In [1]:

```
import numpy as np
from collections import Counter

def data_loader():
    # load data
    with open('train','r') as file:
        train_data = file.read().split('\n')[:-1]
    with open('test','r') as file:
        test_data = file.read().split('\n')[:-1]
    return train_data, test_data

def parser(datum):
    # extract labels and words
    email_addr, label, words = datum.split(' ',2)
    words = words.split()
    # transform words into dictionary
    word_dict = dict(zip([words[i] for i in range(0, len(words), 2)], [int(words[i+1]) for i in ra
    # transform label into 0, 1
    if label == 'spam':
        label = 1
    else:
        label = 0
    return label, word_dict

def data_preprocessing(train_data, test_data):
    y_train = np.zeros(len(train_data))
    y_test = np.zeros(len(test_data))
    x_train = []
    x_test = []
    for i, datum in enumerate(train_data):
        label, word_dict = parser(datum)
        y_train[i] = label
        x_train.append(word_dict)
    for i, datum in enumerate(test_data):
        label, word_dict = parser(datum)
        y_test[i] = label
        x_test.append(word_dict)
    return x_train, y_train, x_test, y_test

def compute_prior(y_train):
    # compte prior distribution P(spam) and P(ham)
    ratio = Counter(y_train)
    return ratio[1]/len(y_train), ratio[0]/len(y_train)

def m_estimation_conditional_probability(x_train_frt, y_train, num_vocab, a):
    # compute P(w_j|spam) and P(w_j|ham)
    spam_idx = np.where(y_train == 1)[0]
    ham_idx = np.where(y_train == 0)[0]
    x_spam = x_train_frt[spam_idx, :]
    x_ham = x_train_frt[ham_idx, :]
    n_c = x_spam.sum(axis = 0)
    n = x_spam.sum()
    p = 1 / num_vocab
    m = num_vocab * a
    p_on_spam = (n_c + m*p) / (n+m)
    n_c = x_ham.sum(axis = 0)
    n = x_ham.sum()
    p_on_ham = (n_c + m*p) / (n+m)
    return p_on_spam, p_on_ham
```

```
def log_estimated_probability(p_spam, p_ham, p_on_spam_m, p_on_ham_m, x_frts):
    # compute log(P(spam, w_1, w_2,..., w_n)) and log(P(ham, w_1, w_2,..., w_n))
    p_spam_lookup = (x_frts > 0) * p_on_spam_m
    p_ham_lookup = (x_frts > 0) * p_on_ham_m
    p_spam_lookup[p_spam_lookup == 0] = 1
    p_ham_lookup[p_ham_lookup == 0] = 1
    log_p_spam = np.log(p_spam) + np.log(p_spam_lookup).sum(axis = 1)
    log_p_ham = np.log(p_ham) + np.log(p_ham_lookup).sum(axis = 1)
    return log_p_spam, log_p_ham

def accuarcy(y_true, y_pred):
    # calculate accuracy
    assert len(y_true) == len(y_pred)
    return (y_true==y_pred).sum()/len(y_true)
```

executed in 25ms, finished 23:23:32 2019-09-15

1 Load and preprocess data

In [2]:

```
# load data
train_data, test_data = data_loader()

# extract labels to 0,1 and features to dicticnary
x_train, y_train, x_test, y_test = data_preprocessing(train_data, test_data)
```

executed in 978ms, finished 23:23:33 2019-09-15

2 Compute prior $P(\text{spam})$ and $P(\text{ham})$

In [3]:

```
# compute_prior
p_spam, p_ham = compute_prior(y_train)
print('Prior:')
print(p_spam, p_ham)
```

executed in 15ms, finished 23:23:33 2019-09-15

```
Prior:
0.5736666666666667 0.42633333333333334
```

Transform word dicts to feature vectors

In [4]:

```
from sklearn.feature_extraction import DictVectorizer
vectorizer = DictVectorizer(sparse=False)
x_train_frt = vectorizer.fit_transform(x_train)
x_test_frt = vectorizer.transform(x_test)
```

executed in 2.31s, finished 23:23:35 2019-09-15

3 Compute $P(w_j \mid spam)$ and $P(w_j \mid ham)$ by m-estimator

In [5]:

```
p_on_spam_m, p_on_ham_m = m_estimation_conditional_probability(x_train_frt, y_train, x_train_frt.sha
```

executed in 77ms, finished 23:23:35 2019-09-15

Top 5 spam word given spam

In [6]:

```
dict(zip(np.array(vectorizer.feature_names_)[np.array(np.argsort(p_on_spam_m)[::-1]).ravel()[:5]],
         p_on_spam_m[np.array(np.argsort(p_on_spam_m)[::-1]).ravel()[:5]]))
```

executed in 18ms, finished 23:23:35 2019-09-15

Out[6]:

```
{'enron': 0.0381943878447375,
 'a': 0.023618529446035274,
 'corp': 0.021737909849797 96,
 'the': 0.02142517760233378,
 'to': 0.019687038335056983}
```

Top 5 spam word given ham

In [7]:

```
dict(zip(np.array(vectorizer.feature_names_)[np.array(np.argsort(p_on_ham_m)[::-1]).ravel()[:5]],
         p_on_ham_m[np.array(np.argsort(p_on_ham_m)[::-1]).ravel()[:5]]))
```

executed in 11ms, finished 23:23:35 2019-09-15

Out[7]:

```
{'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa': 0.0
44321708617828505,
 'enron': 0.042443723793719 36,
 'the': 0.0331690756387682,
 'to': 0.025329676593911908,
 'a': 0.017736664090903333}
```

4 Predict and validation

Comparing $P(spam|w_1, w_2, \cdots, w_n)$ and $P(ham|w_1, w_2, \cdots, w_n)$ is equivalent to comparing $P(spam, w_1, w_2, \cdots, w_n)$ and $P(ham, w_1, w_2, \cdots, w_n)$. Therefore, $P(spam, w_1, w_2, \cdots, w_n)$ and $P(ham, w_1, w_2, \cdots, w_n)$ are compared to tell whether the email is spam or ham.

In [8]:

```
# compute log(P(spam, w_1, w_2,..., w_n)) and log(P(ham, w_1, w_2,..., w_n))
log_p_spam,log_p_ham = log_estimated_probability(p_spam, p_ham, p_on_spam_m, p_on_ham_m, x_test_frt)
test_pred = (log_p_spam > log_p_ham)
# compute accuracy
accuarcy(y_test, test_pred)
```

executed in 48ms, finished 23:23:35 2019-09-15

Out[8]:

0.908

5 Grid search for the best m

In [9]:

```
def pipeline(x_train_frt, y_train, x_test_frt, y_test, a):
    p_spam, p_ham = compute_prior(y_train)
    p_on_spam_m, p_on_ham_m = m_estimation_conditional_probability(x_train_frt, y_train, x_train_frt
    log_p_spam,log_p_ham = log_estimated_probability(p_spam, p_ham, p_on_spam_m, p_on_ham_m, x_test_
    test_pred = (log_p_spam > log_p_ham)
    print(str(a) + ":" + str(accuarcy(y_test, test_pred)))
```

executed in 10ms, finished 23:23:35 2019-09-15

In [10]:

```
a_grid = [1, 10,100,1000,10000]
for a in a_grid:
    pipeline(x_train_frt, y_train, x_test_frt, y_test, a)
```

executed in 685ms, finished 23:23:36 2019-09-15

1:0.908
10:0.911
100:0.916
1000:0.863
10000:0.778

We have the highest accuarcy at $m = 100$. For m small, the impact of prior is weak, $P(w_j \mid spam)$ are dominated by n_c/n. This might leads to easy overfit. For m large, the impact of prior is strong, $P(w_j \mid spam)$ dominated by p. In this case, different word won't have different impact on the final decision, which may leads to underfit. Therefore, m can be neither too larger nor to small, and our experiment also indicate that $m = 100$ is a good hyperparameter.

6 How to beat the classifier?

I will try to paraphrase words with high $P(w_j \mid spam)$ and low $P(w_j \mid ham)$ in the email with some other words with low $P(w_j \mid spam)$ or high $P(w_j \mid ham)$. If the core idea of the email made some words with high $P(w_j \mid spam)$ or low $P(w_j \mid ham)$ inevtiable. I would add redundant sentences with words have low $P(w_j \mid spam)$ or high $P(w_j \mid ham)$ to weaken the effect of bad words.

In [ ]:

```
```