딥러닝 Classification Model 구현

- 설명 : https://towardsdatascience.com/deep-learning-for-tabular-data-using-pytorch-1807f2858320
- 코드 : https://jovian.ai/aakanksha-ns/shelter-outcome
- 데이터 : https://www.kaggle.com/c/shelter-animal-outcomes/data
- 데이터 사용 동의 처리 필요 : https://www.kaggle.com/c/quora-question-pairs -> data 메뉴 -> data 다운로드 버튼 클릭

데이터 파일 다운로드

```
! pip install -q kaggle
```

```
from google.colab import files

files.upload()
```

```
파일 선택 kaggle.json
```

- **kaggle.json**(application/json) - 64 bytes, last modified: 2021. 5. 16. - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"kgpark88","key":"43d297843da2e99840b860daaae58cc1"}'}
```

```
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

```
! kaggle competitions download -c shelter-animal-outcomes
```

```
Warning: Looks like you're using an outdated API Version, please consider updating (server 1.5.12 / client 1.5.4)
Downloading train.csv.gz to /content
  0% 0.00/521k [00:00<?, ?B/s]
```

```
100% 521k/521k [00:00<00:00, 32.7MB/s]
Downloading test.csv.gz to /content
  0% 0.00/191k [00:00<?, ?B/s]
100% 191k/191k [00:00<00:00, 62.6MB/s]
Downloading sample_submission.csv.gz to /content
  0% 0.00/15.1k [00:00<?, ?B/s]
100% 15.1k/15.1k [00:00<00:00, 16.6MB/s]
```

```
ls -ltr
```

```
total 3928
drwxr-xr-x 1 root root    4096 May  6 13:44 sample_data/
-rw-r--r-- 1 root root      64 May 16 12:18 kaggle.json
-rw-r--r-- 1 root root 2824793 May 16 12:19 train.csv
-rw-r--r-- 1 root root 1009045 May 16 12:19 test.csv
-rw-r--r-- 1 root root  172243 May 16 12:19 sample_submission.csv
```

```
! gzip -d train.csv.gz
! gzip -d test.csv.gz
! gzip -d sample_submission.csv.gz
```

Deep Learning for tabular data using Pytorch

Dataset - https://www.kaggle.com/c/shelter-animal-outcomes

Problem Statement: Given certain features about a shelter animal (like age, sex, color, breed), predict its outcome.

There are 5 possible outcomes: Return_to_owner, Euthanasia, Adoption, Transfer, Died. We are expected to find the probability of an animal's outcome belonging to each of the 5 categories.

Library imports

```
import pandas as pd
import numpy as np
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import torch
from torch.utils.data import Dataset, DataLoader
import torch.optim as torch_optim
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models
from datetime import datetime
```

Load Data

Training set

```
train = pd.read_csv('train.csv')
print("Shape:", train.shape)
train.head()
```

Shape: (26729, 10)

	AnimalID	Name	DateTime	OutcomeType	OutcomeSubtype	AnimalType	SexuponOutcome	AgeuponOutcome	Breed	
0	A671945	Hambone	2014-02-12 18:22:00	Return_to_owner	NaN	Dog	Neutered Male	1 year	Shetland Sheepdog Mix	B
1	A656520	Emily	2013-10-13	Euthanasia	Suffering	Cat	Spayed Female	1 year	Domestic Shorthair Mix	

Test set

```
test = pd.read_csv('test.csv')
print("Shape:", test.shape)
test.head()
```

Shape: (11456, 8)

	ID	Name	DateTime	AnimalType	SexuponOutcome	AgeuponOutcome	Breed	Color
0	1	Summer	2015-10-12 12:15:00	Dog	Intact Female	10 months	Labrador Retriever Mix	Red/White
1	2	Cheyenne	2014-07-26 17:59:00	Dog	Spayed Female	2 years	German Shepherd/Siberian Husky	Black/Tan
2	3	Gus	2016-01-13 12:20:00	Cat	Neutered Male	1 year	Domestic Shorthair Mix	Brown Tabby
3	4	Pongo	2013-12-28 18:12:00	Dog	Intact Male	4 months	Collie Smooth Mix	Tricolor
4	5	Skooter	2015-09-24 17:59:00	Dog	Neutered Male	2 years	Miniature Poodle Mix	White

Sample submission file

For each row, each outcome's probability needs to be filled into the columns

```
sample = pd.read_csv('sample_submission.csv')
sample.head()
```

	ID	Adoption	Died	Euthanasia	Return_to_owner	Transfer
0	1	1	0	0	0	0
1	2	1	0	0	0	0
2	3	1	0	0	0	0
3	4	1	0	0	0	0
4	5	1	0	0	0	0

Very basic data exploration

How balanced is the dataset?

Adoption and Transfer seem to occur a lot more than the rest

```
Counter(train['OutcomeType'])
```

```
Counter({'Adoption': 10769,
         'Died': 197,
         'Euthanasia': 1555,
         'Return_to_owner': 4786,
         'Transfer': 9422})
```

What are the most common names and how many times do they occur?

There seem to be too many Nan values. Name might not be a very important factor too

```
Counter(train['Name']).most_common(5)
```

```
[(nan, 7691), ('Max', 136), ('Bella', 135), ('Charlie', 107), ('Daisy', 106)]
```

Data preprocessing

OutcomeSubtype column seems to be of no use, so we drop it. Also, since animal ID is unique, it doesn't help in training

```
train_X = train.drop(columns= ['OutcomeType', 'OutcomeSubtype', 'AnimalID'])
Y = train['OutcomeType']
test_X = test
```

Stacking train and test set so that they undergo the same preprocessing

```
stacked_df = train_X.append(test_X.drop(columns=['ID']))
```

splitting datetime into month and year

```
# stacked_df['DateTime'] = pd.to_datetime(stacked_df['DateTime'])
# stacked_df['year'] = stacked_df['DateTime'].dt.year
# stacked_df['month'] = stacked_df['DateTime'].dt.month
stacked_df = stacked_df.drop(columns=['DateTime'])
stacked_df.head()
```

	Name	AnimalType	SexuponOutcome	AgeuponOutcome	Breed	Color
0	Hambone	Dog	Neutered Male	1 year	Shetland Sheepdog Mix	Brown/White

dropping columns with too many nulls

```
for col in stacked_df.columns:
    if stacked_df[col].isnull().sum() > 10000:
        print("dropping", col, stacked_df[col].isnull().sum())
        stacked_df = stacked_df.drop(columns = [col])
```

```
dropping Name 10916
```

```
stacked_df.head()
```

	AnimalType	SexuponOutcome	AgeuponOutcome	Breed	Color
0	Dog	Neutered Male	1 year	Shetland Sheepdog Mix	Brown/White
1	Cat	Spayed Female	1 year	Domestic Shorthair Mix	Cream Tabby
2	Dog	Neutered Male	2 years	Pit Bull Mix	Blue/White
3	Cat	Intact Male	3 weeks	Domestic Shorthair Mix	Blue Cream
4	Dog	Neutered Male	2 years	Lhasa Apso/Miniature Poodle	Tan

label encoding

```
for col in stacked_df.columns:
    if stacked_df.dtypes[col] == "object":
        stacked_df[col] = stacked_df[col].fillna("NA")
    else:
        stacked_df[col] = stacked_df[col].fillna(0)
    stacked_df[col] = LabelEncoder().fit_transform(stacked_df[col])
```

```
stacked_df.head()
```

	AnimalType	SexuponOutcome	AgeuponOutcome	Breed	Color
0	1	3	5	1482	146
1	0	4	5	775	184
2	1	3	21	1293	97
3	0	1	26	775	47
4	1	3	21	1101	311

```
# making all variables categorical
for col in stacked_df.columns:
    stacked_df[col] = stacked_df[col].astype('category')
```

splitting back train and test

```
X = stacked_df[0:26729]
test_processed = stacked_df[26729:]

#check if shape[0] matches original
print("train shape: ", X.shape, "orignal: ", train.shape)
print("test shape: ", test_processed.shape, "original: ", test.shape)
```

```
train shape:  (26729, 5) orignal:  (26729, 10)
test shape:  (11456, 5) original:  (11456, 8)
```

Encoding target

```
Y = LabelEncoder().fit_transform(Y)
```

```
#sanity check to see numbers match and matching with previous counter to create target dictionary
print(Counter(train['OutcomeType']))
print(Counter(Y))
target_dict = {
    'Return_to_owner' : 3,
    'Euthanasia': 2,
    'Adoption': 0,
    'Transfer': 4,
    'Died': 1
}
```

```
Counter({'Adoption': 10769, 'Transfer': 9422, 'Return_to_owner': 4786, 'Euthanasia': 1555, 'Died': 197})
Counter({0: 10769, 4: 9422, 3: 4786, 2: 1555, 1: 197})
```

train-valid split

```
X_train, X_val, y_train, y_val = train_test_split(X, Y, test_size=0.10, random_state=0)
X_train.head()
```

	AnimalType	SexuponOutcome	AgeuponOutcome	Breed	Color
6917	1	3	5	1293	146
13225	0	4	33	1515	231
2697	1	4	5	1353	43
21905	1	3	31	245	40
17071	0	4	37	775	156

Choosing columns for embedding

```
#categorical embedding for columns having more than two values
```

```
embedded_cols = {n: len(col.cat.categories) for n,col in X.items() if len(col.cat.categories) > 2}
embedded_cols
```

```
{'AgeuponOutcome': 46, 'Breed': 1678, 'Color': 411, 'SexuponOutcome': 6}
```

```
embedded_col_names = embedded_cols.keys()
len(X.columns) - len(embedded_cols) #number of numerical columns
```

```
1
```

Determining size of embedding

(borrowed from https://www.usfca.edu/data-institute/certificates/fundamentals-deep-learning lesson 2)

```
embedding_sizes = [(n_categories, min(50, (n_categories+1)//2)) for _,n_categories in embedded_cols.items()]
embedding_sizes
```

```
[(6, 3), (46, 23), (1678, 50), (411, 50)]
```

Pytorch Dataset

```
class ShelterOutcomeDataset(Dataset):
    def __init__(self, X, Y, embedded_col_names):
        X = X.copy()
        self.X1 = X.loc[:,embedded_col_names].copy().values.astype(np.int64) #categorical columns
        self.X2 = X.drop(columns=embedded_col_names).copy().values.astype(np.float32) #numerical columns
        self.y = Y

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.X1[idx], self.X2[idx], self.y[idx]
```

```
#creating train and valid datasets
train_ds = ShelterOutcomeDataset(X_train, y_train, embedded_col_names)
valid_ds = ShelterOutcomeDataset(X_val, y_val, embedded_col_names)
```

Making device (GPU/CPU) compatible

(borrowed from https://jovian.ml/aakashns/04-feedforward-nn)

In order to make use of a GPU if available, we'll have to move our data and model to it.

```
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')


def to_device(data, device):
    """Move tensor(s) to chosen device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)


class DeviceDataLoader():
    """Wrap a dataloader to move data to a device"""
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        """Yield a batch of data after moving it to device"""
        for b in self.dl:
            yield to_device(b, self.device)
```

```
    def __len__(self):
        """Number of batches"""
        return len(self.dl)


device = get_default_device()
device
```

```
      device(type='cpu')
```

Model

(modified from https://www.usfca.edu/data-institute/certificates/fundamentals-deep-learning lesson 2)

```
class ShelterOutcomeModel(nn.Module):
    def __init__(self, embedding_sizes, n_cont):
        super().__init__()
        self.embeddings = nn.ModuleList([nn.Embedding(categories, size) for categories,size in embedding_sizes])
        n_emb = sum(e.embedding_dim for e in self.embeddings) #length of all embeddings combined
        self.n_emb, self.n_cont = n_emb, n_cont
        self.lin1 = nn.Linear(self.n_emb + self.n_cont, 200)
        self.lin2 = nn.Linear(200, 70)
        self.lin3 = nn.Linear(70, 5)
        self.bn1 = nn.BatchNorm1d(self.n_cont)
        self.bn2 = nn.BatchNorm1d(200)
        self.bn3 = nn.BatchNorm1d(70)
        self.emb_drop = nn.Dropout(0.6)
        self.drops = nn.Dropout(0.3)


    def forward(self, x_cat, x_cont):
        x = [e(x_cat[:,i]) for i,e in enumerate(self.embeddings)]
        x = torch.cat(x, 1)
        x = self.emb_drop(x)
        x2 = self.bn1(x_cont)
```

```
        x = torch.cat([x, x2], 1)
        x = F.relu(self.lin1(x))
        x = self.drops(x)
        x = self.bn2(x)
        x = F.relu(self.lin2(x))
        x = self.drops(x)
        x = self.bn3(x)
        x = self.lin3(x)
        return x


model = ShelterOutcomeModel(embedding_sizes, 1)
to_device(model, device)
```

```
ShelterOutcomeModel(
  (embeddings): ModuleList(
    (0): Embedding(6, 3)
    (1): Embedding(46, 23)
    (2): Embedding(1678, 50)
    (3): Embedding(411, 50)
  )
  (lin1): Linear(in_features=127, out_features=200, bias=True)
  (lin2): Linear(in_features=200, out_features=70, bias=True)
  (lin3): Linear(in_features=70, out_features=5, bias=True)
  (bn1): BatchNorm1d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn3): BatchNorm1d(70, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (emb_drop): Dropout(p=0.6, inplace=False)
  (drops): Dropout(p=0.3, inplace=False)
)
```

Optimizer

```
def get_optimizer(model, lr = 0.001, wd = 0.0):
    parameters = filter(lambda p: p.requires_grad, model.parameters())
    optim = torch_optim.Adam(parameters, lr=lr, weight_decay=wd)
    return optim
```

Training function

```
def train_model(model, optim, train_dl):
    model.train()
    total = 0
    sum_loss = 0
    for x1, x2, y in train_dl:
        batch = y.shape[0]
        output = model(x1, x2)
        loss = F.cross_entropy(output, y)
        optim.zero_grad()
        loss.backward()
        optim.step()
        total += batch
        sum_loss += batch*(loss.item())
    return sum_loss/total
```

Evaluation function

```
def val_loss(model, valid_dl):
    model.eval()
    total = 0
    sum_loss = 0
    correct = 0
    for x1, x2, y in valid_dl:
        current_batch_size = y.shape[0]
        out = model(x1, x2)
        loss = F.cross_entropy(out, y)
        sum_loss += current_batch_size*(loss.item())
        total += current_batch_size
        pred = torch.max(out, 1)[1]
        correct += (pred == y).float().sum().item()
    print("valid loss %.3f and accuracy %.3f" % (sum_loss/total, correct/total))
    return sum_loss/total, correct/total
```

```
    return sum_loss/total, correct/total

def train_loop(model, epochs, lr=0.01, wd=0.0):
    optim = get_optimizer(model, lr = lr, wd = wd)
    for i in range(epochs):
        loss = train_model(model, optim, train_dl)
        print("training loss: ", loss)
        val_loss(model, valid_dl)
```

Training

```
batch_size = 1000
train_dl = DataLoader(train_ds, batch_size=batch_size,shuffle=True)
valid_dl = DataLoader(valid_ds, batch_size=batch_size,shuffle=True)


train_dl = DeviceDataLoader(train_dl, device)
valid_dl = DeviceDataLoader(valid_dl, device)


train_loop(model, epochs=8, lr=0.05, wd=0.00001)
```

```
training loss:  1.1929032033886036
valid loss 0.959 and accuracy 0.593
training loss:  1.0077161927099516
valid loss 0.889 and accuracy 0.626
training loss:  0.9727969404506969
valid loss 0.875 and accuracy 0.640
training loss:  0.9615953660582164
valid loss 0.895 and accuracy 0.641
training loss:  0.9480069062421993
valid loss 0.892 and accuracy 0.630
training loss:  0.9471241884109464
valid loss 0.882 and accuracy 0.641
training loss:  0.9440713606088471
valid loss 0.868 and accuracy 0.643
```

```
training loss:  0.9443685124988448
valid loss 0.864 and accuracy 0.646
```

Test Output

```
test_ds = ShelterOutcomeDataset(test_processed, np.zeros(len(test_processed)), embedded_col_names)
test_dl = DataLoader(test_ds, batch_size=batch_size)
```

```
preds = []
with torch.no_grad():
    for x1,x2,y in test_dl:
        out = model(x1, x2)
        prob = F.softmax(out, dim=1)
        preds.append(prob)
```

```
final_probs = [item for sublist in preds for item in sublist]
```

```
len(final_probs)
```

```
11456
```

```
target_dict
```

```
{'Adoption': 0,
 'Died': 1,
 'Euthanasia': 2,
 'Return_to_owner': 3,
 'Transfer': 4}
```

```
sample.head()
```

	ID	Adoption	Died	Euthanasia	Return_to_owner	Transfer
0	1	1	0	0	0	0
1	2	1	0	0	0	0
2	3	1	0	0	0	0
3	4	1	0	0	0	0

```
sample['Adoption'] = [float(t[0]) for t in final_probs]
sample['Died'] = [float(t[1]) for t in final_probs]
sample['Euthanasia'] = [float(t[2]) for t in final_probs]
sample['Return_to_owner'] = [float(t[3]) for t in final_probs]
sample['Transfer'] = [float(t[4]) for t in final_probs]
sample.head()
```

	ID	Adoption	Died	Euthanasia	Return_to_owner	Transfer
0	1	0.061584	0.013002	0.145855	0.090792	0.688767
1	2	0.422662	0.004715	0.045230	0.321695	0.205698
2	3	0.354422	0.006220	0.053112	0.102744	0.483502
3	4	0.060504	0.012325	0.127738	0.089680	0.709753
4	5	0.539624	0.001352	0.010254	0.278997	0.169774

```
sample.to_csv('samp.csv', index=False)
```