# CS5489 - Machine Learning

## Lecture 2b - Naive Bayes Classifier


### Prof. Antoni B. Chan

#### Dept. of Computer Science, City University of Hong Kong


# Outline

1. Naive Bayes Gaussian Classifier - Iris dataset
2. Gaussian Classifier - Iris dataset
3. Naive Bayes Spam Classifier - Spam dataset

# Naive Bayes Classifier

- How to deal with multiple features?
  - e.g., $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$
- **Naive Bayes assumption**
  - assume each feature dimension is modeled independently.
    - the joint probability is the product of the individual probabilities
    - e.g., for 2 dimensions, $p(x_1, x_2|y) = p(x_1|y)p(x_2|y)$
    - accumulates evidence from each feature dimension:
      - $\log p(x_1, x_2|y) = \log p(x_1|y) + \log p(x_2|y)$
  - allows us to model each dimension of the observation with a simple univariate distribution.
- **Example: Naive Bayes Gaussian classifier**
  - We will consider the 2-dimensional iris data shown in the beginning of lecture.

# Setup Python

In [1]:
```
%matplotlib inline
import matplotlib_inline   # setup output image format
matplotlib_inline.backend_inline.set_matplotlib_formats('retina')
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 100  # display larger images
import matplotlib
from mpl_toolkits import mplot3d
from numpy import *
from sklearn import *
from scipy import stats
random.seed(100)          # specify a seed so results are reproducible
```

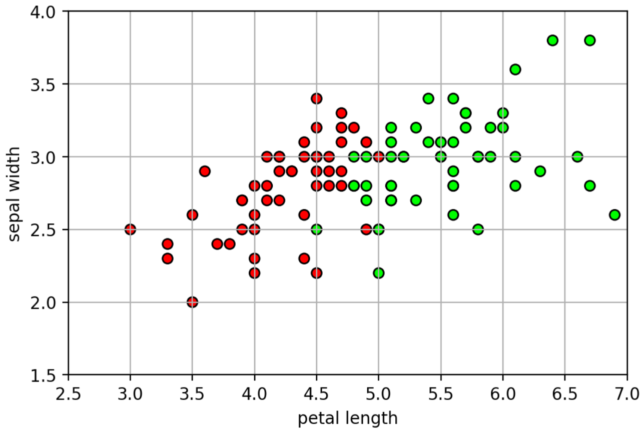
# Load data

In [2]:
```
# load iris data each row is (petal length, sepal width, class)
irisdata = loadtxt('iris2.csv', delimiter=',', skiprows=1)

X = irisdata[:,0:2]  # the first two columns are features (petal length, sepal width)
Y = irisdata[:,2]    # the third column is the class label (versicolor=1, virginica=2)
Y = Y.astype('int')  # convert to integer
print(X.shape)
```

```
(100, 2)
```

# View data

In [4]:
```
# show the data
plt.figure()
plt.scatter(X[:,0], X[:,1], c=Y, cmap=mycmap, edgecolors='k')
# c is the color value, drawn from colormap mycmap
irisaxis()
```



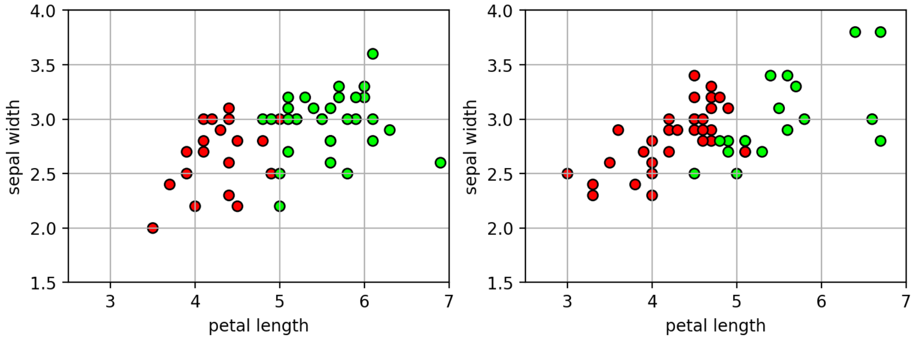

# Split training/test data

- We will select 50% of the data for training, and 50% for testing
  - use `model_selection` module
    - `train_test_split` - give the percentage for training and testing.
    - `StratifiedShuffleSplit` - also preserves the percentage of examples for each class.

In [5]:
```
# randomly split data into 50% train and 50% test set
trainX, testX, trainY, testY = \
  model_selection.train_test_split(X, Y,
     train_size=0.5, test_size=0.5, random_state=4487)

print(trainX.shape)
print(testX.shape)
```

```
(50, 2)
(50, 2)
```

In [6]:
```
# view train & test data
plt.figure(figsize=(9,3))

plt.subplot(1,2,1)    # put two subplots in the same figure
# scatter plot - Y value selects the color
plt.scatter(trainX[:,0], trainX[:,1], c=trainY, cmap=mycmap, edgecolors='k')
irisaxis()

plt.subplot(1,2,2)
plt.scatter(testX[:,0], testX[:,1], c=testY, cmap=mycmap, edgecolors='k')
irisaxis()
```




# Learn Gaussian NB model

- treat each feature dimension as an independent Gaussian
- class conditionals densities:
  - $p(\mathbf{x}|y=c) = \mathcal{N}(x_1|\mu_{c,1},\sigma_{c,1}^2)\mathcal{N}(x_2|\mu_{c,2},\sigma_{c,2}^2)$
  - each dimension $j$ has its own mean $\mu_{c,j}$ and variance $\sigma_{c,j}^2$ for class $c$.
- $\mathcal{N}(x|\mu,\sigma^2)$ is a Gaussian with mean $\mu$ and variance $\sigma^2$.

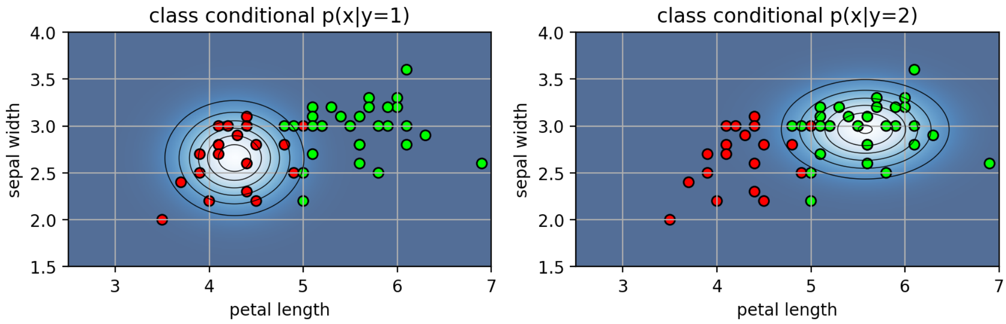
In [7]:
```
# get the NB Gaussian model from sklearn
model = naive_bayes.GaussianNB()

# fit the model to training data
model.fit(trainX, trainY)

# see the parameters
print("class prior: ",  model.class_prior_)
print("class 1 mean: ", model.theta_[0,:])
print("class 1 var:  ", model.var_[0,:])
print("class 2 mean: ", model.theta_[1,:])
print("class 2 var:  ", model.var_[1,:])
```

```
class prior:  [0.38 0.62]
class 1 mean:  [4.26842105 2.65789474]
class 1 var:   [0.14426593 0.09927978]
class 2 mean:  [5.57741935 2.96451613]
class 2 var:   [0.22045786 0.07777315]
```

- View 2d class conditionals:
  - $p(x_1,x_2|y=c) = \mathcal{N}(x_1|\mu_{c,1},\sigma_{c,1}^2)\mathcal{N}(x_2|\mu_{c,2},\sigma_{c,2}^2)$
- the NB Gaussian defines a "hill" of probability, whose contours are concentric ellipses.
  - ellipses are aligned with the axes.

In [10]:
```
ccd
```

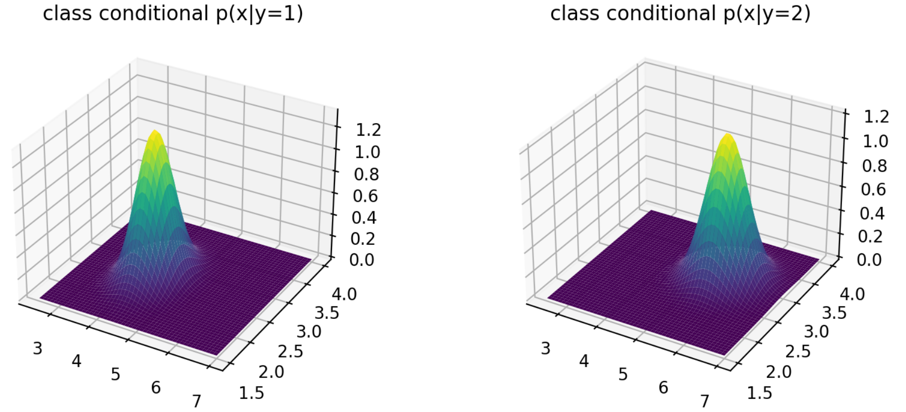
Out[10]:




- 3d surface plots

In [12]:
```
ccd3d
```

Out[12]:

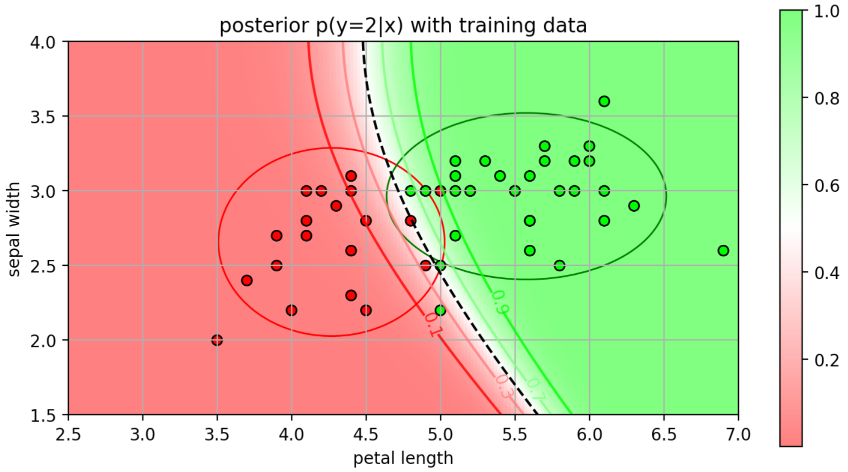



# View the Posterior

- the posterior probability decreases near the class boundary due to the uncertainty in the prediction.
- the ellipses are the contours of the CCD.

In [16]:
```
pfig
```

Out[16]:




# Evaluate on the test set

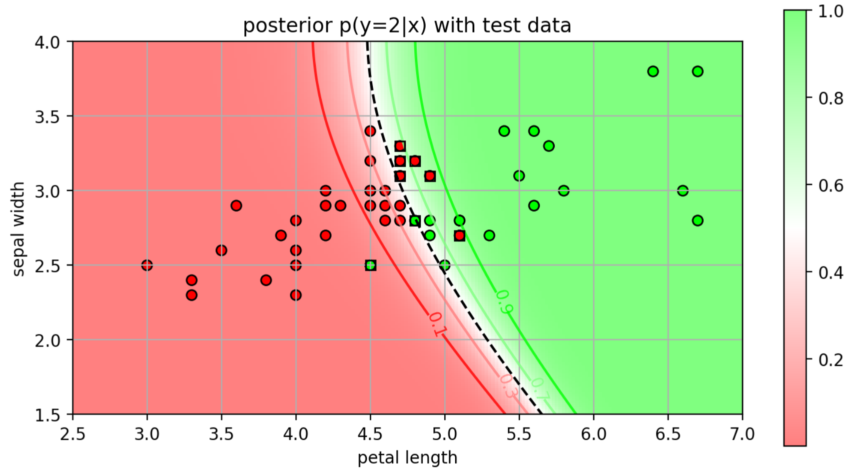
In [17]:
```
# predict from the model
predY = model.predict(testX)
print("pred: ", predY)
print("true: ", testY)

# calculate accuracy
acc = metrics.accuracy_score(testY, predY)
print("accuracy=", acc)
```

```
pred:  [2 2 1 2 1 2 1 1 1 2 2 2 2 2 1 2 2 1 2 1 1 1 1 1 1 1 1 2 1 2 1 2 2 1 1 1
 1 1 1 2 1 2 2 2 1 2 2 1 1]
true:  [2 2 1 2 1 2 1 1 1 1 1 2 2 2 2 1 2 2 1 1 1 1 2 1 1 1 1 1 2 1 2 1 2 2 1 1 2
 1 1 1 1 1 1 2 2 1 1 2 1 1]
accuracy= 0.84
```

# Viewing test results

In [19]:
```
tfig
```

Out[19]:




# NB Assumption

- NB Gaussian assumes the features are independent
  - the features do not vary together
    - e.g., knowing one feature tells us nothing about the other
  - in the iris data, the features for class 1 seem to vary together.
    - e.g., larger petal length $\longrightarrow$ larger sepal width
- How to model covariance between features?
  - need to remove the NB assumption, and model the distribution of feature vectors $\mathbf{x}$.

- Multivariate Gaussian:
  - $\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}\|\mathbf{x}-\mu\|^2_{\mathbf{\Sigma}}}$
    - parameters: mean $\mu$, covariance matrix $\mathbf{\Sigma}$.
    - Mahalanobis distance: $|\mathbf{x} - \mathbf{\mu}|^2_{\mathbf{\Sigma}} = (\mathbf{x} -

\mathbf{\mu})^T\mathbf{\Sigma}^{-1} (\mathbf{x} - \mathbf{\mu})$ is a weighted distance according to \mathbf{\Sigma}. - Determinant : $|\mathbf{\Sigma}|$ is the determinant of \mathbf{\Sigma}$, corresponds to the "volume" defined by the matrix.

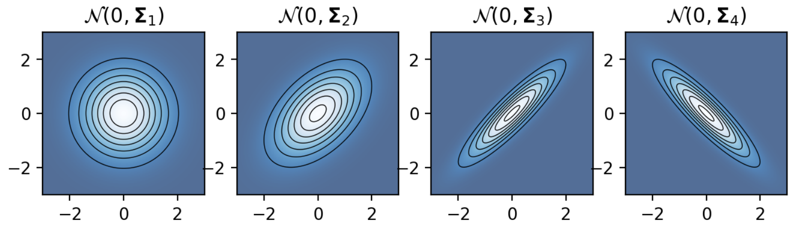
- Parameters
  - mean vector $\mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_d \end{bmatrix}$, $\mu_j$ is the mean for the $j$-th feature.
  - covariance matrix $\begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{d1} & \sigma_{d2} & \cdots & \sigma_d^2 \end{bmatrix}$
    - $\sigma_j^2$ is the variance of the j-th feature.
    - $\sigma_{ij}$ is the covariance between the i-th and j-th features.
      - if $(i, j)$ feature pair varies in the same direction, then $\sigma_{ij} > 0$.
      - if $(i, j)$ feature pair varies in opposite directions, then $\sigma_{ij} < 0$.
      - if $i, j$ features are independent (don't vary together), then $\sigma_{ij} = 0$

- 2D examples:

$$\mathbf{\Sigma}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{\Sigma}_2 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}, \mathbf{\Sigma}_3 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}, \mathbf{\Sigma}_4 = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 1 \end{bmatrix}$$

In [21]:
```
mvnfig
```

Out[21]:

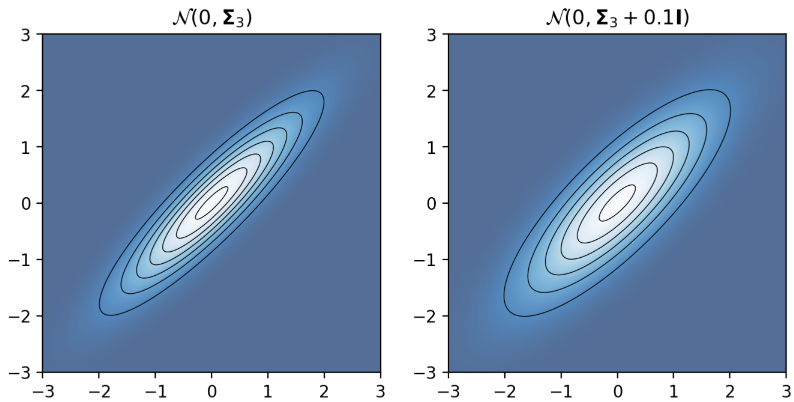



- model the class conditional density as a multivariate Gaussian distribution:
  - $p(\mathbf{x}|y=c) = \mathcal{N}(\mathbf{x}|\mu_c, \mathbf{\Sigma}_c)$
- Estimate the parameters with MLE:
  - Given the samples $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$.
    - $\mu = \frac{1}{N}\sum_{i=1}^N \mathbf{x}_i \Rightarrow$ sample mean
    - $\mathbf{\Sigma} = \frac{1}{N}\sum_{i=1}^N (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T \Rightarrow$ sample covariance

- regularize the covariance by adding a constant to the diagonal.
  - $\mathbf{\Sigma} \leftarrow \mathbf{\Sigma} + \alpha\mathbf{I}$
  - expands the Gaussian outwards in all directions, prevents collapsing on a single point.

In [23]:
```
mvnfig2
```

Out[23]:




- Create class for Gaussian Bayes Classifier

In [24]:
```
# multivariate Gaussian functions
from scipy.stats import multivariate_normal as mvn
from scipy.special import logsumexp

class GaussianBayes:
    # constructor:
    #  alpha is the regularizer on the covariance matrix: Sigma + alpha*I
    def __init__(self, alpha=0.0):
        self.alpha = alpha

    # Fit the model: assumes classes are [0,1,2,...K-1]
    # K is the max value in y
    def fit(self, X, y):
        # get the number of classes
        K = max(y)+1
        self.K = K

        # estimate mean and covariance
        self.mu = []
        self.Sigma = []
        for c in range(K):
            Xc = X[y==c]   # select samples for this class
            # estimate the mean and covariance
            self.mu.append( mean(Xc, axis=0) )
            self.Sigma.append( cov(Xc, rowvar=False) + self.alpha*eye(len(Xc[0])) )
```

```python
        # estimate class priors
        tmp = []
        for c in range(K):
            tmp.append( count_nonzero(y==c) ) # number of Class c
        self.pi = array(tmp) / len(y)  # divide by the total

    # compute the log CCD for class c, log p(x|y=c)
    def compute_logccd(self, X, c):
        lx = mvn.logpdf(X, mean=self.mu[c], cov=self.Sigma[c])
        return lx

    # compute the joint log-likelihood: log p(x,y)
    def compute_logjoint(self, X):
        # compute log joint likelihood: log p(x|y) + log p(y)
        jl = []
        for c in range(self.K):
            jl.append( self.compute_logccd(X, c) + log(self.pi[c]) )

        # p[i,c] = log p(X[i]|y=c)
        p = stack( jl, axis=-1 )
        return p

    # compute the posterior log-probability of each class given X
    def predict_logproba(self, X):
        lp = self.compute_logjoint(X)  # compute joint loglikelihoods
        lpx = logsumexp(lp, axis=1)    # compute log p(x) = log sum_c exp( log p(x,y))
        return lp - lpx[:,newaxis]     # compute log posterior: log p(y|x) = log p(x,y) - lo

    # compute the posterior probability of each class given X
    def predict_proba(self, X):
        return exp( self.predict_logproba(X) )

    # compute the most likely class given X
    def predict(self, X):
        lp = self.compute_logjoint(X)   # compute joint likelihoods
        c = argmax(lp, axis=1)          # find the maximum
        return c                        # return the class label
```

- fit the Gaussian classifier

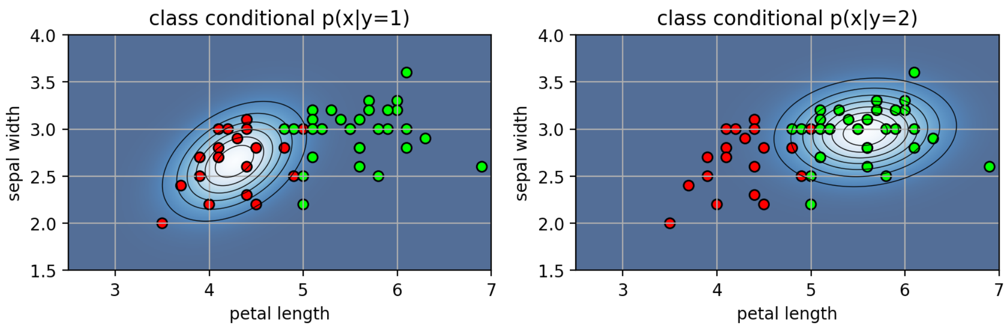
In [25]:
```python
gb = GaussianBayes()
gb.fit(trainX, trainY-1)  # map from 1...2 to 0...1
print(gb.mu)
print(gb.Sigma)
```

```
[array([4.26842105, 2.65789474]), array([5.57741935, 2.96451613])]
[array([[0.1522807 , 0.05081871],
       [0.05081871, 0.10479532]]), array([[0.22780645, 0.01650538],
       [0.01650538, 0.08036559]])]
```

- CCDs for each class
  - the contours of the Gaussian are tilted with the data
    - thus, the features are covarying.

In [27]:
```python
ccd2
```

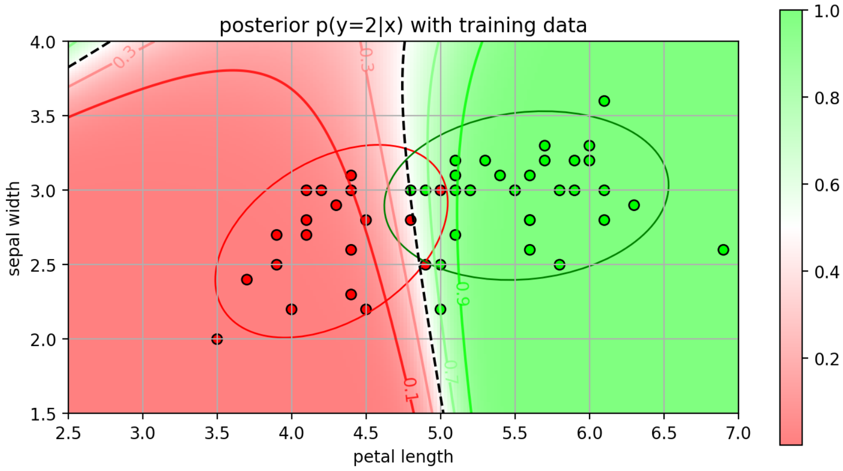
Out[27]:



- Posterior probability for the Gaussian classifier
  - the boundary better separates the data

In [29]:
```
p2fig
```

Out[29]:




- test accuracy is better than NB Gaussian
  - m.v. Gaussian is a better choice for the CCD for this dataset.

In [30]:
```
# predict from the model
predY = gb.predict(testX)+1    # map from 0..1 to 1..2
print("pred: ", predY)
print("true: ", testY)

# calculate accuracy
acc = metrics.accuracy_score(testY, predY)
print("accuracy=", acc)
```

```
pred:  [2 2 1 2 1 2 1 1 1 1 2 2 2 2 2 1 2 2 1 2 1 1 1 1 1 1 1 1 2 1 2 1 2 2 1 1 1
 1 1 1 1 1 2 2 2 1 1 2 1 1]
true:  [2 2 1 2 1 2 1 1 1 1 1 2 2 2 2 1 2 2 1 1 1 1 2 1 1 1 1 1 2 1 2 1 2 2 1 1 2
 1 1 1 1 1 1 2 2 1 1 2 1 1]
accuracy= 0.9
```

# Example: Naive Bayes Spam Classifier

- Goal: given an input email, predict whether it is spam or not
  - input: text string

> A home based business opportunity is knocking at your door. Don't be rude and let this chance go by. You can earn a great income and find your financial life transformed. Learn more Here. To Your Success. Work From Home Finder Experts

- output: spam, not spam (or ham)

# Text Document Representation

- Text document is a string!
  - we need to pick a suitable representation.
- **Bag-of-Words (BoW) model**
  - Let $\mathcal{V} = \{w_1, w_2, \cdots w_V\}$ be a list of $V$ words (called a **vocabulary**).
  - represent a text document as a vector $\mathbf{x} \in \mathbb{R}^V$.
    - each entry $x_j$ represents the number of times word $w_j$ appears in the document.
- **Example:**

- Document: "This is a test document"
- Vocabulary: $\mathcal{V} = \{"\text{this}", "\text{test}", "\text{spam}", "\text{foo}"\}$

- **NOTE:**
  - the order of the words is not used!
  - rearranging words leads to the same representation!
- Example:
  - "this is spam" $\rightarrow \mathbf{x} = [1, 0, 1, 0]$
  - "is this spam" $\rightarrow \mathbf{x} = [1, 0, 1, 0]$
- This is why it is called "bag-of-words"

# Steps to make BoW

1. Build a vocabulary $\mathcal{V}$.

- remove *stopwords*
  - the most common words that provide little information
  - examples: "the", "a", "on"
- convert to all lower case

2. Calculate the vector for each document

- count the occurence of each word in the vocabulary

In [31]:
```
# Load text data from directories
#   each sub-directory contains text files for 1 class
textdata = datasets.load_files("email", encoding="utf8", decode_error="replace")

# target names
print("class names = ", textdata.target_names)
print("classes = ", unique(textdata.target))
print("num samples = ", len(textdata.target))
```

```
class names =  ['ham', 'spam']
classes =  [0 1]
num samples =  50
```

In [32]:
```
# look at first sample
print("Sample 1 is Class " + str(textdata.target[0]) + \
  "(" + textdata.target_names[textdata.target[0]] + ")")
print("---")
print(textdata.data[0])
```

```
Sample 1 is Class 1(spam)
---
Get Up to 75% OFF at Online WatchesStore

Discount Watches for All Famous Brands

* Watches: aRolexBvlgari, Dior, Hermes, Oris, Cartier, AP and more brands
* Louis Vuitton Bags & Wallets
* Gucci Bags
* Tiffany & Co Jewerly

Enjoy a full 1 year WARRANTY
```

```
Shipment via reputable courier: FEDEX, UPS, DHL and EMS Speedpost
You will 100% recieve your order
```

In [33]:
```
# randomly split data into 50% train and 50% test set
traintext, testtext, trainY, testY = \
  model_selection.train_test_split(textdata.data, textdata.target,
  train_size=0.5, test_size=0.5, random_state=11)

print(len(traintext))
print(len(testtext))
```

```
25
25
```

In [34]:
```
# setup the document vectorizer to make BoW
# - use english stop words
# - max_features: only use the most frequent words in the dataset
#                 (remove this to use all words in documents)
cntvect = feature_extraction.text.CountVectorizer(stop_words='english', max_features=100)

# create the vocabulary
# NOTE: we only use the training data!
cntvect.fit(traintext)

# calculate the vectors for the training data
trainX = cntvect.transform(traintext)

# calculate vectors for the test data
testX  = cntvect.transform(testtext)

# print the vocabulary
# - (key,value) pairs correspond to (word,vector index)
print(cntvect.vocabulary_)
```

```
{'day': 31, 'mr': 63, 'john': 52, 'frank': 41, 'united': 92, 'nations': 65, 'repre
sentative': 78, 'states': 88, 'payment': 70, 'bank': 16, 'inheritance': 50, 'provi
de': 74, 'info': 47, 'names': 64, 'contact': 26, 'phone': 71, 'number': 68, 'addre
ss': 12, 'information': 49, 'required': 80, 'funds': 44, 'forward': 40, 'compensat
ion': 25, 'david': 30, 'email': 34, 'send': 85, 'country': 28, 'nigeria': 67, 'tod
ay': 91, 'going': 45, 'codeine': 23, '15mg': 4, '30': 7, '70': 10, '30mg': 8, 'pil
ls': 72, '60': 9, 'brand': 18, 'watson': 93, 'mg': 60, '120': 3, '10': 2, 'days':
32, 'major': 56, 'interesting': 51, 'let': 55, 'know': 54, 'thanks': 90, 'scifinan
ce': 84, 'gpu': 46, 'enabled': 35, 'pricing': 73, 'risk': 82, 'model': 62, 'sourc
e': 87, 'code': 22, 'new': 66, '20': 5, 'extended': 36, 'release': 77, 'inform': 4
8, 'york': 99, 'fund': 43, 'following': 39, 'details': 33, 'soon': 86, 'working':
98, 'right': 81, 'financial': 38, 'man': 58, 'wheeler': 94, 'office': 69, 'curren
t': 29, 'account': 11, 'chief': 19, 'ryan': 83, 'commented': 24, 'status': 89, '00
0': 1, 'andrew': 15, 'agaliofu': 14, 'regards': 76, 'just': 53, 'federal': 37, 're
public': 79, 'receive': 75, '00': 0, 'wilson': 96, 'freeviagra': 42, 'make': 57,
'choice': 20, 'cost': 27, 'adobe': 13, 'microsoft': 61, '2010': 6, 'windows': 97,
'benoit': 17, 'mandelbrot': 59, 'wilmott': 95, 'close': 21}
```

In [36]:
```
# show the vocabulary with prettier outtput
showVocab(cntvect.vocabulary_)
```

```
 0. 00                          1. 000
 2. 10                          3. 120
 4. 15mg                        5. 20
 6. 2010                        7. 30
 8. 30mg                        9. 60
10. 70                         11. account
12. address                    13. adobe
14. agaliofu                   15. andrew
16. bank                       17. benoit
18. brand                      19. chief
20. choice                     21. close
22. code                       23. codeine
24. commented                  25. compensation
26. contact                    27. cost
28. country                    29. current
30. david                      31. day
32. days                       33. details
34. email                      35. enabled
```

```
36. extended                    37. federal
38. financial                   39. following
40. forward                     41. frank
42. freeviagra                  43. fund
44. funds                       45. going
46. gpu                         47. info
48. inform                      49. information
50. inheritance                 51. interesting
52. john                        53. just
54. know                        55. let
56. major                       57. make
58. man                         59. mandelbrot
60. mg                          61. microsoft
62. model                       63. mr
64. names                       65. nations
66. new                         67. nigeria
68. number                      69. office
70. payment                     71. phone
72. pills                       73. pricing
74. provide                     75. receive
76. regards                     77. release
78. representative              79. republic
80. required                    81. right
82. risk                        83. ryan
84. scifinance                  85. send
86. soon                        87. source
88. states                      89. status
90. thanks                      91. today
92. united                      93. watson
94. wheeler                     95. wilmott
96. wilson                      97. windows
98. working                     99. york
```

In [37]:

```
# show a document vector
# sparse representation: only the non-zero entries are printed
print(trainX[0])
```

```
  (0, 12)	1
  (0, 16)	1
  (0, 26)	2
  (0, 31)	2
  (0, 40)	1
  (0, 41)	2
  (0, 44)	1
  (0, 47)	1
  (0, 49)	1
  (0, 50)	1
  (0, 52)	2
  (0, 63)	2
  (0, 64)	1
  (0, 65)	1
  (0, 68)	1
  (0, 70)	5
  (0, 71)	1
  (0, 74)	1
  (0, 78)	1
  (0, 80)	1
  (0, 88)	1
  (0, 92)	2
```

- because most of the entries are zero, the document vector is stored in "sparse" matrix format to save memory

In [38]:

```
type(trainX[0])
```

Out[38]:

```
scipy.sparse._csr.csr_matrix
```

In [39]:

```
# convert to numpy array
trainX[0].toarray()
```

Out[39]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0,
```

```
        1, 0, 0, 1, 0, 1, 1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1,
        0, 0, 1, 0, 5, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
        1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0]])
```

In [40]:
```python
# show the actual words
showVocab(cntvect.vocabulary_, trainX[0])
print("---")
print(traintext[0])
```

```
 12. (1.0000) address          16. (1.0000) bank
 26. (2.0000) contact          31. (2.0000) day
 40. (1.0000) forward          41. (2.0000) frank
 44. (1.0000) funds            47. (1.0000) info
 49. (1.0000) information      50. (1.0000) inheritance
 52. (2.0000) john             63. (2.0000) mr
 64. (1.0000) names            65. (1.0000) nations
 68. (1.0000) number           70. (5.0000) payment
 71. (1.0000) phone            74. (1.0000) provide
 78. (1.0000) representative   80. (1.0000) required
 88. (1.0000) states           92. (2.0000) united
---
Attn:Good Day,

Compliment of the day to you, my name is Mr John Frank Harmon a UNITED
NATIONS Representative here in UNITED STATES this year 2014 last
payment quarter for all outstanding payment from World Bank on overdue
contracts,inheritance and all other payment has commenced,you are to
provide the below info asap so that the payment processing can start
off.

Your full names
Contact phone number
Contact Address

The above information is required so as to go through your payment
file and start the processing of this long and overdue funds.

looking forward to hearing from you

Mr John Frank Harmon
```

- For CountVectorizer, `fit` and `transform` are also combined into one function `fit_transform`.
  - build the vocabulary from training data, and return the training document vectors.

In [41]:
```python
# setup the document vectorizer to make BoW
# - use english stop words
# - only use the most frequent 100 words in the dataset
cntvect = feature_extraction.text.CountVectorizer(stop_words='english', max_features=100)

# create the vocabulary AND compute the training vectors
trainX = cntvect.fit_transform(traintext)

# calculate vectors for the test data
testX  = cntvect.transform(testtext)
```

# Naive Bayes model for Boolean vectors

- Model each word independently
  - absence/presence of a word $w_j$ in document
  - Bernoulli distribution
    - present: $p(x_j = 1|y) = \pi_j$
    - absent: $p(x_j = 0|y) = 1 - \pi_j$
  - MLE parameters: $\pi_j = N_j/N$,
    - $N_j$ is the number of documents in class $y$ that contain word $j$.
    - $N$ is the number of documents in class $y$.

- Class-conditional distribution

$$p(x_1, \cdots, x_V | y = \text{spam}) = \prod_{j=1}^{V} p(x_j | y = \text{spam})$$

$$\log p(x_1, \cdots, x_V | y = \text{spam}) = \sum_{j=1}^{V} \log p(x_j | y = \text{spam})$$

- for a document, the log-probabilities of the words being in a spam message adds.
  - accumulate evidence over all words in the document.
  - more words that are associated with spam --> more likely the document is spam

In [42]:

```
# fit the NB Bernoulli model.
# the model automatically converts count vector into binary vector
bmodel = naive_bayes.BernoulliNB(alpha=0.0)
bmodel.fit(trainX, trainY)
```

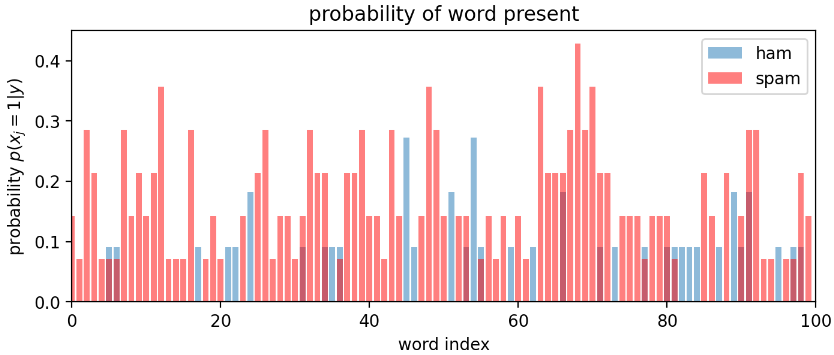
```
/Users/abc/miniforge3/envs/py39np/lib/python3.9/site-packages/sklearn/naive_bayes.
py:591: UserWarning: alpha too small will result in numeric errors, setting alpha
= 1.0e-10
  warnings.warn(
```

Out[42]:

```
▾  BernoulliNB
BernoulliNB(alpha=0.0)
```

In [45]:

```
# make plot
plotWordProb(bmodel)
plt.title('probability of word present');
```




In [46]:

```
# prediction
predY = bmodel.predict(testX)
print("predictions: ", predY)
print("actual:      ", testY)

# calculate accuracy
acc = metrics.accuracy_score(testY, predY)
print(acc)
```

```
predictions:  [0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1]
actual:       [0 1 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 1]
0.68
```

In [47]:

```
# show examples of misclassified
inds = where(predY != testY)
print(inds)
for i in inds[0]:
    print("---- true={}, pred={}".format(testY[i], predY[i]))
    print(testtext[i])
```

```
(array([ 4,  7, 12, 17, 19, 21, 22, 23]),)
---- true=0, pred=1
LinkedIn
```

Julius O requested to add you as a connection on LinkedIn:

Hi Peter.

Looking forward to the book!

Accept  View invitation from Julius O

---- true=0, pred=1
Hi Peter,

The hotels are the ones that rent out the tent. They are all lined up on the hotel
grounds : )) So much for being one with nature, more like being one with a couple
dozen tour groups and nature.
I have about 100M of pictures from that trip. I can go through them and get you jp
gs of my favorite scenic pictures.

Where are you and Jocelyn now? New York? Will you come to Tokyo for Chinese New Ye
ar? Perhaps to see the two of you then. I will go to Thailand for winter holiday t
o see my mom : )

Take care,
D

---- true=1, pred=0
Get Up to 75% OFF at Online WatchesStore

Discount Watches for All Famous Brands

* Watches: aRolexBvlgari, Dior, Hermes, Oris, Cartier, AP and more brands
* Louis Vuitton Bags & Wallets
* Gucci Bags
* Tiffany & Co Jewerly

Enjoy a full 1 year WARRANTY
Shipment via reputable courier: FEDEX, UPS, DHL and EMS Speedpost
You will 100% recieve your order
Save Up to 75% OFF Quality Watches
---- true=0, pred=1
This e-mail was sent from a notification-only address that cannot accept incoming
e-mail. Please do not reply to this message.

Thank you for your online reservation. The store you selected has located the item
you requested and has placed it on hold in your name. Please note that all items a
re held for 1 day.  Please note store prices may differ from those online.

If you have questions or need assistance with your reservation, please contact the
store at the phone number listed below. You can also access store information, suc
h as store hours and location, on the web at http://www.borders.com/online/store/S
toreDetailView_98.
---- true=1, pred=0
Get Up to 75% OFF at Online WatchesStore

Discount Watches for All Famous Brands

* Watches: aRolexBvlgari, Dior, Hermes, Oris, Cartier, AP and more brands
* Louis Vuitton Bags & Wallets
* Gucci Bags
* Tiffany & Co Jewerly

Enjoy a full 1 year WARRANTY
Shipment via reputable courier: FEDEX, UPS, DHL and EMS Speedpost
You will 100% recieve your order
---- true=1, pred=0
Get Up to 75% OFF at Online WatchesStore

Discount Watches for All Famous Brands

* Watches: aRolexBvlgari, Dior, Hermes, Oris, Cartier, AP and more brands
* Louis Vuitton Bags & Wallets
* Gucci Bags
* Tiffany & Co Jewerly

```
Enjoy a full 1 year WARRANTY
Shipment via reputable courier: FEDEX, UPS, DHL and EMS Speedpost
You will 100% recieve your order
---- true=0, pred=1
Ok I will be there by 10:00 at the latest.
---- true=0, pred=1
Hello,

Since you are an owner of at least one Google Groups group that uses the customize
d welcome message, pages or files, we are writing to inform you that we will no lo
nger be supporting these features starting February 2011. We made this decision so
that we can focus on improving the core functionalities of Google Groups -- mailin
g lists and forum discussions.  Instead of these features, we encourage you to use
products that are designed specifically for file storage and page creation, such a
s Google Docs and Google Sites.

For example, you can easily create your pages on Google Sites and share the site
(http://www.google.com/support/sites/bin/answer.py?hl=en&answer=174623) with the m
embers of your group. You can also store your files on the site by attaching files
to pages (http://www.google.com/support/sites/bin/answer.py?hl=en&answer=90563) on
the site. If you�re just looking for a place to upload your files so that your gro
up members can download them, we suggest you try Google Docs. You can upload files
(http://docs.google.com/support/bin/answer.py?hl=en&answer=50092) and share access
with either a group (http://docs.google.com/support/bin/answer.py?hl=en&answer=663
43) or an individual (http://docs.google.com/support/bin/answer.py?hl=en&answer=86
152), assigning either edit or download only access to the files.

you have received this mandatory email service announcement to update you about im
portant changes to Google Groups.
```

# Smoothing

- Some words are not present in any documents for a given class.
  - $N_j = 0$, and thus $\pi_j = 0$.
    - i.e., the document in the class **definitely** will not contain the word.
    - can be a problem since we simply may not have seem an example with that word.
- Smoothed MLE
  - add a smoothing parameter $\alpha$ that adds a "virtual" count
  - parameter: $\pi_j = (N_j + \alpha)/(N + 2\alpha)$,
  - this is called *Laplace smoothing*
- In general, *regularizing* or *smoothing* of the estimate helps to prevent *overfitting* of the parameters.

In [48]:
```
# fit the NB Bernoulli model w/ smoothing (0.1)
bmodels = naive_bayes.BernoulliNB(alpha=0.1)
bmodels.fit(trainX, trainY)
```

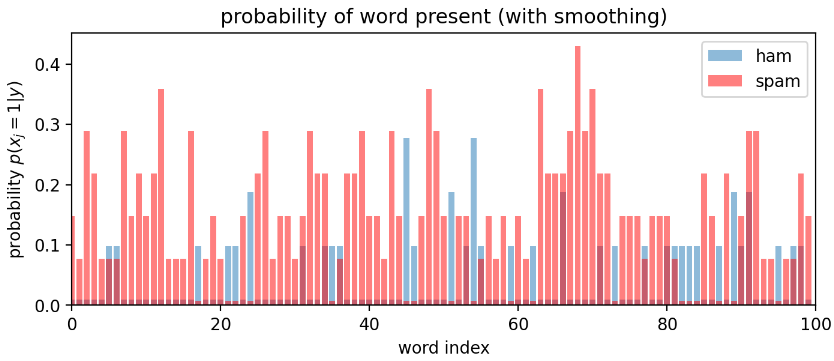
Out[48]:
```
▾ BernoulliNB
BernoulliNB(alpha=0.1)
```

In [49]:
```
# prediction
predY = bmodels.predict(testX)
print("predictions: ", predY)
print("actual:      ", testY)

# calculate accuracy
acc = metrics.accuracy_score(testY, predY)
print(acc)
# a little better!
```

```
predictions:  [0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1]
actual:       [0 1 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 1]
0.72
```

- word probabilites

In [50]:
```
# make plot
plotWordProb(bmodels)
plt.title('probability of word present (with smoothing)');
# note the small probabilities are all slightly above 0.
```




- top-10 frequent words for spam class

In [51]:
```
# get the word names
fnames = asarray(cntvect.get_feature_names_out())
# feature_log_prob_ contains the scores for each word
# sort the coefficients in ascending order, and take the 10 largest.
tmp = argsort(bmodel.feature_log_prob_[1])[-10:]

for i in tmp:
    print("{:3d}. {:15s} ({:.5f})".format(i, fnames[i], bmodel.feature_log_prob_[1][i]))
```

```
 16. bank            (-1.25276)
 67. nigeria         (-1.25276)
 26. contact         (-1.25276)
 69. office          (-1.25276)
 32. days            (-1.25276)
 48. inform          (-1.02962)
 70. payment         (-1.02962)
 63. mr              (-1.02962)
 12. address         (-1.02962)
 68. number          (-0.84730)
```

# Most informative words

- The most informative words are those with high probability of being in one class, and low probability of being in other classes.
  - e.g., For class 1, find large values of $\log p(w_j|y=1) - \log p(w_j|y=0)$

In [52]:
```
# get the word names
fnames = asarray(cntvect.get_feature_names_out())

# feature_log_prob_ contains the scores for each word
# (higher means more informative)
# calculate the log-probability difference
score = bmodel.feature_log_prob_[1] - bmodel.feature_log_prob_[0]

# sort the coefficients in ascending order, and take the 10 largest.
tmp = argsort(score)[-10:]

for i in tmp:
    print("{:3d}. {:15s} ({:.5f})".format(i, fnames[i], score[i]))
```

```
 43. fund            (24.17098)
 69. office          (24.17098)
 67. nigeria         (24.17098)
 16. bank            (24.17098)
 49. information     (24.17098)
 12. address         (24.39413)
 70. payment         (24.39413)
```

```
63. mr              (24.39413)
48. inform          (24.39413)
68. number          (24.57645)
```

# Naive Bayes for Count Vectors

- Now we consider using the number of times each word appears in the document $D$.
- Two ways to create a document vector $x$ based on the word counts.
- **Term-Frequency (TF)**
  - handles documents with different lengths (number of words).
  - normalize the count to a frequency, by dividing by the number of words in the document.
    - $x_j = \frac{w_j}{|D|}$
      - $w_j$ is the number of times word $j$ appears in the document
      - $|D|$ is the number of words in the document.
- **Term-Frequency Inverse Document Frequency (TF-IDF)**
  - some words are common among many documents
    - common words are less informative because they appear in both classes.
  - **inverse document frequency (IDF)** - measure rarity of each word
    - $IDF(j) = \log \frac{N}{N_j}$
      - $N$ is the number of documents.
      - $N_j$ is the number of documents with word $j$.
    - IDF is:
      - 0 when a word is common to all documents
      - large value when the word appears in few documents
  - **TF-IDF vector:** downscale words that are common in many documents
    - multiply TF and IDF terms
    - $x_j = \frac{w_j}{|D|}\log \frac{N}{N_j}$

In [53]:
```
# TF-IDF representation
# (For TF, pass use_idf=False)
tf_trans = feature_extraction.text.TfidfTransformer(use_idf=True, norm='l1')
# 'l1' - entries sum to 1

# setup the TF-IDF representation, and transform the training set
trainXtf = tf_trans.fit_transform(trainX)

# transform the test set
testXtf = tf_trans.transform(testX)

print(trainXtf[0])
```

```
  (0, 92)	0.05924804472389545
  (0, 88)	0.03211977983336993
  (0, 80)	0.03211977983336993
  (0, 78)	0.03533737083022898
  (0, 74)	0.03533737083022898
  (0, 71)	0.029624022361947725
  (0, 70)	0.13792420014458123
  (0, 68)	0.025860735932526913
  (0, 65)	0.03211977983336993
  (0, 64)	0.03211977983336993
  (0, 63)	0.055169680057832494
  (0, 52)	0.07067474166045797
  (0, 50)	0.03533737083022898
  (0, 49)	0.029624022361947725
  (0, 47)	0.03533737083022898
  (0, 44)	0.03533737083022898
  (0, 41)	0.07067474166045797
  (0, 40)	0.03533737083022898
```

```
(0, 31)       0.06423955966673986
(0, 26)       0.05924804472389545
(0, 16)       0.029624022361947725
(0, 12)       0.027584840028916247
```

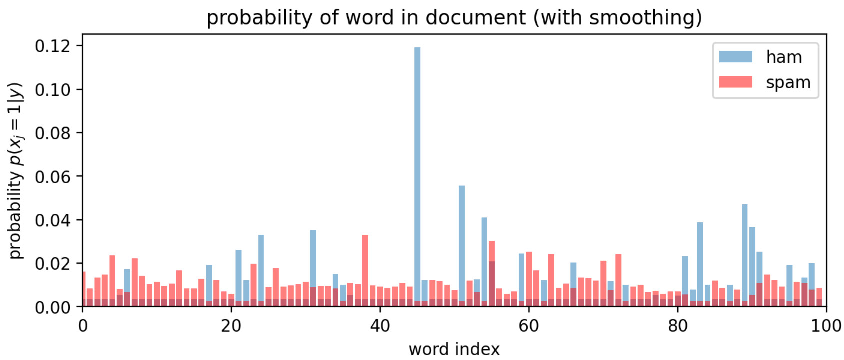
In [54]:
```
showVocab(cntvect.vocabulary_, trainXtf[0])
```

```
12. (0.0276) address          16. (0.0296) bank
26. (0.0592) contact          31. (0.0642) day
40. (0.0353) forward          41. (0.0707) frank
44. (0.0353) funds            47. (0.0353) info
49. (0.0296) information      50. (0.0353) inheritance
52. (0.0707) john             63. (0.0552) mr
64. (0.0321) names            65. (0.0321) nations
68. (0.0259) number           70. (0.1379) payment
71. (0.0296) phone            74. (0.0353) provide
78. (0.0353) representative   80. (0.0321) required
88. (0.0321) states           92. (0.0592) united
```

# Naive Bayes Multinomial

- TF or TF-IDF representation
  - Document word vector $\mathbf{x}$
    - $x_j$ is the frequency of word $j$ occuring in the document.
    - vector $\mathbf{x}$ sums to 1, i.e. $\sum_j x_j = 1$.
- Use a multinomial distribution as the class conditional
  - based on the frequency that a word appears in a document of a class.
  - $p(\mathbf{x}|y) = \frac{(\sum_j x_j)!}{\prod_j x_j!}\left(\prod_j \pi_{j,y}^{x_j}\right)$
    - $\pi_{j,y}$ = the probability that word $w_j$ occurs in class $y$.
    - $\sum_{j=1}^{V} \pi_{j,y} = 1$

In [55]:
```
# fit a multinomial model (with smoothing)
mmodel_tf = naive_bayes.MultinomialNB(alpha=0.05)
mmodel_tf.fit(trainXtf, trainY)

# show the word probabilites
plotWordProb(mmodel_tf)
plt.title('probability of word in document (with smoothing)');
```




In [56]:
```
# prediction
predYtf = mmodel_tf.predict(testXtf)
print("prediction: ", predYtf)
print("actual:     ", testY)

# calculate accuracy
acc = metrics.accuracy_score(testY, predYtf)
print(acc)
```

```
prediction:  [0 1 1 1 1 1 1 1 0 1 0 0 1 1 1 0 0 1 1 1 1 1 1 1 1]
actual:      [0 1 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 1]
0.68
```

In [57]:
```
# most frequent TF-IDF words for spam class
fnames = asarray(cntvect.get_feature_names_out())
tmp = argsort(mmodel_tf.feature_log_prob_[1])[-10:]
for i in tmp:
    print("{:3d}. {:15s} ({:.5f})".format(i, fnames[i], mmodel_tf.feature_log_prob_[1][i]))
```

```
 26. contact         (-4.03474)
 23. codeine         (-3.92301)
 70. payment         (-3.85287)
  7. 30              (-3.80167)
  4. 15mg            (-3.74496)
 72. pills           (-3.72604)
 63. mr              (-3.71758)
 60. mg              (-3.68161)
 55. let             (-3.49983)
 38. financial       (-3.40804)
```

In [58]:
```
# most informative TF-IDF for spam class
fnames = asarray(cntvect.get_feature_names_out())
score = mmodel_tf.feature_log_prob_[1] - mmodel_tf.feature_log_prob_[0]
tmp = argsort(score)[-10:]
for i in tmp:
    print("{:3d}. {:15s} ({:.5f})".format(i, fnames[i], score[i]))
```

```
 13. adobe           (1.61513)
 26. contact         (1.66904)
 23. codeine         (1.78077)
 70. payment         (1.85091)
  7. 30              (1.90211)
  4. 15mg            (1.95882)
 72. pills           (1.97775)
 63. mr              (1.98621)
 60. mg              (2.02217)
 38. financial       (2.29575)
```

# Summary

- **Generative classification model**
  - estimate probability distributions of features generated from each class.
  - given feature observation predict class with largest posterior probability.
- **Advantages:**
  - works with small amount of data.
  - works with multiple classes.
- **Disadvantages:**
  - accuracy depends on selecting an appropriate probability distribution.
    - if the probability distribution doesn't model the data well, then accuracy might be bad.

- **Other text preprocessing**
  - *Stemming*
    - convert related words into a common root word
    - example: testing, tests → "test"
    - see NLTK toolbox (http://www.nltk.org)
  - *Lemmatisation*
    - similar to stemming
    - groups inflections of word together (gone, going, went → go)
    - see NLTK
  - Removing numbers and punctuation.

- **Other word models**
  - *N-grams*
    - similar to BoW except look at pairs of consecutive words (or N consecutive words in general)

- *word vectors*
  - each word is a real vector, where direction indicates the "concept"
  - words about similar things point in the same direction
  - adding and subtracting word vectors yield new word vectors