Naive Bayes and Gaussian Bayesian Inference

Thomas Schwarz

Conditional Probability

- Given two events A and B, we define the conditional probability as

$$P(A\,|\,B) = \frac{P(A \cap B}{P(B)}$$

"probability of A given B"

- Write also as:

$$P(A \cap B) = P(A\,|\,B)P(B)$$

Conditional Probability

- Bayes' Theorem: An observation of extreme importance
  - Giving rise to a new way of statistics

  Theorem: $$P(A\,|\,B) = \frac{P(B\,|\,A) \cdot P(A)}{P(B)}$$

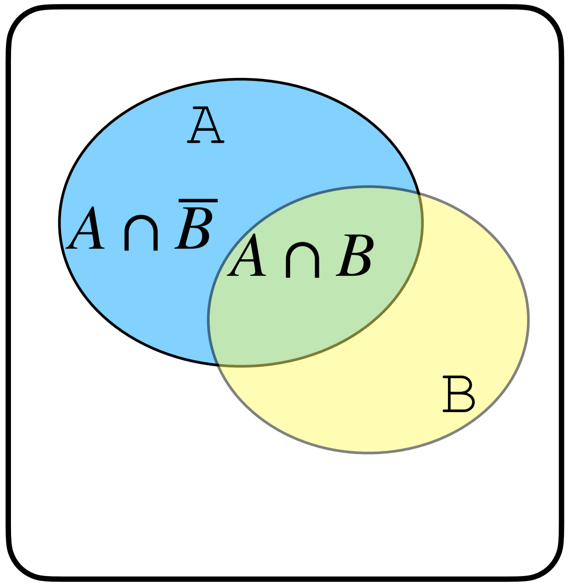
- Expresses a probability conditioned on B in one conditioned on A
- Proof:
  $$P(A\,|\,B)P(B) = P(A \cap B) = P(B \cap A) = P(B\,|\,A)P(A)$$
- Now solve for $P(A\,|\,B)$

Conditional Probability

- We can express a probability for one event in terms of another event happening or not

$$P(A) = P(A \cap B) + P(A \cap \overline{B})$$

$$= P(A \mid B)P(B) + P(A \mid \overline{B})P(\overline{B})$$



Conditional Probability

- We can expand Bayes by calculating $P(B)$ as probabilities conditioned on A

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

$$= \frac{P(B \mid A) \cdot P(A)}{P(B \cap A) + P(B \cap \bar{A})}$$

$$= \frac{P(B \mid A) \cdot P(A)}{P(B \mid A)P(A) + P(B \mid \bar{A})P(\bar{A})}$$

Conditional Probability

- Example: Medical Tests
  - An HIV test is positive. What is the probability that you have HIV?
  - Need some data: The quality of the test
    - Type 1 error: Test is negative, but there is illness
    - Type 2 error: Test is positive, but there is no illness

Conditional Probability

- Abbreviate probabilities
  - T : Test is positive
  - H : Person infected with HIV
  - Interested in $P(H \mid T)$. The quality of the test is expressed in terms of the opposite conditional probability.
    - Type I error probability: $P(\overline{T} \mid H)$
    - Type II error probability: $P(T \mid \overline{H})$

Conditional Probability

- We calculate

$$P(H \mid T) = \frac{P(T \mid H)P(H)}{P(T \mid H)P(H) + P(T \mid \overline{H})P(\overline{H})}$$

- Assume test has 5% type I (false positive) error probability and 1% type II (false negative) error probability:

$$P(T \mid \overline{H}) = 0.95$$

$$P(T \mid H) = 0.99$$

- The probability still depends on the prevalence of HIV in the population

Conditional Probability

$$P(H|T) = \frac{0.99P(H)}{0.99P(H) + 0.95(1 - P(H))}$$

- Example: HIV rate in general population in the US is 13.3/100000 = 0.000,133
- After a positive test:
  - 0.000138599 (Almost no change!)
- Example 2: HIV in a high risk group in the US is 1,753.1/100000 = 0.017531
- After a positive test:
  - 0.0182557

Conditional Probability

- With these type I and type II error rates
    - the test is almost unusable at low incidence rates

Classification with Bayes

- Bayes' theorem inverts conditional probabilities
- Can use this for classification based on observations
- Idea: Assume we have observations $\overrightarrow{x}$
  - We have calculated the probabilities of seeing these observations given a certain classification
  - I.e.: for each category, we know $P(\overrightarrow{x}, c_i)$
    - Probability to observe $\overrightarrow{x}$ assuming that point lies in c_i
  - We use Bayes formula in order to calculate $P(c_i, \overrightarrow{x})$
  - And then select the category with highest probability

Classification with Bayes

- Document classification:
  - Spam detection:
    - Is email spam or ham?
  - Sentiment analysis:
    - Is a review good or bad

Classification with Bayes

- Bag of words method:
  - Model a document by only counting words
    - Restrict ourselves to non-structure = non-common words

"I love this movie! It's sweet, but with satirical humor. The dialogs are great and the adventure scenes are fun. It manages to be romantic and whimsical while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I have seen it several times and I'm always happy to see it again"

fun	1
great	2
happy	1
humor	1
love	1
recommend	1
satirical	1
sweet	1

Classification with Bayes

- There is a whole theory about recognizing key-words automatically
  - Easy out:
    - Use all words that are not common

Classification with Bayes

- Recognizing words
  - Actual documents have misspelling and grammatical forms
    - Grammatical forms less common in English but typical in other languages
      - Lemmatization: Recognize the form of the word
        - `जाओ, जाओगे, … –> जाना`
        - `went, goes –> to go`
    - Usually difficult to automatize

Classification with Bayes

- Recognizing words
  - Stemming
    - Several methods to automatically extract the stem
      - English: Porter stemmer (1980)
      - Other languages: Can use similar ideas
      - https://www.emerald.com/insight/content/doi/10.1108/00330330610681295/full/pdf?title=the-porter-stemming-algorithm-then-and-now

Classification with Bayes

- Need to calculate the probability to observe a set of keywords given a classification
  - This is too specific:
    - There are too many sets of keywords
- First reduction:
  - Only use existence of words.

Classification with Bayes

- Want: $P(w_1, w_2, w_3, \ldots, w_n \,|\, c_i)$
  - The probability to find a certain word in documents of a certain category depends on the existence of other words.
    - E.g.: "Malicious Compliance"
  - We make now a big assumptions:
    - The probabilities of a keyword showing up are independent of each other
    - That's why this method is called "***<u>Naïve</u> Bayes***"

Classification with Naïve Bayes

- Want:

$$P(w_1, w_2, w_3, \ldots, w_n \mid c_i) = P(w_1 \mid c_i) \times P(w_2 \mid c_i) \times P(w_3 \mid c_i) \times \ldots P(w_n \mid c_i)$$

- Can estimate this from a training set:
  - E.g. a set of movie reviews classified with the sentiment
  - Algorithm:

```
for document in set:
    sentiment = document.sentiment
    for word in document:
        count[word]+=1
        if sentiment=='positive':
            countPos[word]+=1
        else:
            countNeg[word]+=1
    return countPos/count, countNeg/count
```

Classification with Naïve Bayes

- This algorithm has a problem:
  - It can return a probability as zero
    - Because we use multiplication in our estimator:

$$P(w_1, w_2, w_3, \ldots, w_n \mid c_i) = P(w_1 \mid c_i) \times P(w_2 \mid c_i) \times P(w_3 \mid c_i) \times \ldots P(w_n \mid c_i)$$

    - Would create zero probabilities
  - Solution: start all counts at 1
    - No more zero probabilities

Classification with Naïve Bayes

- Result: Simple classifier

Classification with Naïve Bayes

- Example: Use NLTK, a natural language processor
  - NLTK has several corpus (which you might have to download separately)

```
import nltk
from nltk.corpus import movie_reviews
import random
```

Classification with Naïve Bayes

- First step: Get the documents

```
documents = [(list(movie_reviews.words(fileid)), category)
              for category in movie_reviews.categories()
              for fileid in movie_reviews.fileids(category)]
random.shuffle(documents)
train_set, test_set = featuresets[500:], featuresets[:500]
```

Classification with Naïve Bayes

- Second step: Get all "features" (important words)
- Strategy: Get a list of all words, then order it, then select the frequent ones with exception of the most frequent ones.

```
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
word_features = list(all_words)[200:2000]
```

- Here is all_words:
    - `FreqDist({',': 77717, 'the': 76529, '.': 65876, 'a': 38106, 'and': 35576, 'of': 34123, 'to': 31937, "'": 30585, 'is': 25195, 'in': 21822, …})`
- Therefore, just drop the first ones.

Classification with Naïve Bayes

- Create a bag of words for each document

```
def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in
document_words)
    return features

featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[500:], featuresets[:500]
```

Classification with Naïve Bayes

- Use NLTK Naive Bayes Classifier

```
classifier = nltk.NaiveBayesClassifier.train(train_set)

print(nltk.classify.accuracy(classifier, test_set))
```

Classification with Naïve Bayes

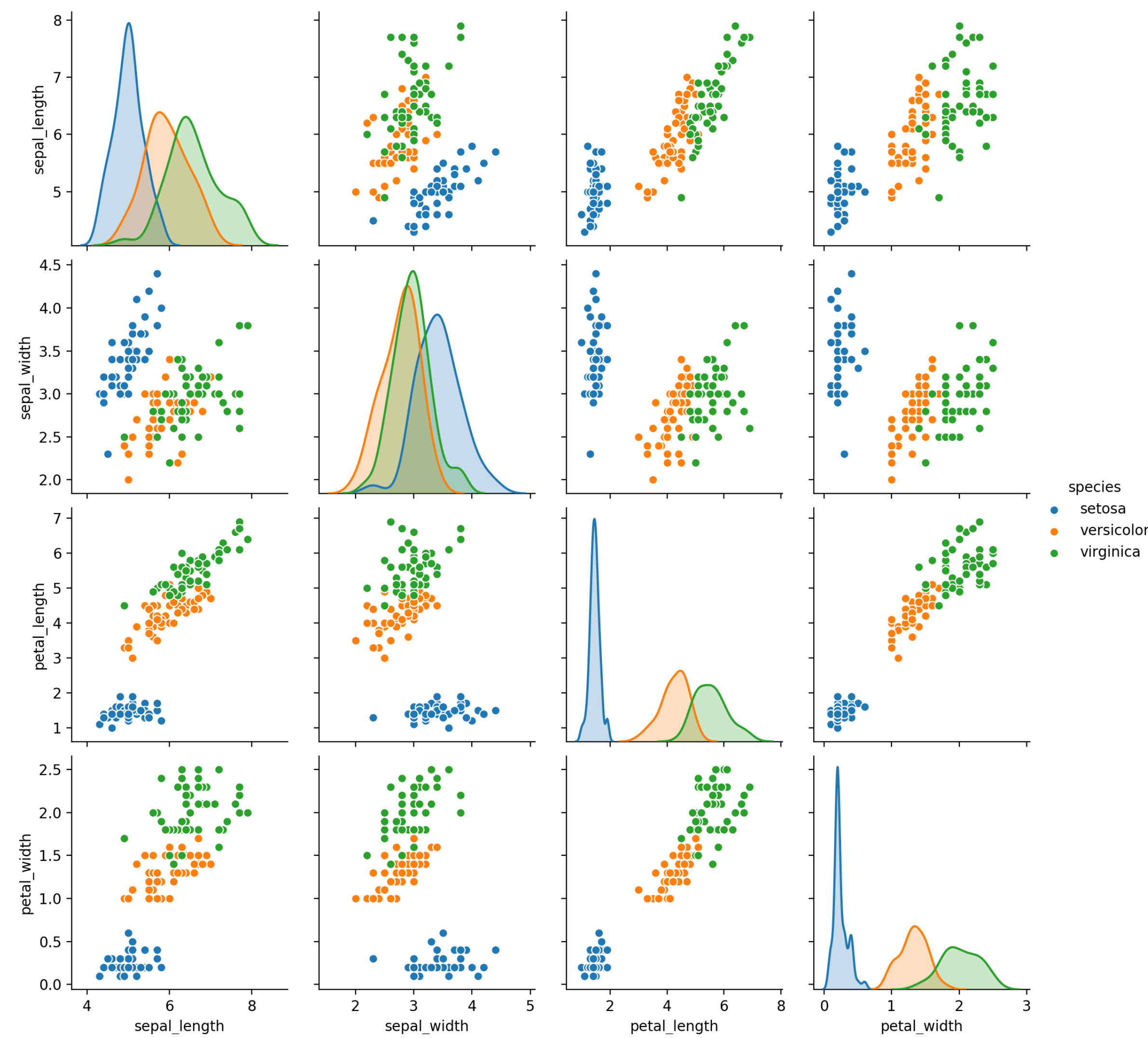
- Results: 80.2% sentiments classified correctly
- Can see how the classifier works

```
>>> classifier.show_most_informative_features(5)
Most Informative Features
        contains(segal) = True              neg : pos    =     11.3 : 1.0
   contains(outstanding) = True              pos : neg    =      8.6 : 1.0
        contains(wasted) = True              neg : pos    =      7.3 : 1.0
         contains(mulan) = True              pos : neg    =      7.2 : 1.0
   contains(wonderfully) = True              pos : neg    =      6.3 : 1.0
```

- And already can see improvements

Classification with Gaussian Bayes

- Continuous features
  - Assumption: Features are distributed normally
  - Example: Look again at Iris set
    - All features are look normally distributed



Classification with Gaussian Naïve Bayes

- Possibility one: Disregard correlation —> Naïve
  - For each feature:
    - Calculate sample mean μ and sample standard deviation σ
    - Use these as estimators of the population mean and deviation
  - For a given feature value *x*, calculate the probability density assuming that x is in a category c
    - $P(x \mid c) \sim \mathcal{N}(\mu_c, \sigma_c)$

Classification with Gaussian Naïve Bayes

- Estimate the probability for observation $(x_1, x_2, \ldots, x_n)$ as the product of the densities

$$P((x_1, \ldots, x_n) \,|\, c_j) \sim \mathcal{N}(x_1, \sigma_{1,c_j}, \mu_{1,c_j}) \cdot \ldots \cdot \mathcal{N}(x_1, \sigma_{n,c_j}, \mu_{1,c_j})$$

- Then use Bayes formula to invert the conditional probabilities
    - This means estimating the prevalence of the categories
    - $$P(c_j \,|\, (x_1, \ldots, x_n)) = \frac{P((x_1, \ldots, x_n) \,|\, c_j) P(c_j)}{P((x_1, \ldots, x_n))}$$

Classification with Gaussian Naïve Bayes

- The denominator does not depend on the category c_j
- So, we just leave it out:
  - $P(c_j \mid (x_1, \ldots, x_n)) \sim P((x_1, \ldots, x_n) \mid c_j) P(c_j)$
- We calculate $P((x_1, \ldots, x_n) \mid c_j) P(c_j)$
  - And select the highest value

Classification with Gaussian Naïve Bayes

- Implemented in sklearn.naive_bayes
  - Example with Iris data-set

```
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB

iris = datasets.load_iris()
model = GaussianNB()
model.fit(iris.data, iris.target)
print('means', model.theta_)
print('stds', model.sigma_)

for x,t, p in zip(iris.data, iris.target, model.predict(iris.data)):
    print(x, t, p)
```

Classification with Gaussian Naïve Bayes

```
means [[5.006 3.428 1.462 0.246]
 [5.936 2.77  4.26  1.326]
 [6.588 2.974 5.552 2.026]]
stds [[0.121764 0.140816 0.029556 0.010884]
 [0.261104 0.0965   0.2164   0.038324]
 [0.396256 0.101924 0.298496 0.073924]]
[5.1 3.5 1.4 0.2] 0
[4.9 3.  1.4 0.2] 0
[4.7 3.2 1.3 0.2] 0
[4.6 3.1 1.5 0.2] 0
[5.  3.6 1.4 0.2] 0
[5.4 3.9 1.7 0.4] 0
```

Classification with Gaussian Naïve Bayes

- There are a few errors:

```
[6.9 3.1 4.9 1.5] 1 2
[5.9 3.2 4.8 1.8] 1 2
[6.7 3.  5.  1.7] 1 2
[4.9 2.5 4.5 1.7] 2 1
[6.  2.2 5.  1.5] 2 1
[6.3 2.8 5.1 1.5] 2 1
```

- Caution: We did not divide the data set into a training and verification set.

Classification with Not-So-Naïve Gaussian Bayes

- We did not use correlation between features
  - If we do, use the multi-variate probability density
  - Need to estimate correlation coefficients:

$$\sigma_{k,l} = \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} (x_k - \mu_k)(x_l - \mu_l)$$

  - Then use the multi-variate normal probability density

$$\text{norm}_{\mu,\Sigma}(x) = \frac{1}{(\sqrt{2\pi})^d \sqrt{|\Sigma|}} \exp\left(-\frac{(x-\mu)^{\mathsf{T}} \Sigma^{-1} (x-\mu)}{2} \right)$$

Classification with Not-So-Naïve Gaussian Bayes

- Luckily, implemented in scipy.stats

```
from scipy.stats import multivariate_normal
```

- Estimate means and correlations
- Similarly to before, estimate category by looking at the multi-variate normal density for each category and updating

```
def diagnose(tupla):
    return np.argmax(
   [multivariate_normal.pdf(tupla,mean=Gl.mu_setosa, cov=Gl.sigma_setosa),
    multivariate_normal.pdf(tupla,mean=Gl.mu_ver, cov=Gl.sigma_ver),
    multivariate_normal.pdf(tupla,mean=Gl.mu_vgc, cov=Gl.sigma_vgc)])
```

Classification with Not-So-Naïve Gaussian Bayes

- This works slightly better: three mis-classifications
  - Example:
    - Virginica features:

```
>>> get_probs((6.3, 2.8, 5.1, 1.5))
setosa 6.551299963143457e-116
versicolor 0.3895029363227387
virginica 0.2572025404570884 6
```

      - Versicolor and virginica probs are similar

Classification with Not-So-Naïve Gaussian Bayes

- This works slightly better: three mis-classifications
  - Example:
    - Versicolor features:

```
>>> get_probs((6.0, 2.7, 5.1, 1.6))
setosa 3.460160789261244 5e-119
versicolor 0.09776449471242309
virginica 0.5655860779779 2
```

    - Versicolor and virginica probs are somewhat similar

Scipy.learn

- A more modern set of tools in scipy
    - Running example:
        - How to predict the newsgroup from the contents
        - Data set:
        - `from sklearn.datasets import fetch_20newsgroups`

Scipy.learn

- A set of 18846 newsgroup contributions from way back
    - Split 2/3 : 1/3 into a training set (before a certain date) and a test set (after a certain date)

```
data = fetch_20newsgroups()
print(data.target_names)
```

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles',
'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt',
'sci.electronics', 'sci.med', 'sci.space',
'soc.religion.christian', 'talk.politics.guns',
'talk.politics.mideast', 'talk.politics.misc',
'talk.religion.misc']
```

Scipy.learn

- We do not want all of them:

```
categories = [‘talk.religion.misc',
              'soc.religion.christian','alt.atheism',
              'sci.space', 'comp.graphics']
```

- Split into training and test sets

```
train = fetch_20newsgroups(subset='train', categories = categories)
test = fetch_20newsgroups(subset='test', categories = categories)
```

Scipy.learn

- Bag Of Words uses CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
```

- We extract the Bag of Words
- To display, we make the result into a Pandas Dataframe

```
vec = CountVectorizer()
X = vec.fit_transform(train.data)
df = pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
```

Scipy.learn

- The result is a matrix
  - Columns by words that appear
  - Rows by document number

```
>>> df.iloc[0:15, 10300:10330]
    comm  command  commanded  ...  commercialization  commercialized  commercially
0      0        0          0  ...                  0               0             0
1      0        0          0  ...                  0               0             0
2      0        0          0  ...                  0               0             0
3      0        0          0  ...                  0               0             0
4      0        0          0  ...                  0               0             0
5      0        0          0  ...                  0               0             0
6      0        0          0  ...                  0               0             0
7      0        0          0  ...                  0               0             0
8      0        0          0  ...                  0               0             0
9      0        0          0  ...                  0               0             0
10     0        0          0  ...                  0               0             0
11     0        0          0  ...                  0               0             0
12     0        0          0  ...                  0               0             0
13     0        0          0  ...                  0               0             0
14     0        0          0  ...                  0               0             0

[15 rows x 30 columns]
```

Scipy.learn

- Get better result by dividing the words by their frequency

```
vec = TfidfVectorizer()
X = vec.fit_transform(train.data)
df = pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
```

Scipy.learn

- Term Frequency
  - Take raw count and divide by the number of words in the document
- Inverse Document Frequency
  - — Logarithm of (Number of Documents w. word) / (Number of Documents)
- Term-Frequency — Inverse Document Frequency (TfIDF)
  - Product of these two

Scipy.learn

- Let’s make the difference clearer

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import pandas as pd


sample = ['in the beginning of time', 'at dawn we slept',
 'this is the story’, 'beginning and end', 'frequent beginning’,
 'beginning python']

vec = CountVectorizer()
X = vec.fit_transform(sample)
df = pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
print(df)

vec = TfidfVectorizer()
X1 = vec.fit_transform(sample)
df1 = pd.DataFrame(X1.toarray(), columns=vec.get_feature_names())
print(df1)
```

Scipy.learn

- CountVectorizer

```
   and  at  beginning  dawn  end  frequent  ...  slept  story  the  this  time  we
0    0   0          1     0    0         0  ...      0      0    1     0     1   0
1    0   1          0     1    0         0  ...      1      0    0     0     0   1
2    0   0          0     0    0         0  ...      0      1    1     1     0   0
3    1   0          1     0    1         0  ...      0      0    0     0     0   0
4    0   0          1     0    0         1  ...      0      0    0     0     0   0
5    0   0          1     0    0         0  ...      0      0    0     0     0   0

[6 rows x 16 columns]
```

Scipy.learn

- TfIdfVectorizer

```
        and   at  beginning  dawn  ...       the      this      time   we
0  0.000000  0.0   0.295730   0.0  ...  0.408763  0.000000  0.498483  0.0
1  0.000000  0.5   0.000000   0.5  ...  0.000000  0.000000  0.000000  0.5
2  0.000000  0.0   0.000000   0.0  ...  0.427903  0.521823  0.000000  0.0
3  0.652057  0.0   0.386839   0.0  ...  0.000000  0.000000  0.000000  0.0
4  0.000000  0.0   0.510227   0.0  ...  0.000000  0.000000  0.000000  0.0
5  0.000000  0.0   0.510227   0.0  ...  0.000000  0.000000  0.000000  0.0

[6 rows x 16 columns]
```

Scipy.learn

- CountVectorizer and TfidfVectorizer generate **sparse** matrices
  - Storage is compressed

Scipy.learn

- Multinomial Bayes is in sklearn
  - `from sklearn.naive_bayes import MultinomialNB`
- sklearn has a pipeline constructor
  - Combines feature extraction with training multinomial NB

```
from sklearn.pipeline import make_pipeline

model = make_pipeline(TfidfVectorizer(), MultinomialNB())
model.fit(train.data, train.target)
labels = model.predict(test.data)
```

Scipy.learn

- To measure success:
  - Use a confusion matrix
    - For the test set: Show how often group elements are predicted to belong to another group
    - Fictitious example: Can a NN distinguish cats and dogs

	actual	
	dog	cat
predicted dog	1023	245
predicted cat	134	1183

Scipy.learn

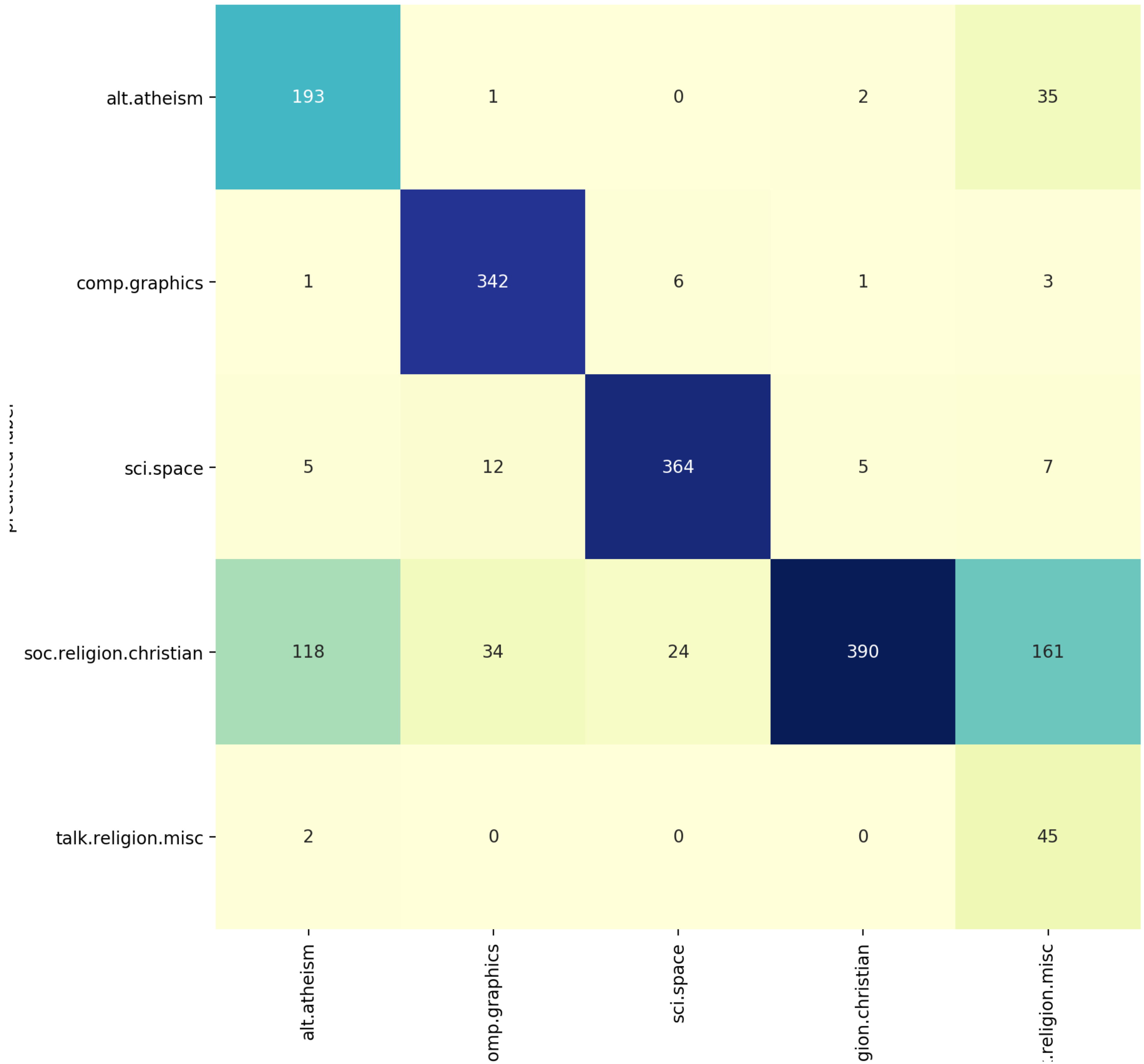
- Can find confusion matrix

```
from sklearn.metrics import confusion_matrix
```

- Import pyplot and seaborn

```
import seaborn as sns
import matplotlib.pyplot as plt

mat = confusion_matrix(test.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=train.target_names,
            yticklabels=train.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.show()
```



alt.atheism
comp.graphics
sci.space
soc.religion.christian
talk.religion.misc
193
1
0
2
35
1
342
6
1
3
5
12
364
5
7
118
34
24
390
161
2
0
0
0
45
alt.atheism
omp.graphics
sci.space
gion.christian
.religion.misc