

    model = DNN().to(device)
    model.double()
    optimizer = optim.Adam(model.parameters(),lr=0.01,weight_decay=0.0001)

    epochs = 30
    print("Start Training!")
    for epoch in range(1, epochs + 1):
        print("Epoch %d Start!" % epoch)
        train(model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)

    if (True):
        torch.save(model.state_dict(),"dnn.pt")

if __name__ == '__main__':
    main()
```

```
Start Training!
Epoch 1 Start!
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:18: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to incl
```

```
Train set: Average loss: 0.0202, Accuracy: 4716/7456 (63.25107%)
Test set: Average loss: 0.7181, Accuracy: 5153/8284 (62.20425%)
Epoch 2 Start!
Train set: Average loss: 0.0199, Accuracy: 4867/7456 (65.27629%)
Test set: Average loss: 0.6427, Accuracy: 5403/8284 (65.22211%)
Epoch 3 Start!
Train set: Average loss: 0.0196, Accuracy: 4951/7456 (66.40290%)
Test set: Average loss: 0.6419, Accuracy: 5389/8284 (65.05311%)
Epoch 4 Start!
Train set: Average loss: 0.0196, Accuracy: 4946/7456 (66.33584%)
Test set: Average loss: 0.6400, Accuracy: 5331/8284 (64.35297%)
Epoch 5 Start!
Train set: Average loss: 0.0196, Accuracy: 4942/7456 (66.28219%)
Test set: Average loss: 0.6570, Accuracy: 5206/8284 (62.84404%)
Epoch 6 Start!
Train set: Average loss: 0.0195, Accuracy: 4971/7456 (66.67114%)
Test set: Average loss: 0.6501, Accuracy: 5308/8284 (64.07533%)
Epoch 7 Start!
Train set: Average loss: 0.0195, Accuracy: 4975/7456 (66.72479%)
```

```
Test set: Average loss: 0.6311, Accuracy: 5387/8284 (65.02897%)
Epoch 8 Start!
Train set: Average loss: 0.0194, Accuracy: 4968/7456 (66.63090%)
Test set: Average loss: 0.6342, Accuracy: 5390/8284 (65.06519%)
Epoch 9 Start!
Train set: Average loss: 0.0192, Accuracy: 5010/7456 (67.19421%)
Test set: Average loss: 0.6331, Accuracy: 5395/8284 (65.12554%)
Epoch 10 Start!
Train set: Average loss: 0.0191, Accuracy: 5023/7456 (67.36856%)
Test set: Average loss: 0.6331, Accuracy: 5398/8284 (65.16176%)
Epoch 11 Start!
Train set: Average loss: 0.0190, Accuracy: 5056/7456 (67.81116%)
Test set: Average loss: 0.6297, Accuracy: 5413/8284 (65.34283%)
Epoch 12 Start!
Train set: Average loss: 0.0190, Accuracy: 5058/7456 (67.83798%)
Test set: Average loss: 0.6276, Accuracy: 5445/8284 (65.72912%)
Epoch 13 Start!
Train set: Average loss: 0.0192, Accuracy: 4967/7456 (66.61749%)
Test set: Average loss: 0.6379, Accuracy: 5355/8284 (64.64268%)
Epoch 14 Start!
Train set: Average loss: 0.0191, Accuracy: 5057/7456 (67.82457%)
Test set: Average loss: 0.6285, Accuracy: 5452/8284 (65.81362%)
Epoch 15 Start!
Train set: Average loss: 0.0191, Accuracy: 5032/7456 (67.48927%)
Test set: Average loss: 0.6295, Accuracy: 5401/8284 (65.19797%)
Epoch 16 Start!
Train set: Average loss: 0.0189, Accuracy: 5070/7456 (67.99893%)
Test set: Average loss: 0.6338, Accuracy: 5378/8284 (64.92033%)
Epoch 17 Start!
Train set: Average loss: 0.0189, Accuracy: 5083/7456 (68.17328%)
Test set: Average loss: 0.6322, Accuracy: 5411/8284 (65.31869%)
Epoch 18 Start!
Train set: Average loss: 0.0188, Accuracy: 5098/7456 (68.37446%)
Test set: Average loss: 0.6325, Accuracy: 5403/8284 (65.22211%)
Epoch 19 Start!
Train set: Average loss: 0.0188, Accuracy: 5084/7456 (68.18670%)
Test set: Average loss: 0.6310, Accuracy: 5427/8284 (65.51183%)
Epoch 20 Start!
Train set: Average loss: 0.0190, Accuracy: 5099/7456 (68.38788%)
Test set: Average loss: 0.6312, Accuracy: 5420/8284 (65.42733%)
Epoch 21 Start!
Train set: Average loss: 0.0188, Accuracy: 5126/7456 (68.75000%)
Test set: Average loss: 0.6282, Accuracy: 5404/8284 (65.23419%)
Epoch 22 Start!
Train set: Average loss: 0.0189, Accuracy: 5085/7456 (68.20011%)
Test set: Average loss: 0.6429, Accuracy: 5392/8284 (65.08933%)
Epoch 23 Start!
Train set: Average loss: 0.0188, Accuracy: 5110/7456 (68.53541%)
Test set: Average loss: 0.6352, Accuracy: 5391/8284 (65.07726%)
Epoch 24 Start!
Train set: Average loss: 0.0188, Accuracy: 5135/7456 (68.87071%)
Test set: Average loss: 0.6293, Accuracy: 5387/8284 (65.02897%)
Epoch 25 Start!
Train set: Average loss: 0.0188, Accuracy: 5119/7456 (68.65612%)
Test set: Average loss: 0.6380, Accuracy: 5395/8284 (65.12554%)
Epoch 26 Start!
Train set: Average loss: 0.0187, Accuracy: 5113/7456 (68.57564%)
Test set: Average loss: 0.6572, Accuracy: 5365/8284 (64.76340%)
Epoch 27 Start!
Train set: Average loss: 0.0187, Accuracy: 5118/7456 (68.64270%)
Test set: Average loss: 0.6392, Accuracy: 5403/8284 (65.22211%)
Epoch 28 Start!
Train set: Average loss: 0.0187, Accuracy: 5134/7456 (68.85730%)
Test set: Average loss: 0.6541, Accuracy: 5399/8284 (65.17383%)
Epoch 29 Start!
Train set: Average loss: 0.0187, Accuracy: 5078/7456 (68.10622%)
Test set: Average loss: 0.6364, Accuracy: 5382/8284 (64.96861%)
Epoch 30 Start!
Train set: Average loss: 0.0187, Accuracy: 5161/7456 (69.21942%)
Test set: Average loss: 0.6473, Accuracy: 5385/8284 (65.00483%)
```