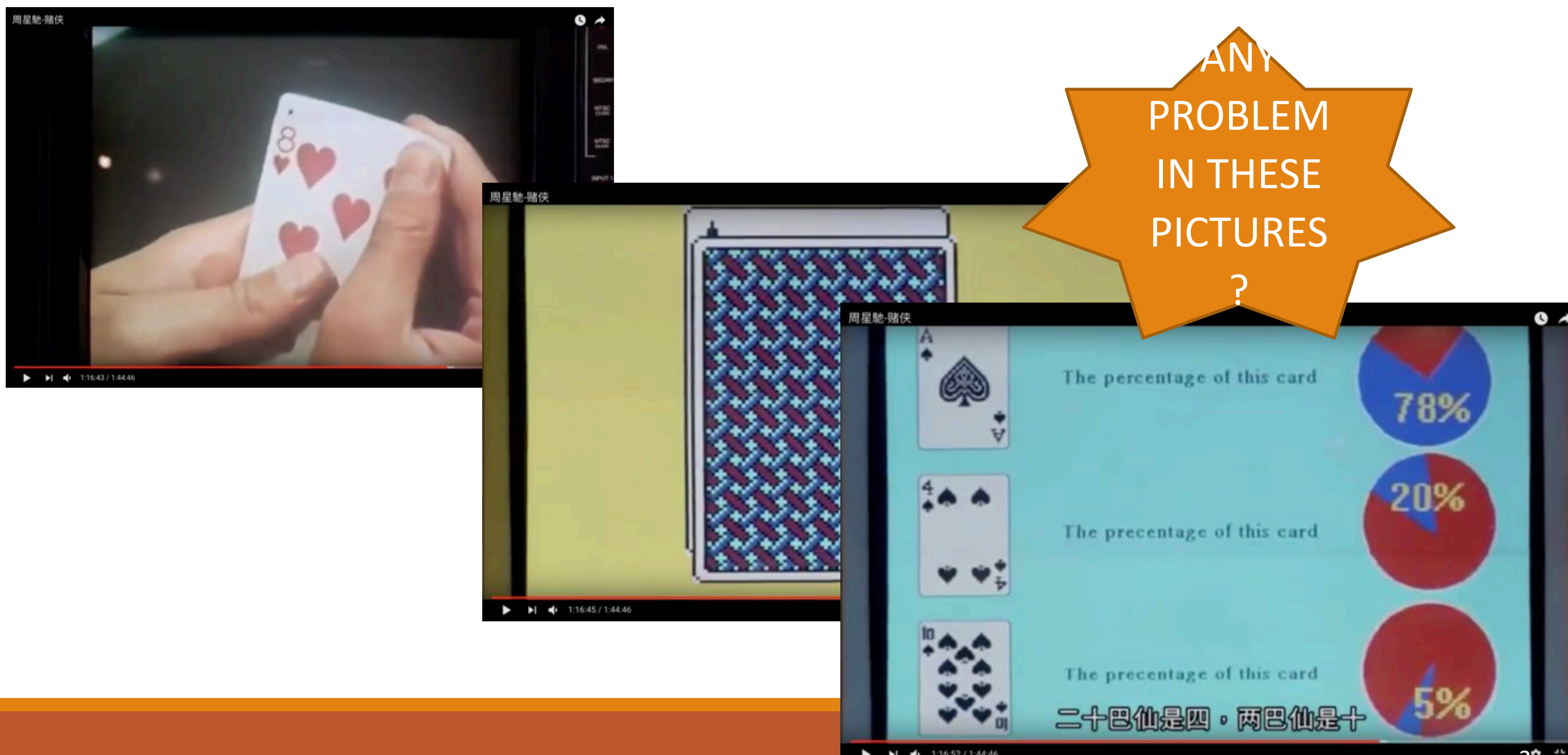
Probability-based Classification

Naïve Bayes Model

CHIEN CHIN CHEN

Naïve Bayes Classification (1/15)

A very welcome classification method, because it shows you **probabilities** about classification predictions.



ANY PROBLEM IN THESE PICTURES ?

Naïve Bayes Classification (2/15)

Naïve Bayes classification is a **probabilistic** learning method.

In text classification, its goal is to find the "**best**" class for a document:

- The class with the **maximum a posteriori** (MAP) probability:
- $$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c \mid d)$$

  $$= \underset{c \in C}{\operatorname{argmax}} \frac{P(c)P(d \mid c)}{P(d)}$$

  $$= \boxed{\underset{c \in C}{\operatorname{argmax}} P(c)P(d \mid c)}$$

The probability of class *c* once we observe *d*

This is the classifier (*classification model*) of NB classification

- The distributions of ***P*(*c*)** and ***P*(*d*|*c*)** are **model parameters** that we learned (estimated) from training data.

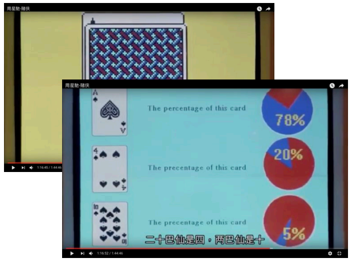
Naïve Bayes Classification (3/15)

Naïve Bayes classification is a way of **updating probabilities**.

- ***From P(c) to P(c|d)***
- *P*(*c*) is called the ***prior probability*** of *c*.
  - You guess a class according to your prior knowledge.
- *P*(*c*|*d*) is called the ***posterior probability*** of *c*.
  - It updates (modifies) confidence that *c* holds after we have seen *d*.

Recall the card playing example:

- Before seeing the “tip”, every card is possible (1/52).
- After seeing the “tip”, the probability distribution is updated!!



Naïve Bayes Classification (4/15)

A further look at $\underset{c \in C}{\operatorname{argmax}} P(c) \boldsymbol{P(d \mid c)}$

$P(d \mid c) = P(t_1, t_2, \ldots, t_l \mid c)$

- E.g., *P*(‘The’, ‘breakfast’, ‘is’, ‘terrible’ | FOOD)

NB classification makes a **conditional independence assumption**:

- Terms are independent of each other!!

Without independence assumption,
P(A B C) = *P*(A) * *P*(B|A) * *P*(C|A B)
Very complex!!

$$P(d \mid c) = P(t_1, t_2, \ldots, t_l \mid c)$$

$$= P(t_1 \mid c) P(t_2 \mid c) \ldots P(t_l \mid c)$$

$$= \prod_{1 \leq k \leq l} P(t_k \mid c)$$

- E.g., *P*(‘The’, ‘breakfast’, ‘is’, ‘terrible’ | FOOD)
  = *P*(‘The’|Food) *P*(‘breakfast’|Food) *P*(‘is’|Food) *P*(‘terrible’|Food)

Naïve Bayes Classification (5/15)

NB classification **ALSO** makes a **positional independence assumption**:

- The probability of seeing a term is irrelevant to the term's position in a document!!

The two assumptions are too **naïve**....

- The probability of seeing 'Once' in the beginning of a sentence (document) is certainly higher than seeing it in other positions.
- The probabilities of seeing 'Hong' and 'Kong' may not be independent!!

We just make do with the assumptions. Without them, model parameters will be too many to implement the classification method.

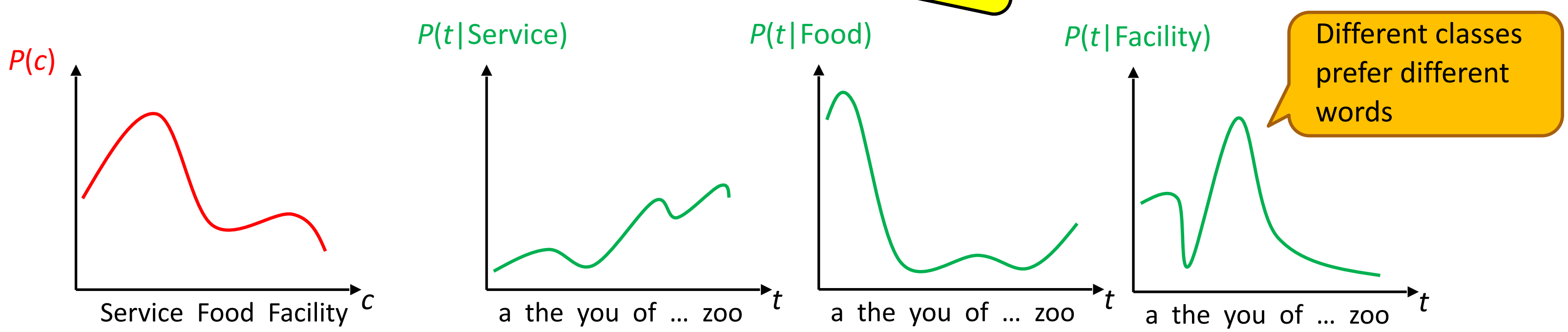
Naïve Bayes Classification (6/15)

The classification model now becomes:

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c) P(d \mid c) = \underset{c \in C}{\operatorname{argmax}} \color{red}{\boldsymbol{P(c)}} \prod_{1 \le k \le l} \color{green}{\boldsymbol{P(t_k \mid c)}}$$

To classify a document, we need to have the distributions of $P(c)$ and $P(t_k | c)$.

BUT … HOW???



Naïve Bayes Classification (7/15)

The distributions are acquired from the training data *D*.

$P(c)$ – the probability that a document belongs to category *c* (without seeing the document content).

$$P(c) = \frac{N_c}{N}$$

The number of training documents in class *c*

The number of training documents

Naïve Bayes Classification (8/15)

$P(t_k | c)$ – the probability of seeing term t_k in a document about c.

$$P(tk \mid c) = \frac{T_{kc}}{\sum_{1 \le m \le M} T_{mc}}$$

The number of occurrences of t_k in all training documents about *c*

M – the vocabulary size.
The number of terms in all training documents about *c*

Naïve Bayes Classification (9/15)

Example:

Training Phase

	review content	**About "Food"?**
Training data	i like the breakfast	Yes
	the breakfast is good	Yes
	that breakfast is terrible	Yes
	the location is perfect	No

- Vocabulary V = {i, like, the, breakfast, is, good, that, terrible, location, perfect}, M = 10.
- P("Food") = 3/4 and P("not Food") = 1/4.
- P(breakfast | "Food") = 3/12, P(good | "Food") = 1/12, …
- P(the | "not Food") = 1/4, …

Naïve Bayes Classification (10/15)

Testing Phase

- New (testing) document: “good breakfast”

- The classifier (model):
  - $$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{1 \le k \le l} P(t_k | c)$$

- For class “Food”: ¾ * 1/12 * 3/12 = **0.0156**
- For class “not Food”: ¼ * 0/4 * 0/4 = 0

- So… NB will label the document “**Food**” related.

Naïve Bayes Classification (11/15)

What about “perfect breakfast”?

- The classifier (model):
  - $$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{1 \le k \le l} P(t_k \mid c)$$
- For class “Food”: ¾ * 0/12 * 3/12 = 0
- For class “not Food”: ¼ * 1/4 * 0/4 = 0

Tie … NB cannot classify the document ☹

- Worst of all … the probabilities are all **zero**!!!

Naïve Bayes Classification (12/15)

$P(t_k \mid c)$ is zero if a term t_k never occurs in the training documents about *c*.

- In the last example, it ruins the classification no matter how strong the evidence for the class "Food" from other terms.

The **zero probability problem** is **unavoidable** because of the text sparseness phenomenon (recall **Zipf's Law**).

Don't give up … here comes the solution – **add-one smoothing**.

- Also called **Laplace smoothing**.

Naïve Bayes Classification (13/15)

The add-one smoothing:

- $$P(tk \mid c) = \frac{T_{kc} + 1}{\sum_{1 \le m \le M} (T_{mc} + 1)} = \frac{T_{kc} + 1}{\sum_{1 \le m \le M} T_{mc} + \boldsymbol{M}}$$

Add a "pseudo" count (term occurrence) to every term in vocabulary

What about "perfect breakfast" now?

- The classifier (model):
  - $$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{1 \le k \le l} P(t_k \mid c)$$
- For class "Food": ¾ * (0+1)/(12+10) * (3+1)/(12+10) = **0.006**
- For class "not Food": ¼ * (1+1)/(4+10) * (0+1)/(4+10) = 0.002

Assign to class "Food" again

Naïve Bayes Classification (14/15)

Specifically, the classification model is the **multinomial** Naïve Bayes model:

$$c_{MAP} = \underset{c\in C}{\operatorname{argmax}} P(c) \prod_{1\le k\le l} P(t_k|c)$$

$$= \underset{c\in C}{\operatorname{argmax}} P(c) \prod_{1\le m\le M} P(t_m|c)^{tf_{m,d}}$$

WHY BRING THIS TO ME!!
☹

Multinomial distribution:

- Whenever an experiment is performed, one of the **disjoint** outcomes $A_1, A_2, \dots, A_M$ will occur.
- Let $P(A_i) = p_i$, $1 \le i \le M$, and $p_1 + p_2 + \dots + p_M = 1$.
- If, in n independent performances of this experiment, Y_i, $i = 1, 2, \dots, M$, denotes the number of times that A_i occurs.
- Then, $P(Y_1=y_1, Y_2=y_2, \dots, Y_M=y_M)$, with $y_1+y_2+\dots+y_M = n$, is called *multinomial distribution*.

$$\text{P(Y1=y1, Y2=y2, …, YM=yM)} = \frac{n!}{y_1!y_2!\dots y_M!} \prod_{1\le i\le M} p_i^{y_i}$$

Naïve Bayes Classification (15/15)

Because we are going to use `SKLearn` to build a NB model for text classification.

Before showing the code, the last equation implies **a term frequency vector** of each document!!

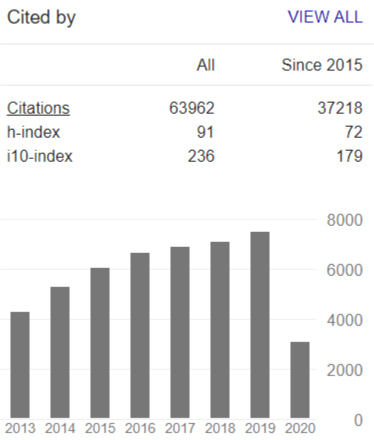
$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{1 \le m \le M} P(t_m | c)^{tf_{m,d}}$$

	t_1	t_2	…	t_{M-1}	t_M
d =	0	2	…	1	0

Coding Exercise (1/7)

`SKLearn` offers a set of APIs for the NB classification we just learned, and names it `MultinomialNB`.

Let's code it up!!



Data: ***Blog Author Gender Classification dataset***

- Arjun Mukherjee and **Bing Liu**. "Improving Gender Classification of Blog Authors." Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP-10).
- http://www.cs.uic.edu/~liub/FBS/blog-gender-dataset.rar

Coding Exercise (2/7)

The format of the Excel file

- 3227 records. Each consists of text of a blog and a gender label.
  - Male: 1679 (52%), Female: 1548 (48%)

3218	I am a sucker for a Liberty print. I still have several items, including a paisley bag, which I bought from the store when I used to live in London. So it's no surprise that I like these pretty - in a old-fashioned yet contemporary	F
3219	I love the atmosphere the olympics are stirring up in my city. I love how I can get downtown after a 20 minute bus	F
3220	Monetary values change as consciousness changes. Prosperity Consciousness is Higher Consciousness becau	M
3221	…And as the rain fell brilliantly against my window on a Saturday afternoon I noticed a pattern within the glitter o	M
3222	That's the second book in Timothy Zahn's trilogy.I bet y'all thought that I'd forgotten about you concerning our fun	M
3223	"weird"let's discuss what this word means to us. to me it's one of the most positive words i know. i think i generally	F
3224	There are two types of fall guys; one who willingly accepts responsibility for something he didn't do, to cover for t	M
3225	I like…flipping my blankets over to the cold side when I sleep, the smell of sun-dried laundry, back scratches, foc	F
3226	Alone for so long walking down the path of darkness with a lamp to guide my journey of life, the fates gust by the	M
3227	It's been more than a month since I posted anything here. There is so much to do, plus my mood has been off. E	F
3228	It was a scavenger style race with checkpoints throughout Boston and the surrounding neighborhoods. At each c	M
3229	Finally! I got a full day's work done. Almost 4k, and if my hand hadn't started hurting I would have gotten even mc	F
3230	At the height of laughter, the universe is flung into a kaleidoscope of new possibilities. ~Jean HoustonWhen peop	M
3231	I like birds, especially woodpeckers and MOST especially the huge Campephilus woodpeckers. Four years ago	M
3232	Oh friends, it's finally here! I thought the month between Christmas break and midwinter break wouldn't be too slo	F

text **Label: M/F**

Coding Exercise (3/7)

Load data from the Excel file:

Excel reader

```
In [1]: import xlrd

In [2]: workbook = xlrd.open_workbook('./blog-gender-dataset.xlsx')

In [3]: booksheet = workbook.sheet_by_name('data')

In [4]: texts = []
        labels = []

In [5]: for i in range(booksheet.nrows):
            labels.append(booksheet.cell(i,1).value)
            texts.append(booksheet.cell(i,0).value)
```

- `texts`: all text data
- `labels`: corresponding labels

Coding Exercise (4/7)

Convert the documents into frequency vectors

- Which are the input of the `MultinomialNB` classification.

```
In [7]: from sklearn.feature_extraction.text import CountVectorizer

In [8]: TF_vectorizer = CountVectorizer()

In [9]: TF_vectors = TF_vectorizer.fit_transform(texts)

In [10]: TF_vectors.shape

Out[10]: (3227, 52456)

In [11]: x_train = TF_vectors[0:2500]
         x_test = TF_vectors[2500:]
         y_train = labels[0:2500]
         y_test = labels[2500:]

In [12]: x_train.shape

Out[12]: (2500, 52456)

In [13]: x_test.shape

Out[13]: (727, 52456)
```

Split the data into training/testing parts

Training: The first 2500 instances

Testing: the remaining 727 instances

Coding Exercise (5/7)

Create a multinomial NB model

- The inputs of **`fit`** are the feature vectors and the corresponding labels

```
In [14]: from sklearn. naive_bayes import MultinomialNB

In [15]: model = MultinomialNB()

In [16]: model.fit(x_train,y_train)

Out[16]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

`alpha`: the Laplace smoothing parameter

No given class priors, and learn prior probabilities

You can set the parameters when creating the model object

Coding Exercise (6/7)

Classify new data using `predict`

```
In [17]: predicted_results = []
         expected_results = []

In [18]: expected_results.extend(y_test)

In [19]: predicted_results.extend(model.predict(x_test))

In [22]: print(predicted_results)

['F', 'F', 'M', 'F', 'M', 'M', 'F', 'F', 'F', 'F', 'F'
'F', 'M', 'F', 'F', 'F', 'M', 'F', 'F', 'M', 'M', 'F',
'M', 'F', 'F', 'F', 'F', 'M', 'M', 'F', 'F', 'F', 'M',
'F', 'M', 'F', 'F', 'M', 'F', 'F', 'F', 'M', 'M', 'F',
'F', 'F', 'M', 'F', 'F', 'F', 'M',
'F', 'F', 'F', 'F', 'F', 'F', 'F',
'F', 'F', 'M', 'M', 'F', 'F', 'M',
'F', 'F', 'M', 'M', 'M', 'M', 'F',
'F', 'F', 'F', 'F', 'F', 'F', 'F',
'F', 'F', 'F', 'F', 'F', 'F', 'F',
'M', 'M', 'F', 'F', 'F', 'F', 'F', 'M', 'F', 'F', 'M',
'M', 'F', 'F', 'M', 'F', 'M', 'M', 'M', 'F', 'F', 'F',
```

```
In [26]: model.predict_proba(x_test[2])

Out[26]: array([[0.02662801, 0.97337199]])
```

Training …

Testing …

Everything done?

NO!! We need to know how good the model is

Coding Exercise (7/7)

`Sklearn` also provides a lot of APIs for performance measurement.

```
In [20]: from sklearn import metrics

In [21]: print(metrics.classification_report(expected_results, predicted_results))

              precision    recall  f1-score   support

           F       0.66      0.83      0.74       370
           M       0.76      0.56      0.65       357

    accuracy                           0.70       727
   macro avg       0.71      0.70      0.69       727
weighted avg       0.71      0.70      0.69       727
```

Bernoulli NB Text Classification Model (1/7)

Remember the **multi-hot vector**???

- A vector element $x_{t,d}$ is 1 if term *t* is present in the document, otherwise, it is 0.

	'the'	'a'	'cake'	'tea'	…	'dog'	'pay'	'bad'
doc_1	1	1	0	0	…	1	1	1
doc_2	1	1	1	1	…	1	1	0

If you represent documents as multi-hot vectors, you are using **Bernoulli** NB model!!

Bernoulli NB Text Classification Model (2/7)

Testing – the classification model:

- $$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c \mid d)$$

$$= \underset{c \in C}{\operatorname{argmax}} \frac{P(c)P(d \mid c)}{P(d)}$$

$$= \underset{c \in C}{\operatorname{argmax}} P(c) \mathbf{P(d \mid c)}$$

$$= \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{1 \le m \le M} \mathbf{P(x_{m,d} \mid c)}$$

P(<1, 1, 0, 0, 1, …, 0, 0> | c)

Terms **not occuring** in document *d* also involve in classification decision!!

Training:

$$P(c) = \frac{N_c}{N}$$

N: the number of training documents
N_c: the number of training documents in *c*

$$P(x_{m,d} = 1 \mid c) = \frac{N_{c,m}}{N_c}$$

$$P\left(x_{m,d} = 0 \middle| c\right) = 1 - P(x_{m,d} = 1 \mid c)$$

N_c: the number of training documents in *c*
$N_{c,m}$: the number of training documents in *c* containing term *m*

Bernoulli NB Text Classification Model (3/7)

Example:

Training Phase

	review content	About "Food"?
Training data	i like the breakfast	Yes
	the breakfast is good	Yes
	that breakfast is terrible	Yes
	the location is perfect	No

- Vocabulary V = {i, like, the, breakfast, is, good, that, terrible, location, perfect}, $M = 10$.
- P("Food") = 3/4 and P("not Food") = 1/4.
- P(i | "Food") = (1+1)/(3+2), P(like | "Food") = (1+1)/(3+2), …
- P(i | "not Food") = (0+1)/(1+2), P(like | "not Food") = (0+1)/(1+2), …

Bernoulli NB Text Classification Model (4/7)

Testing Phase

- New (testing) document: “good breakfast”
- The classifier (model):
  - $$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{1 \le m \le M} P(x_{m,d} \mid c)$$
- For class “Food”: ¾ * (1-2/5) * (1-2/5) * … = **0.00318**
- For class “not Food”: ¼ * (1-1/3) * (1-1/3) * … = 6.774E-05
- The Bernoulli NB thus will label the document “Food” related.

Bernoulli NB Text Classification Model (5/7)

`Sklearn` Example:

```
In [1]: import xlrd

In [2]: workbook = xlrd.open_wor

In [3]: booksheet = workbook.she

In [4]: texts = []
        labels = []

In [5]: for i in range(booksheet
            labels.append(booksh
            texts.append(bookshe
```

```
In [6]: from sklearn.feature_extraction.text import CountVectorizer

In [7]: binary_vectorizer = CountVectorizer(binary=True)

In [8]: binary_vectors = binary_vectorizer.fit_transform(texts)
        print(binary_vectors.toarray())

        [[0 0 0 ... 0 0 0]
         [0 0 0 ... 0 0 0]
         [0 0 0 ... 0 0 0]
         ...
         [0 0 0 ... 0 0 0]
         [0 1 0 ... 0 0 0]
         [0 0 0 ... 0 0 0]]

In [9]: binary_vectors.shape

Out[9]: (3227, 52456)

In [10]: x_train = binary_vectors[0:2500]
         x_test = binary_vectors[2500:]
         y_train = labels[0:2500]
         y_test = labels[2500:]

In [11]: x_train.shape

Out[11]: (2500, 52456)

In [12]: x_test.shape

Out[12]: (727, 52456)
```

Bernoulli NB Text Classification Model (6/7)

In [13]:
```
from sklearn. naive_bayes import BernoulliNB
```

In [14]:
```
model = BernoulliNB()
```

In [15]:
```
model.fit(x_train,y_train)
```

Out[15]:
```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

In [16]:
```
predicted_results = []
expected_results = []
```

In [17]:
```
expected_results.extend(y_test)
```

In [18]:
```
predicted_results.extend(model.predict(x_test))
```

In [19]:
```
print(predicted_results)
```

```
['F', 'F', 'F', 'F', 'M', 'M', 'F', 'F', 'F', 'F', 'F',
'F', 'M', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'M', 'F',
'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F',
'F', 'M', 'F', 'F', 'M', 'F', 'F', 'F', 'M', 'F', 'F',
'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'M', 'M',
'F', 'F', 'F', 'F', 'F', 'F', 'F', 'M', 'F', 'F', 'F',
'F', 'F', 'M', 'M', 'F', 'F', 'F', 'F', 'M', 'F', 'F',
'F', 'F', 'M', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'M',
```

Bernoulli NB Text Classification Model (7/7)

```
In [20]: from sklearn import metrics

In [21]: print(metrics.classification_report(expected_results, predicted_results))
              precision    recall  f1-score   support

           F       0.59      0.89      0.71       370
           M       0.76      0.36      0.49       357

    accuracy                           0.63       727
   macro avg       0.67      0.62      0.60       727
weighted avg       0.67      0.63      0.60       727
```

The result is much worse than that of the multinomial NB model … WHY???

- The Bernoulli model uses binary occurrence information.
- Ignore the number of occurrences.
- **Typically make many mistake when classifying long documents.**

Properties of NB classification

The assumptions of NB classification are so naïve … but the classifiers usually have good classification performance.

Also, both training and testing of NB classification are efficient.

The **efficiency** and **effectiveness** of NB classification are reasons why it is a popular text classification method.

- Usually as a **baseline** in text classification research.

K-fold Cross Validation

```
In [1]: import xlrd

In [2]: workbook = xlrd.open_workbook('./blog-gender-dataset.xlsx')

In [3]: booksheet = workbook.sheet_by_name('data')

In [4]: texts = []
        labels = []

In [5]: for i in range(booksheet.nrows):
            labels.append(booksheet.cell(i,1).value)
            texts.append(booksheet.cell(i,0).value)
```

```
In [6]: from sklearn.feature_extraction.text import CountVectorizer

In [7]: TF_vectorizer = CountVectorizer()

In [8]: TF_vectors = TF_vectorizer.fit_transform(texts)

In [9]: TF_vectors.shape

Out[9]: (3227, 52456)

In [10]: x_train = TF_vectors[0:2500]
         x_test = TF_vectors[2500:]
         y_train = labels[0:2500]
         y_test = labels[2500:]
```

```
In [11]: from sklearn.model_selection import cross_val_score
         from sklearn. naive_bayes import MultinomialNB

         model = MultinomialNB()

In [12]: scores = cross_val_score(model, x_train, y_train, cv=10, scoring='f1_macro')

In [13]: scores.mean()

Out[13]: 0.6709620232259874
```

ROC Curve (1/2)

```
In [13]: from sklearn.naive_bayes import MultinomialNB

In [14]: model = MultinomialNB()

In [15]: model.fit(x_train,y_train)

Out[15]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

In [16]: y_probs = model.predict_proba(x_test)[:,0]
```

`predict_proba`: returns the classification probabilities for each testing instance

```
In [17]: from sklearn.metrics import roc_curve
         from sklearn.metrics import auc

In [18]: fpr, tpr, _ = roc_curve(y_test, y_probs, pos_label="F")

In [19]: auc_score = auc(fpr, tpr)
```

ROC Curve (2/2)

In [20]:
```
import matplotlib.pyplot as plt
```

In [21]:
```
plt.figure(1)
plt.plot([0,1], [0,1], "k--")
plt.plot(fpr,tpr, label='test (AUC=%0.2f)' %  auc_score)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc='best')
plt.show
```

Out[21]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```