Game 1：预测 2012 年参加调查的样本在 2014 年是否追踪成功

导入数据

1. 导入数据并查看
2. 查看数据标签
3. 导入处理后数值数据

```
import pandas as pd
from IPython import display

df = pd.read_stata('game1_train_data.dta')
df.to_csv('game1_train_data.csv', encoding="utf_8")

df.head()
```

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

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

	PROVINCE	urban	age	gender	I1_3_1	party	army	sec_lan	f_b_hukou	f_n_hukou	...	fedu	medu	jobmea
0	广东	0	27.0	0	广东	0	0	0	农业户口	农业户口	...	1.0	1.0	-0.94632
1	江西	农村	50.0	0	江西	0	0	0	农业户口	农业户口	...	1.0	1.0	NaN
2	安徽	0	45.0	男	安徽	0	0	0	农业户口	农业户口	...	1.0	1.0	0.18041
3	湖北	0	59.0	男	湖北	0	0	0	农业户口	不适用	...	2.0	1.0	0.78997
4	重庆	农村	54.0	0	重庆	0	0	0	农业户口	不适用	...	1.0	1.0	0.78997

5 rows × 109 columns

```
import json
itr = pd.read_stata('game1_train_data.dta', iterator=True)
df.head()
vl = itr.variable_labels()
print(json.dumps(vl, indent=4, ensure_ascii=False))
```


```
{
    "PROVINCE": "省/自治区/直辖市编码",
    "urban": "调查地点的社区类型",
    "age": "被访者年龄",
    "gender": "被访者性别",
    "I1_3_1": "被访者出生地省/自治区/直辖市编码",
    "party": "被访者政治面貌",
    "army": "被访者是否参过军",
    "sec_lan": "被访者是否懂外语",
    "f_b_hukou": "被访者的父亲出生时的户口性质",
    "f_n_hukou": "被访者的父亲现在的户口性质",
    "m_b_hukou": "被访者的母亲出生时的户口性质",
    "m_n_hukou": "被访者的母亲现在的户口性质",
    "b_hukou": "被访者出生时的户口性质",
    "n_hukou": "被访者现在的户口性质",
    "local_hukou": "被访者户口是否在本地",
    "migrant_hukou": "被访者户口是否迁移过",
    "cungaiju": "被访者的户口性质是否因为村改居发生过变动",
    "u_w_m": "被访者目前是否有城镇职工基本医疗保险",
    "u_c_m": "被访者目前是否有城镇居民基本医疗保险",
    "r_m": "被访者目前是否有新型农村合作医疗",
    "p_m": "被访者目前是否有公费医疗",
    "uint_add_m": "被访者目前是否有单位补充医疗保险",
    "br_m": "被访者目前是否有公务员医疗补助",
```

```
    "bus_m": "被访者目前是否有商业医疗保险",
    "I1_20_1": "被访者目前是否有企业职工基本养老保险",
    "I1_20_2": "被访者目前是否有城镇居民社会养老保险",
    "n_r_o_i": "被访者目前是否有新型农村社会养老保险",
    "I1_20_4": "被访者目前是否有企业年金（企业补充养老保险）",
    "I1_20_5": "被访者目前是否有商业性养老保险",
    "I1_21": "被访者目前是否有住房公积金",
    "I1_22_1": "被访者目前是否有工伤保险",
    "I1_22_2": "被访者目前是否有生育保险",
    "I1_22_3": "被访者目前是否有失业保险",
    "tech_edu": "被访者在过去2年里是否参加过至少5天的专业技术培训",
    "I2_10": "被访者是否获得过专业技术资格证书（执业资格）",
    "work_exp": "被访者是否有工作经历",
    "I3a1_1": "[雇员]被访者是否有固定雇主",
    "I3a1_2": "[雇员]被访者现在的工作是否为自己的家庭/家族企业/公司工作",
    "I3a1_5": "[雇员]被访者目前工作是否签订书面劳动合同",
    "I3a1_9": "[雇员]被访者是否认为做好当前的工作需要接受专门的训练或培训",
    "I3a1_14": "[雇员]被访者在过去一个月，是否加班过",
    "I3a1_16_1": "[雇员]被访者在工作中，工作任务的内容在多大程度上由自己决定",
    "I3a1_16_2": "[雇员]被访者在工作中，工作进度的安排在多大程度上由自己决定",
    "I3a1_16_3": "[雇员]被访者在工作中，工作量和工作强度在多大程度上由自己决定",
    "I3a1_19_1": "[雇员]被访者在工作过程中，是否需要频繁的体力劳动",
    "I3a1_19_2": "[雇员]被访者在工作过程中，是否需要快速而频繁地移动身体的位置",
    "I3a1_19_3": "[雇员]被访者在工作过程中，是否需要快速反应的思考或脑力劳动",
    "I3a1_20_1": "[雇员]被访者在工作中，与顾客/服务对象打交道的频繁程度",
    "I3a1_20_2": "[雇员]被访者在工作中，与客户/供应商打交道的频繁程度",
    "I3a1_20_3": "[雇员]被访者在工作中，与各种来客打交道的频繁程度",
    "I3a1_20_4": "[雇员]被访者在工作中，与上级领导打交道的频繁程度",
    "I3a1_20_5": "[雇员]被访者在工作中，与下级同事打交道的频繁程度",
    "I3a1_20_7": "[雇员]被访者在工作中，与上级部门/单位打交道的频繁程度",
    "I3a1_20_8": "[雇员]被访者在工作中，与下级部门/单位打交道的频繁程度",
    "I3a1_20_9": "[雇员]被访者在工作中，与其他单位打交道的频繁程度",
    "I3a1_21": "[雇员]被访者上周有没有去上班",
    "I3a2_1": "[雇主]被访者是什么时候开业/做生意的",
    "friend": "在本地，被访者有多少关系密切，可以得到支持和帮助的朋友/熟人",
    "psychofriend": "在本地这些关系密切的人中，被访者可以向他/她诉说心事的有几个",
    "discussfriend": "在本地这些关系密切的人中，被访者可以同他/她讨论重要问题的有几个",
    "moneyfriend": "在本地这些关系密切的人中，被访者可以向他/她借钱（5000元为标准）的有几个",
    "memberelect": "在本村/居委会上次的选举中，被访者如何参与",
    "happy": "总的来说，被访者认为生活是否过得幸福",
    "happycp": "被访者觉得与大多数同龄人相比，幸福感如何",
    "I7_7_1": "在过去四周里，被访者是否经常有以下的感受或想法-我觉得自己不能控制生活中的重要事",
    "I7_7_2": "在过去四周里，被访者是否经常有以下的感受或想法-我觉得有信心处理好自己的问题",
    "I7_9": "被访者认为目前的生活水平和自己的努力比起来是否公平",
    "soccls": "被访者认为自己目前在哪个等级上",
    "b_soccls": "被访者认为自己5年前在哪个等级上",
    "I7_10_3_1": "被访者认为自己5年后将会在哪个等级上",
    "t_soccls": "被访者认为在自己14岁时，家庭处在哪个等级上",
    "soctrs": "总的来说，被访者是否同意“大多数人是可以信任的”这种看法",
    "I9_1": "被访者的身高（cm）",
    "I9_2": "被访者的体重（斤）",
    "arm": "被访者的双臂伸展长度（cm）",
    "health": "被访者认为自己现在的健康状况如何",
    "I9_5": "在过去一个月内，是否由于身体健康问题影响到被访者的工作或其他日常活动",
    "I9_6": "在过去一个月内，是否由于情绪问题(如感到沮丧或焦虑)影响到被访者的工作或其他日常活",
    "ill": "被访者过去两周是否生病",
    "I9_8": "过去12月被访者是否住过院",
    "I9_9": "过去12个月，因病休工/休学天数",
    "I9_10": "被访者在多大程度上信任中医",
    "I9_11": "被访者在多大程度上信任西医",
    "I9_12": "被访者是否有吸烟历史",
    "I9_14": "被访者平时是否喝酒?",
    "interviewway": "在正式访问的时候，这份问卷是如何填答",
    "reject": "在访问过程中，被访者有没有表示过拒绝受访的意思呢",
    "impat": "在访问过程中，被访者是否表示不耐烦呢",
    "itviewer_trust": "在访问过程中，被访者对访问员的信任程度如何",
    "fudge": "在访问过程中，被访者是否应付",
    "colla": "被访者合作程度如何",
    "rely": "这份问卷访问所得的可靠程度如何",
    "lang": "访问时所用的语言",
    "sinitviewer": "访问时访问员是否单独作业",
    "look": "访问员觉得被访者的长相怎样（长相越好，评分越高）",
    "sinitviewee": "访问时，有其他人在场吗-没其他人在场",
    "rtype_u": "是否追踪成功",
    "belief": "有无宗教信仰",
    "edu": "受教育年限",
    "fedu": "父亲受教育年限",
    "medu": "母亲受教育年限",
    "jobmean": "工作满意度",
    "dinner": "在外就餐情况",
```

```
    "socresp": "对待困难的态度",
    "localsc": "在本地是否经历过被殴打、偷窃、抢劫、恐吓、诈骗等事件",
    "comlk": "社区融入情况",
    "plan": "是否有离开现在工作的打算",
    "u": "",
    "d": ""
}
```

```
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	I1_3_1	party	army	sec_lan	f_b_hukou	...	fedu	medu	jobmean	dinner
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	0.500000	0.250000
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	0.662233	0.106856
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	0.662233	0.000000
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	0.750000	0.000000
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	0.750000	0.304420

5 rows × 110 columns

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

准备工作

1. 分为测试集与训练集
2. 准备结果展示函数

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

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

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)));
```

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: 0.6723444206008584
Accuracy On Train: 0.6723444206008584
R^2 on Test: 0.6224366706875754
Accuracy On Test: 0.6224366706875754
LassoLogisticCV====================
R^2 on Train: 0.6703326180257511
Accuracy On Train: 0.6703326180257511
R^2 on Test: 0.6139927623642943
Accuracy On Test: 0.6139927623642943
Logistic====================
R^2 on Train: 0.6731491416309013
Accuracy On Train: 0.6731491416309013
R^2 on Test: 0.6188178528347407
Accuracy On Test: 0.6188178528347407
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: 0.6200241254523522
Accuracy On Test: 0.6200241254523522
```

```
.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	0.672344	0.622437	0.667351
1	LassoLogisticCV	0.670333	0.613993	0.664695
2	Logistic	0.673149	0.618818	0.667713
3	LogisticCV	0.671674	0.620024	0.666506

集成学习

- 随机森林 `Random Forest`
- 梯度提升决策树 `Gradient Boosting`
- 自适应增强 `Ada Boost`
- 引导聚集 `Bagging`
- K-近邻 `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====================
R^2 on Train: 1.0
Accuracy On Train: 1.0
R^2 on Test: 0.6513872135102533
Accuracy On Test: 0.6513872135102533
GradientBoosting====================
R^2 on Train: 0.7317596566523605
Accuracy On Train: 0.7317596566523605
R^2 on Test: 0.6658624849215923
Accuracy On Test: 0.6658624849215923
AdaBoost====================
R^2 on Train: 0.686024678111588
Accuracy On Train: 0.686024678111588
R^2 on Test: 0.6537997587454765
Accuracy On Test: 0.6537997587454765
Bagging====================
R^2 on Train: 0.9885997854077253
```

```
Accuracy On Train: 0.9885997854077253
R^2 on Test: 0.6501809408926418
Accuracy On Test: 0.6501809408926418
KNeighbors====================
R^2 on Train: 0.7398068669527897
Accuracy On Train: 0.7398068669527897
R^2 on Test: 0.5971049457177322
Accuracy On Test: 0.5971049457177322
```

```
.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	1.000000	0.651387	0.965118
1	GradientBoosting	0.731760	0.665862	0.725166
2	BaggingClassifier	0.988600	0.650181	0.954737
3	KNeighbors	0.739807	0.597105	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	0.6924	0.723355

深层神经网络 `DNN`

总计 4 层，隐藏层 2 层的深层神经网络

- 第一层：输入层 （节点数 108）
- 第二层：隐藏层 （节点数 200）
- 第二层：隐藏层 （节点数 100）
- 第二层：输出层 （节点数 2）
- 激活函数： `RELU`

f(x) = \begin{cases} x && (x > 0) \\ 0 && (x < 0) \end{cases}

- 正则化： `L2` 惩罚函数
    - `L0` 非零的个数
    - `L1` 参数绝对值的和
    - `L2` 参数平方和

```
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)
        loss = F.nll_loss(output, target)

        loss.backward()
        optimizer.step()

        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()
        train_loss += loss.item()

    train_loss /= len(train_loader.dataset)

    print('Train set: Average loss: %.4f, Accuracy: %.6f %.5f' \
          % (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: %.6f %.5f' \
          % (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: 0.0205, Accuracy: 0.621111 62.11105
Test set: Average loss: 0.6635, Accuracy: 0.604343 60.43426
Epoch 2 Start!
Train set: Average loss: 0.0197, Accuracy: 0.654506 65.45064
Test set: Average loss: 0.6518, Accuracy: 0.613993 61.39928
Epoch 3 Start!
Train set: Average loss: 0.0195, Accuracy: 0.656652 65.66524
Test set: Average loss: 0.6702, Accuracy: 0.606755 60.67551
Epoch 4 Start!
Train set: Average loss: 0.0194, Accuracy: 0.664163 66.41631
Test set: Average loss: 0.6372, Accuracy: 0.633293 63.32931
Epoch 5 Start!
Train set: Average loss: 0.0194, Accuracy: 0.662285 66.22854
Test set: Average loss: 0.6487, Accuracy: 0.623643 62.36429
Epoch 6 Start!
Train set: Average loss: 0.0193, Accuracy: 0.670333 67.03326
Test set: Average loss: 0.6480, Accuracy: 0.626055 62.60555
Epoch 7 Start!
Train set: Average loss: 0.0192, Accuracy: 0.674088 67.40880
Test set: Average loss: 0.6514, Accuracy: 0.630881 63.08806
Epoch 8 Start!
Train set: Average loss: 0.0191, Accuracy: 0.678112 67.81116
Test set: Average loss: 0.6459, Accuracy: 0.641737 64.17370
Epoch 9 Start!
Train set: Average loss: 0.0192, Accuracy: 0.671942 67.19421
Test set: Average loss: 0.6422, Accuracy: 0.626055 62.60555
Epoch 10 Start!
Train set: Average loss: 0.0193, Accuracy: 0.671406 67.14056
Test set: Average loss: 0.6466, Accuracy: 0.618818 61.88179
Epoch 11 Start!
Train set: Average loss: 0.0191, Accuracy: 0.676905 67.69045
Test set: Average loss: 0.6535, Accuracy: 0.613993 61.39928
Epoch 12 Start!
Train set: Average loss: 0.0191, Accuracy: 0.681196 68.11964
Test set: Average loss: 0.6523, Accuracy: 0.630881 63.08806
Epoch 13 Start!
Train set: Average loss: 0.0191, Accuracy: 0.677039 67.70386
Test set: Average loss: 0.6480, Accuracy: 0.622437 62.24367
Epoch 14 Start!
Train set: Average loss: 0.0191, Accuracy: 0.681196 68.11964
Test set: Average loss: 0.6461, Accuracy: 0.632087 63.20869
Epoch 15 Start!
Train set: Average loss: 0.0191, Accuracy: 0.684013 68.40129
Test set: Average loss: 0.6564, Accuracy: 0.632087 63.20869
Epoch 16 Start!
Train set: Average loss: 0.0190, Accuracy: 0.675966 67.59657
Test set: Average loss: 0.6419, Accuracy: 0.623643 62.36429
Epoch 17 Start!
Train set: Average loss: 0.0189, Accuracy: 0.687098 68.70976
Test set: Average loss: 0.6416, Accuracy: 0.634499 63.44994
Epoch 18 Start!
Train set: Average loss: 0.0189, Accuracy: 0.682672 68.26717
Test set: Average loss: 0.6413, Accuracy: 0.629674 62.96743
Epoch 19 Start!
Train set: Average loss: 0.0191, Accuracy: 0.682135 68.21352
Test set: Average loss: 0.6569, Accuracy: 0.624849 62.48492
Epoch 20 Start!
Train set: Average loss: 0.0189, Accuracy: 0.683342 68.33423
```

```
Test set: Average loss: 0.6587, Accuracy: 0.645356 64.53559
Epoch 21 Start!
Train set: Average loss: 0.0190, Accuracy: 0.683342 68.33423
Test set: Average loss: 0.6423, Accuracy: 0.629674 62.96743
Epoch 22 Start!
Train set: Average loss: 0.0189, Accuracy: 0.686561 68.65612
Test set: Average loss: 0.6493, Accuracy: 0.648975 64.89747
Epoch 23 Start!
Train set: Average loss: 0.0189, Accuracy: 0.681465 68.14646
Test set: Average loss: 0.6541, Accuracy: 0.633293 63.32931
Epoch 24 Start!
Train set: Average loss: 0.0189, Accuracy: 0.685756 68.57564
Test set: Average loss: 0.6823, Accuracy: 0.610374 61.03739
Epoch 25 Start!
Train set: Average loss: 0.0188, Accuracy: 0.688439 68.84388
Test set: Average loss: 0.6590, Accuracy: 0.635706 63.57057
Epoch 26 Start!
Train set: Average loss: 0.0188, Accuracy: 0.688439 68.84388
Test set: Average loss: 0.6371, Accuracy: 0.622437 62.24367
Epoch 27 Start!
Train set: Average loss: 0.0189, Accuracy: 0.685220 68.52200
Test set: Average loss: 0.6407, Accuracy: 0.651387 65.13872
Epoch 28 Start!
Train set: Average loss: 0.0189, Accuracy: 0.688036 68.80365
Test set: Average loss: 0.6424, Accuracy: 0.640531 64.05308
Epoch 29 Start!
Train set: Average loss: 0.0188, Accuracy: 0.691926 69.19260
Test set: Average loss: 0.6731, Accuracy: 0.629674 62.96743
Epoch 30 Start!
Train set: Average loss: 0.0188, Accuracy: 0.688841 68.88412
Test set: Average loss: 0.6391, Accuracy: 0.623643 62.36429
```