Load data and data cleaning

```
# read in the file
file = open('/content/drive/MyDrive/zoo.data', 'r')
# create a list to include all the lines in the text file
data = []
for line in file:
    data.append(line)

# close file
file.close()
# remove /n from end of each line
data = [i.strip() for i in data]
```

display the first 5 records in the data

```
data[0:5]
```

```
     ['aardvark,1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,1',
      'antelope,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,1',
      'bass,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,4',
      'bear,1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,1',
      'boar,1,0,0,1,0,0,1,1,1,1,0,0,4,1,0,1,1']
```

We now split each record by “,” and turn each record from string to list. Then we get rid of the animal names as it’s not useful for prediction. And lastly, we use a defined function and translate legs attribute values to (0, 1, 2, 3, 4, 5). The reason for that is it’s easier to follow up computing, as we can directly use attribute value as index to retrieve possibilities from the model.

```
split_data = [d.split(",") for d in data]
animal_names = []
for row in split_data:
    name = row.pop(0)
    animal_names.append(name)
# define a function to convert legs attribute to 0 - 5
def convert_legs_value(data):
    value_list = ['0', '2', '4', '5', '6', '8']
    convert_list = ['0', '1', '2', '3', '4', '5']
    for row in data:
        row[12] = convert_list[value_list.index(row[12])]
    return data
split_data = convert_legs_value(split_data)
```

Remember the last element for each record is the label. And we successfully translated “legs” values (attribute 13). For example for the first two animals, the original value is 4 and now it’s 2.

```
for i in range(5):
    print(split_data[i])

     ['1', '0', '0', '1', '0', '0', '1', '1', '1', '1', '0', '0', '2', '0', '0', '1', '1']
     ['1', '0', '0', '1', '0', '0', '0', '1', '1', '1', '0', '0', '2', '1', '0', '1', '1']
     ['0', '0', '1', '0', '0', '1', '1', '1', '1', '0', '0', '1', '0', '1', '0', '0', '4']
     ['1', '0', '0', '1', '0', '0', '1', '1', '1', '1', '0', '0', '2', '0', '0', '1', '1']
     ['1', '0', '0', '1', '0', '0', '1', '1', '1', '1', '0', '0', '2', '1', '0', '1', '1']
```

Split train and test data

First we define a function for data splitting. Actually for test_size argument, we can input sample number (int) or percentage (float). And the function can handle both

```
def train_test_split(data, test_size):

    if isinstance(test_size, float):
        test_size = round(test_size * len(data))
    test_indices = random.sample(range(len(data)), k=test_size)
    train_indices = [i for i in range(len(data)) if i not in test_indices]

    test_df = []
    for index in test_indices:
        test_d = data[index]
        test_df.append(test_d)

    train_df = []
    for index in train_indices:
        train_d = data[index]
        train_df.append(train_d)

    return train_df, test_df
```

Then we can use this function and randomly split data into train data (71) and test data (30). The line random.seed(0) can help to make sure each time runs the code you can have the same results.

```
import random
random.seed(0)
train_data, test_data = train_test_split(split_data, test_size = 30)
train_sample_num = len(train_data)
attr_num = len(train_data[0])-1
```

We also assigned two global variables: train_sample_num and attr_num, which will be used for later computing. There is a “-1” for attr_num because the last value for each animal is the label, not an attribute.

Separate Data by Class

```
# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
    separated = dict()
    for i in range(len(dataset)):
        vector = dataset[i]
        class_value = vector[-1]
        if (class_value not in separated):
            separated[class_value] = list()
        separated[class_value].append(vector)
    return separated
separated = separate_by_class(train_data)

for key, value in separated.items():
    print(key)
    print(value)
```

```
1
[['1', '0', '0', '1', '0', '0', '1', '1', '1', '1', '0', '0', '2', '0', '0', '1', '1'], ['1', '0', '0', '1', '0', '0',
4
[['0', '0', '1', '0', '0', '1', '1', '1', '1', '0', '0', '1', '0', '1', '0', '0', '4'], ['0', '0', '1', '0', '0', '1',
2
[['0', '1', '1', '0', '1', '0', '0', '0', '1', '1', '0', '0', '1', '1', '1', '0', '2'], ['0', '1', '1', '0', '1', '0',
7
[['0', '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '7'], ['0', '0', '1', '0', '0', '1',
6
[['0', '0', '1', '0', '0', '0', '0', '0', '0', '1', '0', '0', '4', '0', '0', '0', '6'], ['0', '0', '1', '0', '1', '0',
5
[['0', '0', '1', '0', '0', '1', '1', '1', '1', '1', '0', '0', '2', '0', '0', '0', '5'], ['0', '0', '1', '0', '0', '1',
3
[['0', '0', '1', '0', '0', '0', '1', '1', '1', '1', '0', '0', '0', '1', '0', '0', '3'], ['0', '0', '1', '0', '0', '0',
```

Train model by computing possibilities

```
def calculate_class_py(label):
    count = len(label)
    py = count/train_sample_num
    return py


def calculate_pxy(label):
    pxy = []
    for i in range(attr_num):
        if i != 12:
            count0 = 0
            count1 = 0
            for row in label:
                if row[i] == '0':
                    count0 += 1
                else:
                    count1 += 1
            cal = ((count0 + 0.1)/(len(label) + 0.2), (count1 + 0.1)/(len(label) + 0.2))
            pxy.append(cal)
        else:
            count0 = 0
            count1 = 0
            count2 = 0
            count3 = 0
            count4 = 0
            count5 = 0
            for row in label:
                if row[i] == '0':
                    count0 += 1
                elif row[i] == '1':
                    count1 += 1
                elif row[i] == '2':
                    count2 += 1
                elif row[i] == '3':
                    count3 += 1
                elif row[i] == '4':
                    count4 += 1
                else:
                    count5 += 1
            cal = ((count0 + 0.1)/(len(label) + 0.6), (count1 + 0.1)/(len(label) + 0.6), (count2 + 0.1)/(len(label) + 0.6),
                   (count3 + 0.1)/(len(label) + 0.6), (count4 + 0.1)/(len(label) + 0.6), (count5 + 0.1)/(len(label) + 0.6))
            pxy.append(cal)

    return pxy


def summarize_by_class(separated):
    model = dict()
    for class_value, rows in separated.items():
        pxy = calculate_pxy(rows)
        py = calculate_class_py(rows)
        model[class_value] = [pxy, py]
    return model
model = summarize_by_class(separated)

for key, possibilities in model.items():
    print(key)
    print(possibilities)
```

```
1
[[(0.07191780821917809, 0.928082191780822), (0.9965753424657535, 0.003424657534246575 6), (0.9623287671232877, 0.0376712
4
[[(0.986111111111111, 0.01388888888888889), (0.986111111111111, 0.01388888888888889), (0.01388888888888889, 0.986111111
2
[[(0.9938271604938272, 0.00617283950617284), (0.00617283950617284, 0.9938271604938272), (0.00617283950617284, 0.9938271
7
[[(0.9878048780487805, 0.012195121951219514), (0.9878048780487805, 0.012195121951219514), (0.13414634146341467, 0.86585
6
[[(0.7380952380952381, 0.2619047619047619), (0.976190476190476, 0.023809523809523808), (0.023809523809523808, 0.9761904
5
[[(0.976190476190476, 0.023809523809523808), (0.976190476190476, 0.023809523809523808), (0.023809523809523808, 0.976190
3
[[(0.96875, 0.03125), (0.96875, 0.03125), (0.03125, 0.96875), (0.96875, 0.03125), (0.96875, 0.03125), (0.96875, 0.03125
```

Make predictions & model evaluation

```
def calculate_class_probabilities(model, sample):
    probabilities = dict()
    for class_value, class_summaries in model.items():
        probabilities[class_value] = class_summaries[1]
        for i in range(attr_num):
            probabilities[class_value] *= class_summaries[0][i][int(sample[i])]
    return probabilities
```

```
calculate_class_probabilities(model, test_data[0])
```

```
     {'1': 0.013801682164170657,
      '2': 3.5174576845377747e-16,
      '3': 1.5975084126545406e-07,
      '4': 2.6461234937773274e-15,
      '5': 1.8987825780160568e-09,
      '6': 2.468331194908214e-14,
      '7': 6.568440729918505e-13}
```

```
# Predict the class for a given sample
def predict(model, sample):
    probabilities = calculate_class_probabilities(model, sample)
    best_label, best_prob = None, -1
    for class_value, probability in probabilities.items():
        if best_label is None or probability > best_prob:
            best_prob = probability
            best_label = class_value
    return best_label


# get all the predictions
predictions = []
for row in test_data:
    best_label = predict(model, row)
    predictions.append(best_label)
# get all the real labels
actual = []
for row in test_data:
    real_label = row[-1]
    actual.append(real_label)

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
accuracy = accuracy_metric(actual, predictions)
```

```
print("The accuracy of predcition is: %.2f percent!" %accuracy )
```

```
The accuracy of predcition is: 96.67 percent!
```

✓ 0s completed at 12:03 PM