U.S. Census of Income,1994

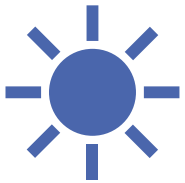
DATA MINING

Group 10

1. 李文锐 2030005036
2. 蓝一凯 2030005032
3. 叶中宇 2030005069
4. 王一安 2030005058
5. 陈俊霖 2030005004

1. 李文锐 2030005036

Introduction



- Adult dataset: “Census Income” data set from the 1994 U.S. Census database.
- 48842 observations; training data: 32561, test data: 16281.
- 23.93% of the total annual income is more than 50k $, and 76.07% of the total annual income is less than 50k $
- Purpose: forecast whether the annual income exceeds 50k$
- Class: whether the annual income exceeds 50k $

Introduction

- Basic Properties: 14 categorical variables
- Attribute: age, workclass, fnlwgt, education, education-num, marital status,…
  - Properties: discrete & contiunous

Variable	type
age	contiunous
workclass	discrete
Fnlwgt	continuous
education	discrete
Education-num	Continuous
Marital-status	Discrete
occupation	discrete

Variable	type
relationship	discrete
race	discrete
sex	discrete
Capital-gain	continuous
Capital-loss	continuous
Hours per week	continuous
Native-country	discrete

Data Preprocess

- Missing Value: filled by mode of this category
- Replace the income by using number(<=50 -->0 & >50 --> 1)
- Cancel `Capital-gain` and `Capital-loss` two categories, because there are more than 75% of people do not have them, they are irrelevant categories.

Data Preprocess

Missing Value

```
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', None)
# PS. we need to install sklearn by using command `pip install scikit-learn==0.22`
df = pd.read_csv('adult.csv', header=None,
                 names=['age', 'workclass', 'fnlwgt', 'education', 'education-num',
                        'relationship', 'race', 'sex', 'capital-gain', 'capital-loss
                        'native-country', 'income'])
df.head()
df.info()  ###
```

- First, directly search data for null values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64
 5   marital-status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64
 11  capital-loss    32561 non-null  int64
 12  hours-per-week  32561 non-null  int64
 13  native-country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Data Preprocess
Missing Value

```
df.apply(lambda x: np.sum(x == ""))
```

- We found that some missing values in the original data were '? '. So we continue to search for '? 'data.

```
memory usage: 3.7+ MB
age                0
workclass       1836
fnlwgt             0
education          0
education-num      0
marital-status     0
occupation      1843
relationship       0
race               0
sex                0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country   583
income             0
dtype: int64
```

```
age                0
workclass          0
fnlwgt             0
education          0
education-num      0
marital-status     0
occupation         0
relationship       0
race               0
sex                0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income             0
dtype: int64
```

Data Preprocess

Income Replacement

```
df.replace(" ?", pd.NaT, inplace=True)
df.replace(" >50K", 1, inplace=True)  # substitude >50K by 1
df.replace(" <=50K", 0, inplace=True)  # substitude <=50K by 1
trans = {'workclass': df['workclass'].mode()[0], 'occupation': df['occupation'].mode()[0],
         'native-country': df['native-country'].mode()[0]}
df.fillna(trans, inplace=True)
f = open("a.txt","w")
f.write(str(df.describe()))  ###
f.close()
```

- Revenue can be divided into two types, which we have simplified. Replace >50K with 1, and <=50K with 0.

Data Preprocess

Cancel Irrelevant Attributes

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	income
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456	0.240810
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429	0.427581
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000	0.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000	0.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000	0.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000	0.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000	1.000000

- As can be seen from the above table, std of capital-gain and capital-loss are very huge which will cause the effct of classification badly. And more than 75% people have no capital-gain and capital-loss., so the 2 attributes cannot represent the majority. We delete them as we considered them irrelevant.

Data Preprocess

Cancel Irrelevant Attributes

```
df.drop('fnlwgt', axis=1, inplace=True)  # cancel the num
df.drop('capital-gain', axis=1, inplace=True)
df.drop('capital-loss', axis=1, inplace=True)
df.head()
```

age	workclass	education	education-num	marital-status	occupation	relationship	race	sex	hours-per-week	native-country	income
39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	40	United-States	0
50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	13	United-States	0
38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	40	United-States	0
53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	40	United-States	0
28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	40	Cuba	0

- The result after removing the irrelevant attribute.

Common code illustration

```
def Adult_data():
    df_train = pd.read_csv('adult.csv', header=None,
                           names=['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
                                  'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
                                  'hours-per-week', 'native-country', 'income'])
    df_test = pd.read_csv('adult.test', header=None, skiprows=1,
                          names=['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
                                 'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
                                 'hours-per-week', 'native-country', 'income'])

    train_target, train_dataset = data_process(df_train, 'train')
    test_target, test_dataset = data_process(df_test, 'test')
    #         进行 One-Hot Encoding 对齐
    test_dataset = fix_columns(test_dataset, train_dataset.columns)
    columns = train_dataset.columns
    #         print(df["income"])

    train_target, test_target = np.array(train_target), np.array(test_target)
    train_dataset, test_dataset = np.array(train_dataset), np.array(test_dataset)

    return train_dataset, train_target, test_dataset, test_target, columns
```

Common code illustration

```
def data_process(df, model):
    df.replace(" ?", pd.NaT, inplace=True)
    if model == 'train':
        df.replace(" >50K", 1, inplace=True)
        df.replace(" <=50K", 0, inplace=True)
    if model == 'test':
        df.replace(" >50K.", 1, inplace=True)
        df.replace(" <=50K.", 0, inplace=True)
    trans = {'workclass': df['workclass'].mode()[0], 'occupation': df['occupation'].mode()[0],
             'native-country': df['native-country'].mode()[0]}
    df.fillna(trans, inplace=True)

    df.drop('fnlwgt', axis=1, inplace=True)
    df.drop('capital-gain', axis=1, inplace=True)
    df.drop('capital-loss', axis=1, inplace=True)
    #         print(df)

    df_object_col = [col for col in df.columns if df[col].dtype.name == 'object']
    df_int_col = [col for col in df.columns if df[col].dtype.name != 'object' and col != 'income']
    target = df["income"]
    dataset = pd.concat([df[df_int_col], pd.get_dummies(df[df_object_col])], axis=1)

    return target, dataset
```

```
def add_missing_columns(d, columns):
    missing_col = set(columns) - set(d.columns)
    for col in missing_col:
        d[col] = 0

    def fix_columns(d, columns):
        add_missing_columns(d, columns)
        assert (set(columns) - set(d.columns) == set())
        d = d[columns]

        return d
```

Concept: One-hot encoding

Convert string to One-hot encoding:

1. save the stroage

2. easier to show the label of category for each node

Decimal	Binary	Unary	One-hot
0	000	00000000	00000001
1	001	00000001	00000010
2	010	00000011	00000100
3	011	00000111	00001000
4	100	00001111	00010000
5	101	00011111	00100000
6	110	00111111	01000000
7	111	01111111	10000000

Model 1: Construction – Decision Tree

- Use `GridSearchCV` to determine what is the best depth for decision tree.

- Use entropy as an criteria to determine whether this node needs to be splited.

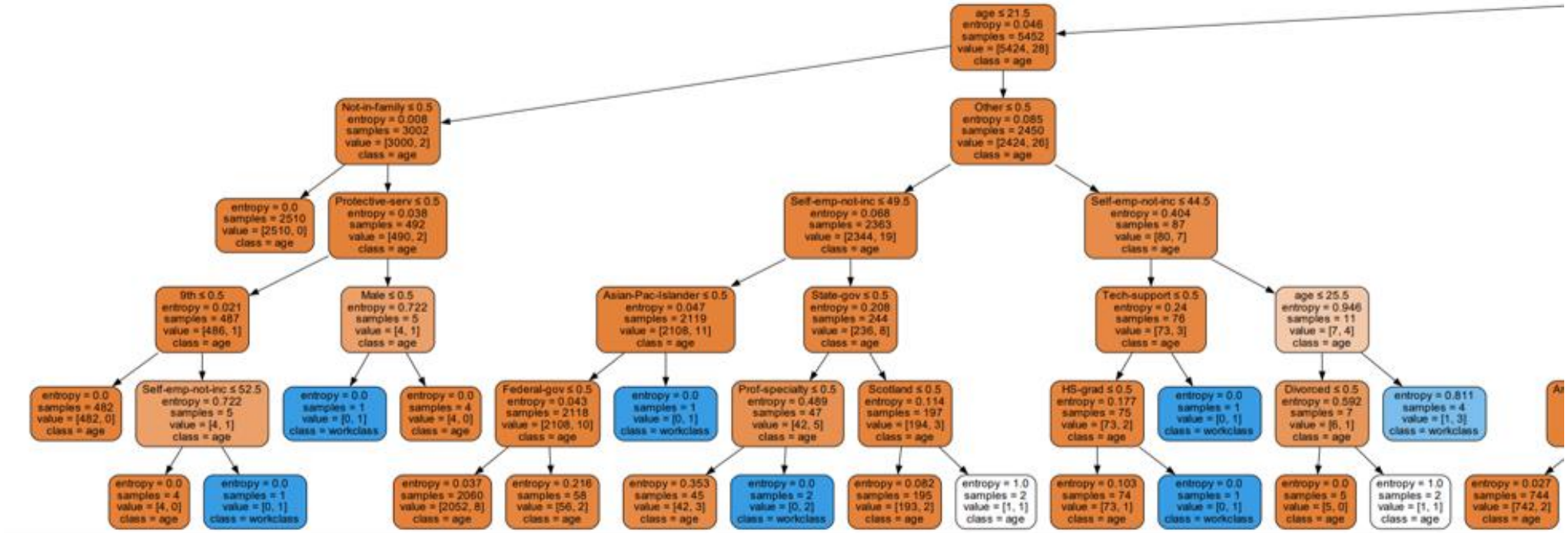
```
params = {'max_depth': range(1, 20)}
best_clf = GridSearchCV(DecisionTreeClassifier(criterion='entropy', random_state=20), param_grid=params,cv=10)  #
best_clf = best_clf.fit(train_dataset, train_target)
print(best_clf.best_params_)
```

```
{'max_depth': 8}
```

```
classes = [' <=50K', ' >50K']
clf = DecisionTreeClassifier(criterion='entropy', max_depth=8, min_samples_split=5)
clf = clf.fit(train_dataset, train_target)
text_representation = tree.export_text(clf)
plot = tree.export_graphviz(clf)
print("================================================================================")
print(text_representation)
print("================================================================================")
pred = clf.predict(test_dataset)
```

```
0.8329955162459308
```

Tree Plot (Part of)



Model 2: Random Forests

```
classes = [' <=50K', ' >50K']
rf = RandomForestClassifier(n_estimators=100, random_state=0)
# n_estimators: the number of trees in the forest
# Controls both the randomness of the bootstrapping of the samples used when building trees.
rf = rf.fit(train_dataset, train_target)
score = rf.score(test_dataset, test_target)
print(score)
print("=========================================================")
print(rf.feature_importances_)
print("=========================================================")
pred = rf.predict(test_dataset)
print(pred)
```

```
0.8143848657944843
```

```
[2.86075787e-01 7.51213232e-02 1.42203615e-01 6.11774633e-03
 7.06819613e-03 6.71202229e-06 1.35865269e-02 7.73836006e-03
 1.04360331e-02 5.96785534e-03 9.70005938e-05 1.98134526e-03
 2.65307046e-03 1.19601579e-03 3.49768581e-04 9.92821802e-04
 2.64331128e-03 1.67318914e-03 3.09920088e-03 3.52919419e-03
 1.61995623e-02 4.55729016e-03 8.87505742e-03 9.93404277e-03
 6.15555606e-05 7.59276641e-03 6.23611274e-03 8.80472907e-03
 5.02194996e-04 6.61993808e-02 1.28945163e-03 2.76671591e-02
 2.41214603e-03 2.32255879e-03 7.05463691e-03 1.70645918e-05
 8.31298931e-03 2.44767153e-02 5.58346402e-03 3.76609978e-03
 5.02601828e-03 8.72513482e-03 2.48565039e-04 1.31999947e-02
 3.82333135e-03 8.53267621e-03 5.71726593e-03 5.70725933e-03
 5.72879798e-02 1.10599666e-02 1.82155834e-03 7.97096927e-03
 5.99666240e-03 1.21626999e-02 2.06527414e-03 3.94481406e-03
 6.45561324e-03 1.26792423e-03 8.40086831e-03 9.67787202e-03
 8.71502526e-03 3.42808170e-04 1.27332408e-03 6.78605044e-04
 2.95119892e-04 9.04875788e-04 3.01348405e-04 2.49095338e-04
 3.85201149e-04 1.26899401e-03 4.36079105e-04 1.40309240e-03
 4.98316992e-04 1.49403947e-04 1.98921249e-04 0.00000000e+00
 6.76810437e-06 2.06915588e-04 1.95051656e-04 8.45988728e-04
 5.40493690e-04 2.44874734e-04 9.16098789e-04 5.54002848e-04
 6.84338770e-04 1.07916592e-04 2.40031967e-03 1.99723756e-04
 2.34889787e-05 1.54782400e-04 1.29312533e-03 7.14203778e-04
 2.20613587e-04 6.77279483e-04 1.18533211e-04 7.91611973e-04
 3.59278629e-04 8.53183011e-05 1.34663498e-04 7.06560317e-03
 4.82242139e-04 3.82053717e-04]
```

Model 3: Construction -- SVM

```
classes = [' <=50K', ' >50K']
clf = svm.SVC(kernel='linear')
clf = clf.fit(train_dataset, train_target)
pred = clf.predict(test_dataset)
score = clf.score(test_dataset, test_target)
print(score)
print(pred)
```

```
[6263 6238]
```

Support vector: 3780

Accurancy:

```
0.8344082058841594
```

Parameters:

```
[[ 0.01289082  0.18028072  0.01935213  0.43806931  0.0799317  -0.29279691
   0.09346641  0.4416909  -0.16516767 -0.08716725 -0.5080265  -0.13175585
  -0.22155213 -0.38263085  0.38964363  0.31862295  0.06126749 -0.01097753
  -0.1728317  -0.11145631  0.19487694  0.18391761 -0.37523857  0.18187064
   0.          0.30996568 -0.23372199 -0.27661621  0.79056899  0.84847202
  -0.34170945 -0.60665888 -0.33654205 -0.07751442  0.09343634  0.
  -0.06058957  0.61501388 -0.65428573 -0.29286018 -0.1582153  -0.39846435
  -0.39609887  0.26169148  0.25797664  0.28627798  0.58427462 -0.13815693
   0.06705388 -0.05222645 -0.29503011 -0.31005248 -0.1296148  | 0.71986997
  -0.28962069  0.1377909   0.13987363 -0.15501595  0.1669721  -0.20387966
   0.20387966  1.0463817   0.39801599 -0.02226602 -1.          0.24949207
  -0.87503818 -0.03593274 -0.0230797   0.24414742  0.50576012  0.40787526
  -0.28077606  0.          -0.00356117  0.          0.          0.0421323
  -0.23607215  0.0357988   0.15361663  0.27295533  0.5065688  -0.03740751
   0.43009548 -0.20021339 -0.00958064 -0.59703675 -0.52562921 -0.404511
   0.45055089  0.12424803  0.          -0.03795715  0.02423269 -0.52179134
   0.20338478 -0.15878502 -0.02720179  0.22810659 -0.52246839  0.19594535]]
```

Model 4: Deep Learning (Neuron Network)

- Construct the neuron network, we used network with two layers
- Define the times of trainning, after each iteration, keep saving the best model.

```
# 构建网络，因为是简单的二分类，这里使用了两层感知机网络，后面做对结果进行softmax归一化。

class Adult_Model(nn.Module):
    def __init__(self):
        super(Adult_Model, self).__init__()
        self.net = nn.Sequential(nn.Linear(102, 64),
                                 nn.ReLU(),
                                 nn.Linear(64, 32),
                                 nn.ReLU(),
                                 nn.Linear(32, 2)
                                 )

    def forward(self, x):
        out = self.net(x)
        #         print(out)
        return F.softmax(out)
```

```
#训练及验证，每经过一个epoch，就进行一次损失比较，当val_loss更小时，保存最好模型，直至迭代结束。

device = torch.device('cuda' if torch.cuda.is_available() else "cpu")
model = Adult_Model().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
criterion = nn.CrossEntropyLoss()
max_epoch = 30
classes = [' <=50K', ' >50K']
mse_loss = 1000000
os.makedirs('MyModels', exist_ok=True)
writer = SummaryWriter(log_dir='logs')
```

Model 4: Deep Learning (Neuron Network)

- Trainning the model

```
for epoch in range(max_epoch):

    train_loss = 0.0
    train_acc = 0.0
    model.train()
    for x, label in train_loader:
        x, label = x.to(device), label.to(device)
        optimizer.zero_grad()

        out = model(x)
        loss = criterion(out, label)
        train_loss += loss.item()
        loss.backward()

        _, pred = torch.max(out, 1)
        #         print(pred)
        num_correct = (pred == label).sum().item()
        acc = num_correct / x.shape[0]
        train_acc += acc
        optimizer.step()

    print(f'epoch : {epoch + 1}, train_loss : {train_loss / len(train_loader.dataset)}, train_acc : {train_acc / len(train_loader)}')
    writer.add_scalar('train_loss', train_loss / len(train_loader.dataset), epoch)
```

Model 4: Deep Learning (Neuron Network)

- Trainning the model

```
    with torch.no_grad():
        total_loss = []
        model.eval()
        for x, label in val_loader:
            x, label = x.to(device), label.to(device)
        out = model(x)
        loss = criterion(out, label)
        total_loss.append(loss.item())

        val_loss = sum(total_loss) / len(total_loss)

    if val_loss < mse_loss:
        mse_loss = val_loss
        torch.save(model.state_dict(), 'MyModels/Deeplearning_Model.pth')
del model
```

```
epoch : 1, train_loss : 0.009711737515683988, train_acc : 0.7438574938574939
epoch : 2, train_loss : 0.008743632740412063, train_acc : 0.7604038697788698
epoch : 3, train_loss : 0.008471577871059977, train_acc : 0.7604038697788698
epoch : 4, train_loss : 0.008271507909681638, train_acc : 0.7604038697788698
epoch : 5, train_loss : 0.008061235772871414, train_acc : 0.762323402948403
epoch : 6, train_loss : 0.007855728706825497, train_acc : 0.7919993857493858
epoch : 7, train_loss : 0.007671554991632187 5, train_acc : 0.8221360565110565
epoch : 8, train_loss : 0.00754701426705798 6, train_acc : 0.8279714373464373
epoch : 9, train_loss : 0.007471159839641111, train_acc : 0.8304284398034398
epoch : 10, train_loss : 0.007422603463127074, train_acc : 0.831656941031941
epoch : 11, train_loss : 0.007388273845552592, train_acc : 0.8332693488943489
epoch : 12, train_loss : 0.007364053037749827, train_acc : 0.8350353194103194
epoch : 13, train_loss : 0.007345126942944248, train_acc : 0.835764742014742
epoch : 14, train_loss : 0.007330227416713495, train_acc : 0.8360334766584766
epoch : 15, train_loss : 0.00731822161727038, train_acc : 0.8364941646191646
epoch : 16, train_loss : 0.007307886268430977 5, train_acc : 0.8365709459459459
epoch : 17, train_loss : 0.007299667121316204, train_acc : 0.8369548525798526
epoch : 18, train_loss : 0.007292353091601291, train_acc : 0.8375307125307125
epoch : 19, train_loss : 0.007286161418857753, train_acc : 0.8381833538083538
epoch : 20, train_loss : 0.007280141031667041, train_acc : 0.8379914004914005
epoch : 21, train_loss : 0.007275205067786816, train_acc : 0.8386440417690417
epoch : 22, train_loss : 0.007270320769426864, train_acc : 0.8391047297297297
epoch : 23, train_loss : 0.007265850135035978, train_acc : 0.8390663390663391
epoch : 24, train_loss : 0.007261692232816167, train_acc : 0.8391431203931204
```

Model 4: Deep Learning (Neuron Network)

- Trainning the model

```
    with torch.no_grad():
        total_loss = []
        model.eval()
        for x, label in val_loader:
            x, label = x.to(device), label.to(device)
        out = model(x)
        loss = criterion(out, label)
        total_loss.append(loss.item())

        val_loss = sum(total_loss) / len(total_loss)

    if val_loss < mse_loss:
        mse_loss = val_loss
        torch.save(model.state_dict(), 'MyModels/Deeplearning_Model.pth')
del model
```

```
test_loss : 0.006785568750707855, test_acc : 0.7693627450980393
test_loss : 0.006819542482981305, test_acc : 0.7721813725490196
test_loss : 0.00684997151630325, test_acc : 0.7751838235294117
test_loss : 0.006877160280637224, test_acc : 0.7785539215686275
test_loss : 0.006904503048348578, test_acc : 0.7819852941176471
test_loss : 0.006936014165354627 5, test_acc : 0.7850490196078431
test_loss : 0.00696547673650781, test_acc : 0.788296568627451
test_loss : 0.006990641137685378, test_acc : 0.7919117647058823
test_loss : 0.007018043336144288, test_acc : 0.7953431372549019
test_loss : 0.007046293940695547, test_acc : 0.7985906862745098
test_loss : 0.007070657283417328, test_acc : 0.8022058823529412
test_loss : 0.007098812248158987, test_acc : 0.8055759803921568
test_loss : 0.0071277726590578475, test_acc : 0.8088235294117647
test_loss : 0.007157285682370582, test_acc : 0.8120098039215686
test_loss : 0.007185506052602151, test_acc : 0.8153799019607844
test_loss : 0.007214113972007068, test_acc : 0.8186887254901961
test_loss : 0.007238068187656336, test_acc : 0.8223651960784314
```

Model 4: Deep Learning (Neuron Network)

- Fetch the best model from CPU

```
best_model = Adult_Model().to(device)
ckpt = torch.load('MyModels/Deeplearning_Model.pth', map_location='cpu')
best_model.load_state_dict(ckpt)
test_loss = 0.0
test_acc = 0.0
best_model.eval()
result = []
```

Model 4: Deep Learning (Neuron Network)

- Test Result

```
test_loss : 0.007404681106844095, test_acc : 0.831154411764706
```

5. 陈俊霖 2030005004

Conclusion: model comparison

According to the prediction results in the test set of the Adult dataset, the accuracy of the deep learning model, decision tree, support vector machine and random forest is 0.831, 0.833, 0.834 and 0.814 respectively, and the accuracy of the four models is similar.

Reasons for low accuracy…	Resolvent…
✓ The robustness of the model is not enough. ✓ There is a large number of discrete data in the dataset, and the data is highly sparse after the unique heat coding.	➢ To adjust the parameters of the model searchingly, you can consider increasing the complexity of the model, and pay attention to over fitting in the process, make sure you are completing your class job, if you have one ➢ Do not select the method of exclusive hot coding to reduce the dimension of data, and you can consider embedding

Thank you for listening!

Code: https://github.com/g20021215/Data-Mining-2022
Dataset: https://archive.ics.uci.edu/ml/machine-learning-databases/adult/