Naïve Bayes Classifier

Associate Prof. Huỳnh Trung Hiếu

Classes at IUH

- Let us assume the classes in IUH start from 7am. Our house are 5 kms far from the IUH. If we leave home by 6:30 am we will in time for classes otherwise depending on the traffic conditions. Following lists ‘in_time’ (coming classes in time) and ‘too_late’ (coming classes is lated) are data showing the situation over some weeks. The first component of each tuple shows the minutes after 6:30 am that we leave home for classes, and the second component shows the number of time this occurred.

- what is the probability for lately coming class if leaving home at 6:32?
- Today I leave home at 6:38 am, what is the probability for lately coming class?

Classes at IUH

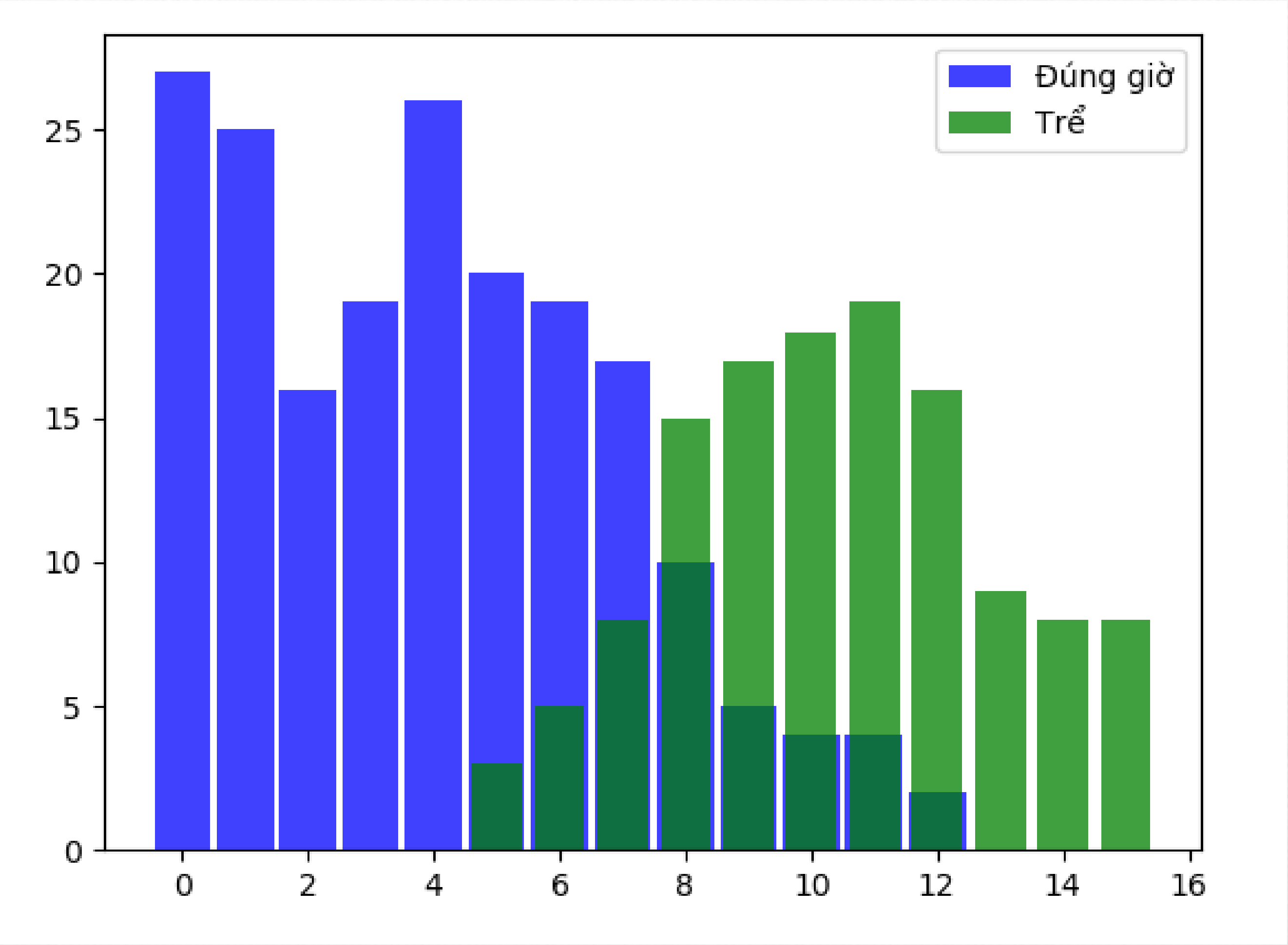
- in_time = [(0, 27), (1, 25), (2, 16), (3, 19), (4, 26), (5, 20), (6, 19), (7, 17), (8, 10), (9, 5), (10, 4), (11, 4), (12,2)]
- cls_late = [(5,3), (6, 5), (7, 8), (8, 15), (9, 17), (10, 18), (11, 19), (12,16), (13, 9), (14, 8), (15, 8)]

Classes at IUH (Lec2_Ex1.py)

- Visualize data:

```
X, Y = zip(*in_time)
X2, Y2 = zip(*cls_late)
bar_width = 0.9
plt.bar(X, Y, bar_width, color="blue", alpha=0.75, label="Đúng giờ")
bar_width = 0.8
plt.bar(X2, Y2, bar_width, color="green", alpha=0.75, label="Trễ")
plt.legend(loc='upper right')
plt.show()
```

Classes at IUH



Classes at IUH

- From this data we can deduce that the probability of lately coming classes when leaving home at 6:32 is 1, because we had 16 successful cases experienced and no late, i.e. there is no tuple with 2 as the first component in ‘cls_late’.
- We will denote the event “Coming classes in time" with S(success) and the event " Coming classes lately" with L(late).

Classes at IUH

- S: “Coming classes in time"
- L: “Coming classes lately”.

- The probability “coming classe in time that we leave home at 6:32” can be defined fomally:

$$P(S|2)=16/16=1$$

Classes at IUH

- Leaving home on 6:38 am:
  - Number of cases in “in_time” list: 10
  - Number of cases in “cls_late” list: 15

=> $P(S|8)=10/(10+15)=0.4$

$P(L|8)=15/(10+15)=0.6$

Classes at IUH

- We can write a 'classifier' function, which will give the probability for coming classes in time:

```
in_time_dict = dict(in_time)
too_late_dict = dict(cls_late)
def catch_the_train(min):
    s = in_time_dict.get(min, 0)
    if s == 0:
        return 0
    else:
        m = too_late_dict.get(min, 0)
        return s / (s + m)
for minutes in range(0, 15):
    print(minutes, catch_the_train(minutes))
```

Person Data

- We will use a file called 'person_data.txt'. It contains 100 random person data, male and female, with body sizes, weights and gender tags.

```
import numpy as np
genders = ["male", "female"]
persons = []
with open("data/person_data.txt") as fh:
    for line in fh:
        persons.append(line.strip().split())
firstnames = {}
heights = {}
for gender in genders:
    firstnames[gender] = [ x[0] for x in persons if x[4]==gender]
    heights[gender] = [ x[2] for x in persons if x[4]==gender]
    heights[gender] = np.array(heights[gender], np.int)

for gender in ("female", "male"):
    print(gender + ":")
    print(firstnames[gender][:10])
    print(heights[gender][:10])
```

Person Data

- We will now define a Python class "Feature" for the features, which we will use for classification later.
- If the feature values are numerical we may want to "bin" them to reduce the number of possible feature values.

```python
class Feature:

    def __init__(self, data, name=None, bin_width=None):
        self.name = name
        self.bin_width = bin_width
        if bin_width:
            self.min, self.max = min(data), max(data)
            bins = np.arange((self.min // bin_width) * bin_width,
                             (self.max // bin_width) * bin_width,
                             bin_width)
            freq, bins = np.histogram(data, bins)
            self.freq_dict = dict(zip(bins, freq))
            self.freq_sum = sum(freq)
        else:
            self.freq_dict = dict(Counter(data))
            self.freq_sum = sum(self.freq_dict.values())


    def frequency(self, value):
        if self.bin_width:
            value = (value // self.bin_width) * self.bin_width
        if value in self.freq_dict:
            return self.freq_dict[value]
        else:
            return 0
```

Person Data

- We will create now two feature classes Feature for the height values of the person data set. One Feature class contains the height for the Naive Bayes class "male" and one the heights for the class "female":

```
fts = {}
for gender in genders:
    fts[gender] = Feature(heights[gender], name=gender, bin_width=5)
    print(gender, fts[gender].freq_dict)
```

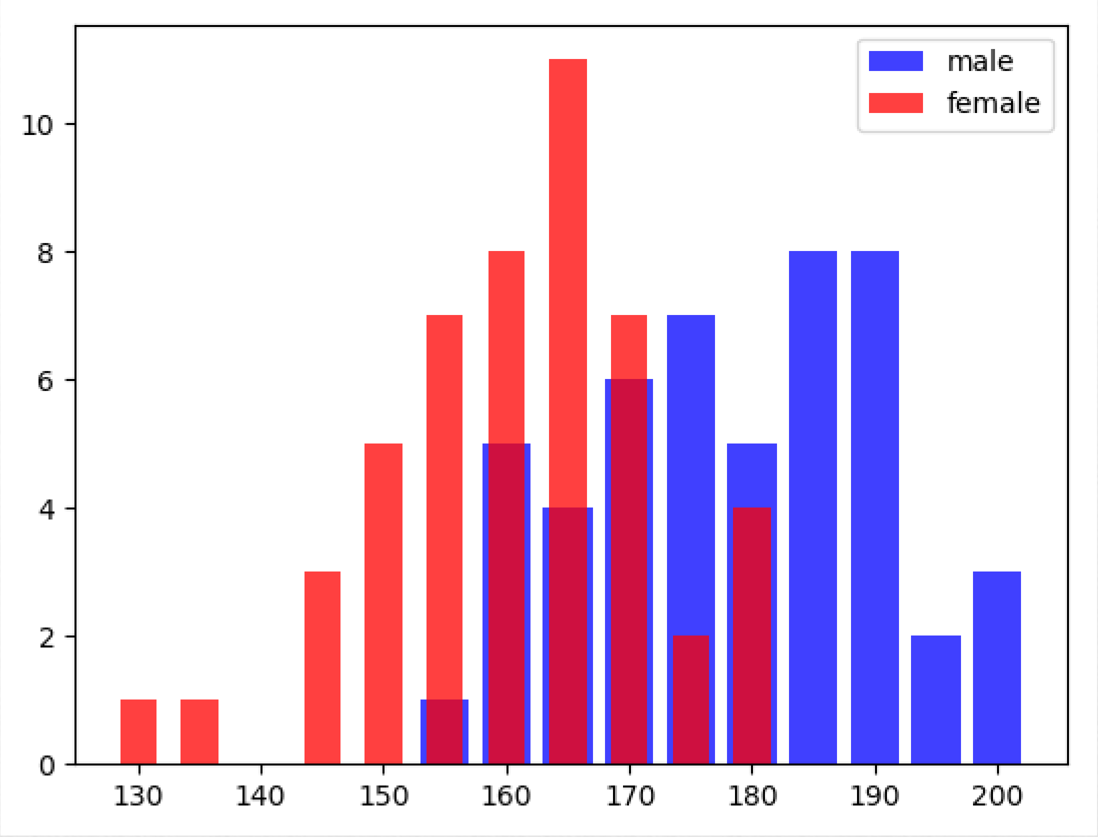
Person Data

- We printed out the frequencies of our bins, but it is a lot better to see these values dipicted in a bar chart

```
for gender in genders:
    frequencies = list(fts[gender].freq_dict.items())
    frequencies.sort(key=lambda x: x[1])
    X, Y = zip(*frequencies)
    color = "blue" if gender=="male" else "red"
    bar_width = 4 if gender=="male" else 3
    plt.bar(X, Y, bar_width, color=color, alpha=0.75, label=gender)
plt.legend(loc='upper right')
plt.show()
```

Person Data

- We printed out the frequencies of our bins, but it is a lot better to see these values dipicted in a bar chart



Person Data

- We have to design now a Naive Bayes class

```
class NBclass:

        def __init__(self, name, *features):
            self.features = features
            self.name = name

        def probability_value_given_feature(self,
                                            feature_value,
                                            feature):
            """
            p_value_given_feature returns the probability p
            for a feature_value 'value' of the feature  to occurr
            corresponds to P(d_i | p_j)
            where d_i is a feature variable of the feature i
            """

            if feature.freq_sum == 0:
                return 0
            else:
                return feature.frequency(feature_value) / feature.freq_sum
```

Person Data

- We will create NBclasses with one feature, i.e. the height feature. We will use the Feature classes of fts, which we have previously created:

```
cls = {}
for gender in genders:
    cls[gender] = NBclass(gender, fts[gender])
```

Person Data

- The final step for creating a simple Naive Bayes classifier consists in writing a class 'Classifier', which will use our classes 'NBclass' and 'Feature'.

```
class Classifier:

    def __init__(self, *nbclasses):
        self.nbclasses = nbclasses


    def prob(self, *d, best_only=True):

        nbclasses = self.nbclasses
        probability_list = []
        for nbclass in nbclasses:
            ftrs = nbclass.features
            prob = 1
            for i in range(len(ftrs)):
                prob *= nbclass.probability_value_given_feature(d[i], ftrs[i])

            probability_list.append( (prob, nbclass.name) )
        prob_values = [f[0] for f in probability_list]
        prob_sum = sum(prob_values)
        if prob_sum==0:
            number_classes = len(self.nbclasses)
            pl = []
            for prob_element in probability_list:
                pl.append( ((1 / number_classes), prob_element[1]))
            probability_list = pl
        else:
            probability_list = [ (p[0] / prob_sum, p[1])  for p in probability_list]
        if best_only:
            return max(probability_list)
        else:
            return probability_list
```

Person Data

- We can also train a classifier with our firstnames:

```
fts = {}
cls = {}
for gender in genders:
    fts_names = Feature(firstnames[gender], name=gender)
    cls[gender] = NBclass(gender, fts_names)

c = Classifier(cls["male"], cls["female"])
testnames = ['Edgar', 'Benjamin', 'Fred', 'Albert', 'Laura',
             'Maria', 'Paula', 'Sharon', 'Jessie']
for name in testnames:
    print(name, c.prob(name))
```

Person Data

- We can also train a classifier with our firstnames:

Edgar (0.5, 'male')

Benjamin (1.0, 'male')

Fred (1.0, 'male')

Albert (1.0, 'male')

Laura (1.0, 'female')

Maria (1.0, 'female')

Paula (1.0, 'female')

Sharon (1.0, 'female')

Jessie (0.6666666666666667, 'female')

Person Data

- We can also train a classifier with our firstnames:

Edgar (0.5, 'male')
Benjamin (1.0, 'male')
Fred (1.0, 'male')
Albert (1.0, 'male')
Laura (1.0, 'female')
Maria (1.0, 'female')
Paula (1.0, 'female')
Sharon (1.0, 'female')
Jessie (0.6666666666666667, 'female')

➡ **The name "Jessie" is an ambiguous name.**

Person Data

The name "Jessie" is an ambiguous name.

```
ambirousJessie=[person for person in persons if person[0] == "Jessie"]

for person in ambirousJessie:
    print(person)
```

```
['Jessie', 'Morgan', '175', '67.0', 'male']
['Jessie', 'Bell', '165', '65', 'female']
['Jessie', 'Washington', '159', '56', 'female']
['Jessie', 'Davis', '174', '45', 'female']
['Jessie', 'Johnson', '165', '30.0', 'male']
['Jessie', 'Thomas', '168', '69', 'female']
```

Person Data

The name "Jessie" is an ambiguous name.

```
['Jessie', 'Morgan', '175', '67.0', 'male']
['Jessie', 'Bell', '165', '65', 'female']
['Jessie', 'Washington', '159', '56', 'female']
['Jessie', 'Davis', '174', '45', 'female']
['Jessie', 'Johnson', '165', '30.0', 'male']
['Jessie', 'Thomas', '168', '69', 'female']
```

Jessie Washington is only 159 cm tall. If we have a look at the results of our Classifier, trained with heights, we see that the likelihood for a person 159 cm tall of being "female" is 0.875. So what about an unknown person called "Jessie" and being 159 cm tall? Is this person female or male?

Person Data

To answer this question, we will train an Naive Bayes classifier with two feature classes, i.e. heights and first names:

```
cls = {}
for gender in genders:
    fts_heights = Feature(heights[gender], name="heights", bin_width=5)
    fts_names = Feature(firstnames[gender], name="names")
    cls[gender] = NBclass(gender, fts_names, fts_heights)
c = Classifier(cls["male"], cls["female"])
for d in [("Maria", 140), ("Anthony", 200), ("Anthony", 153),
          ("Jessie", 188) , ("Jessie", 159), ("Jessie", 160) ]:
    print(d, c.prob(*d, best_only=False))
```

```
('Maria', 140) [(0.5, 'male'), (0.5, 'female')]
('Anthony', 200) [(1.0, 'male'), (0.0, 'female')]
('Anthony', 153) [(0.5, 'male'), (0.5, 'female')]
('Jessie', 188) [(1.0, 'male'), (0.0, 'female')]
('Jessie', 159) [(0.06666666666666667, 'male'), (0.933333333
('Jessie', 160) [(0.23809523809523817, 'male'), (0.761904761
```