Multiclass Classification

CS5154/6054

Yizong Cheng

10/27/2022

13.1 The text classification problem

DOCUMENT SPACE
CLASS

In text classification, we are given a description $d \in \mathbb{X}$ of a document, where $\mathbb{X}$ is the *document space*; and a fixed set of *classes* $\mathbb{C} = \{c_1, c_2, \ldots, c_J\}$. Classes are also called *categories* or *labels*. Typically, the document space $\mathbb{X}$ is some type of high-dimensional space, and the classes are human defined for the needs of an application, as in the examples *China* and *documents that talk about multicore computer chips* above. We are given a *training set* $\mathbb{D}$ of labeled documents $\langle d, c \rangle$,where $\langle d, c \rangle \in \mathbb{X} \times \mathbb{C}$. For example:

TRAINING SET

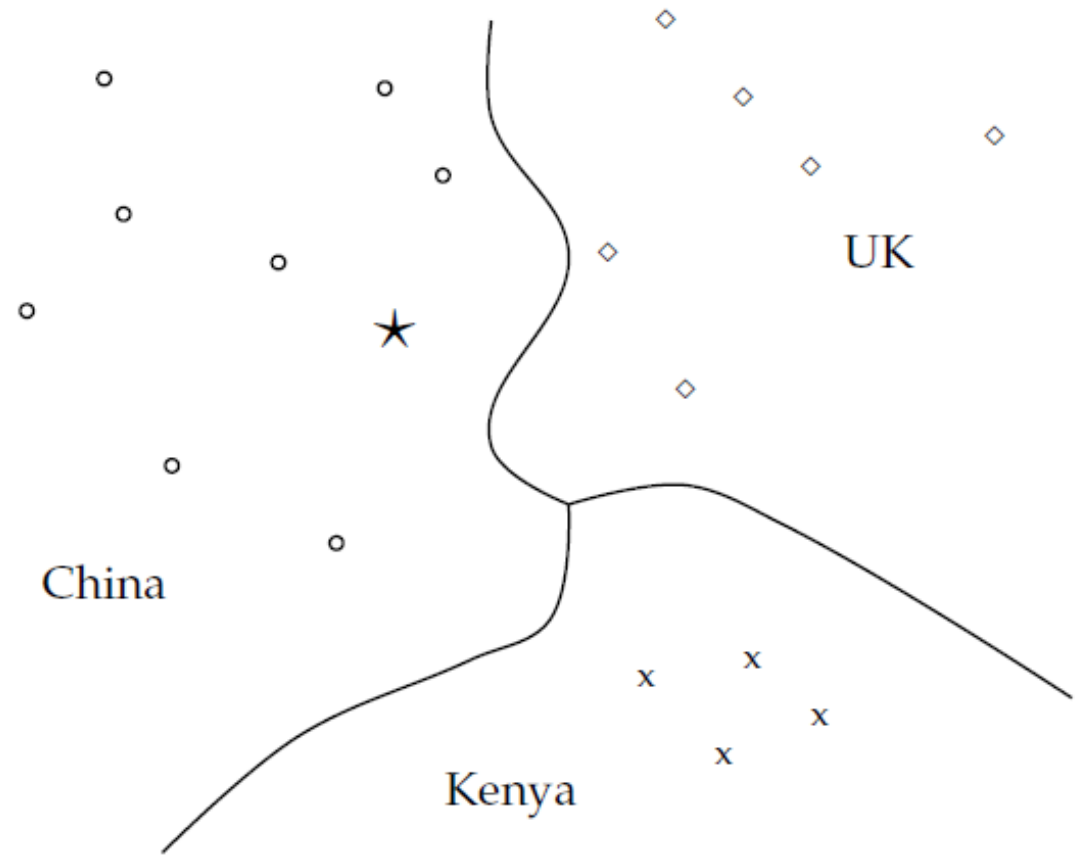
$$\langle d, c \rangle = \langle \text{Beijing joins the World Trade Organization}, \textit{China} \rangle$$

for the one-sentence document *Beijing joins the World Trade Organization* and the class (or label) *China*.

LEARNING METHOD
CLASSIFIER

Using a *learning method* or *learning algorithm*, we then wish to learn a classifier or *classification function* γ that maps documents to classes:

$$\gamma : \mathbb{X} \rightarrow \mathbb{C} \tag{13.1}$$



► **Figure 14.1** Vector space classification into three classes.

14.5 Classification with more than two classes

We can extend two-class linear classifiers to $J > 2$ classes. The method to use depends on whether the classes are mutually exclusive or not.

Any-Of Classification

ANY-OF CLASSIFICATION

Classification for classes that are not mutually exclusive is called *any-of*, *multilabel*, or *multivalue classification*. In this case, a document can belong to several classes simultaneously, or to a single class, or to none of the classes. A decision on one class leaves all options open for the others. It is sometimes said that the classes are *independent* of each other, but this is misleading since the classes are rarely statistically independent in the sense defined on page 275. In terms of the formal definition of the classification problem in Equation (13.1) (page 256), we learn J different classifiers γ_j in any-of classification, each returning either c_j or $\overline{c}_j$: $\gamma_j(d) \in \{c_j, \overline{c}_j\}$.

Solving an any-of classification task with linear classifiers is straightforward:

1. Build a classifier for each class, where the training set consists of the set of documents in the class (positive labels) and its complement (negative labels).

2. Given the test document, apply each classifier separately. The decision of one classifier has no influence on the decisions of the other classifiers.

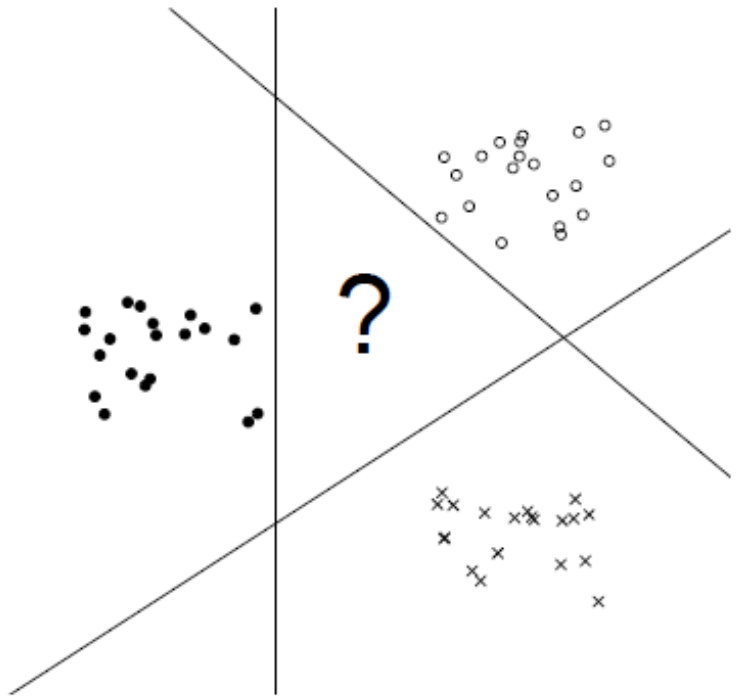
One-Of Classification

ONE-OF CLASSIFICATION

The second type of classification with more than two classes is *one-of classification*. Here, the classes are mutually exclusive. Each document must belong to exactly one of the classes. One-of classification is also called *multinomial*, *polytomous*[4], *multiclass*, or *single-label classification*. Formally, there is a single classification function γ in one-of classification whose range is $\mathbb{C}$, i.e., $\gamma(d) \in \{c_1, \ldots, c_J\}$. kNN is a (nonlinear) one-of classifier.

True one-of problems are less common in text classification than any-of problems. With classes like *UK*, *China*, *poultry*, or *coffee*, a document can be relevant to many topics simultaneously – as when the prime minister of the UK visits China to talk about the coffee and poultry trade.

Nevertheless, we will often make a one-of assumption, as we did in Figure 14.1, even if classes are not really mutually exclusive. For the classification problem of identifying the language of a document, the one-of assumption is a good approximation as most text is written in only one language. In such cases, imposing a one-of constraint can increase the classifier's effectiveness because errors that are due to the fact that the any-of classifiers assigned a document to either no class or more than one class are eliminated.



► **Figure 14.12** J hyperplanes do not divide space into J disjoint regions.

J hyperplanes do not divide $\mathbb{R}^{|V|}$ into J distinct regions as illustrated in Figure 14.12. Thus, we must use a combination method when using two-class linear classifiers for one-of classification. The simplest method is to rank classes and then select the top-ranked class. Geometrically, the ranking can be with respect to the distances from the J linear separators. Documents close to a class's separator are more likely to be misclassified, so the greater the distance from the separator, the more plausible it is that a positive classification decision is correct. Alternatively, we can use a direct measure of confidence to rank classes, e.g., probability of class membership. We can state this algorithm for one-of classification with linear classifiers as follows:

1. Build a classifier for each class, where the training set consists of the set of documents in the class (positive labels) and its complement (negative labels).

2. Given the test document, apply each classifier separately.

3. Assign the document to the class with

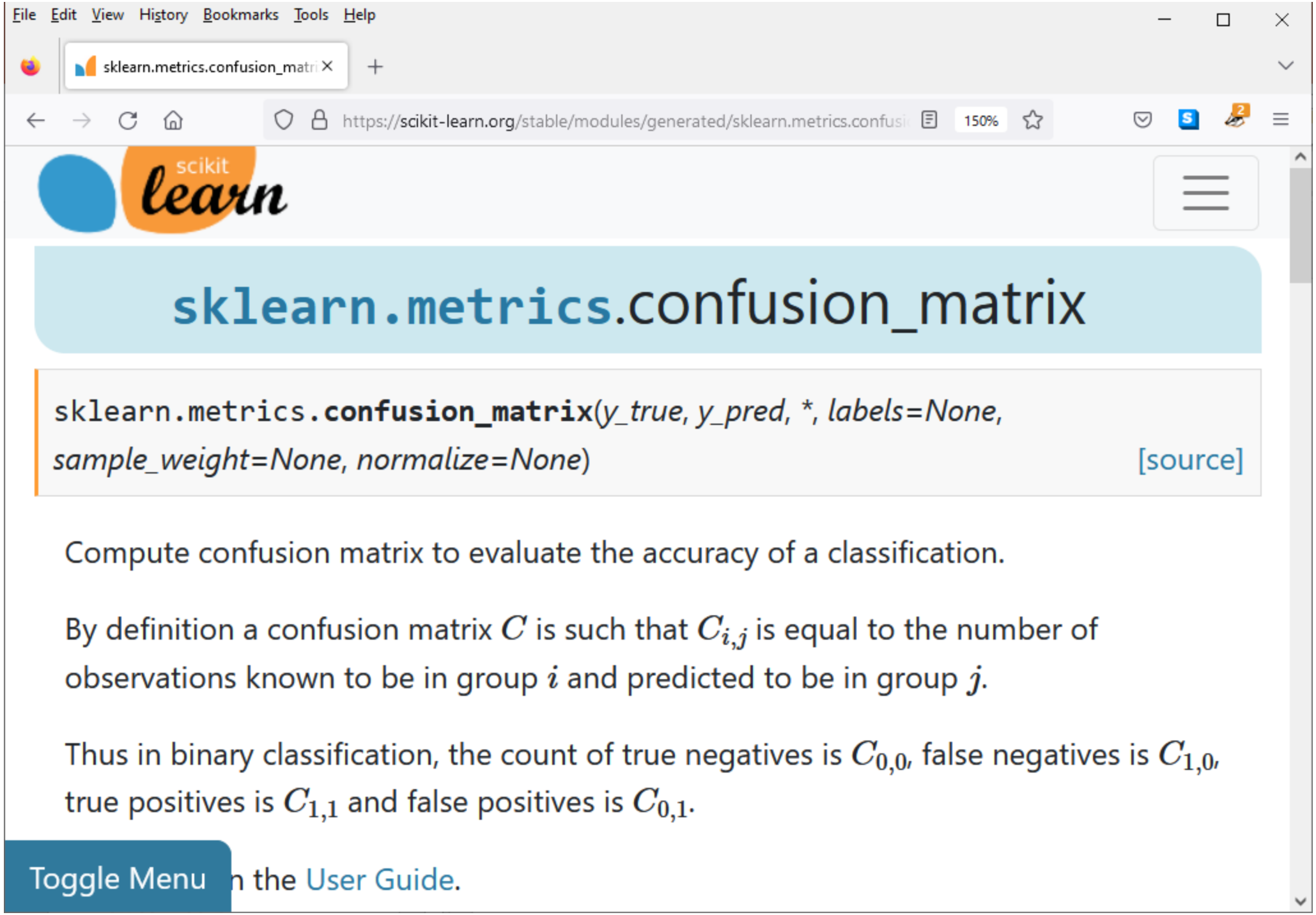
   - the maximum score,
   - the maximum confidence value,
   - or the maximum probability.

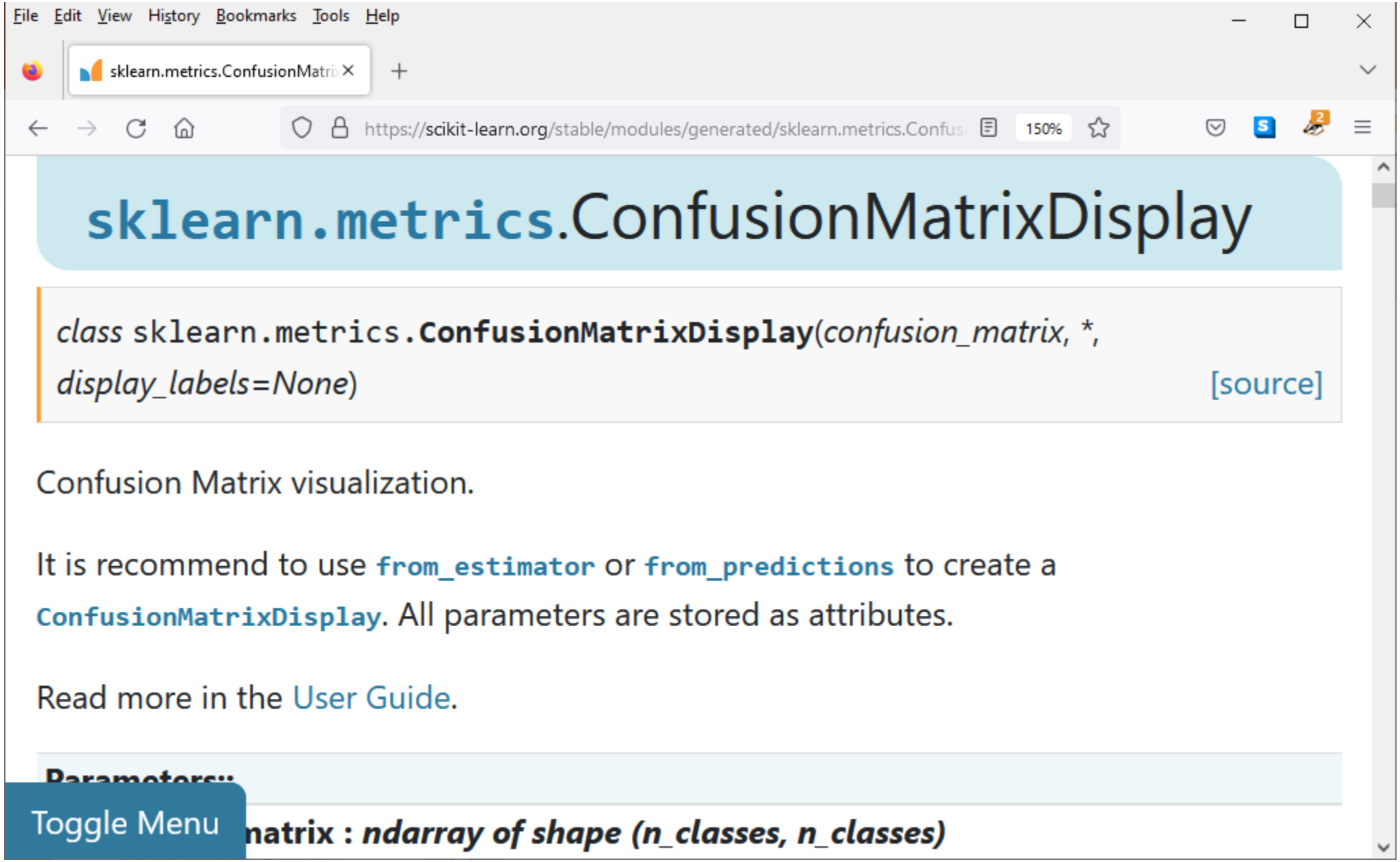
CONFUSION MATRIX

An important tool for analyzing the performance of a classifier for $J > 2$ classes is the *confusion matrix*. The confusion matrix shows for each pair of classes $\langle c_1, c_2 \rangle$, how many documents from c_1 were incorrectly assigned to c_2. In Table 14.5, the classifier manages to distinguish the three financial classes *money-fx*, *trade*, and *interest* from the three agricultural classes *wheat*, *corn*, and *grain*, but makes many errors within these two groups. The confusion matrix can help pinpoint opportunities for improving the accuracy of the system. For example, to address the second largest error in Table 14.5 (14 in the row *grain*), one could attempt to introduce features that distinguish *wheat* documents from *grain* documents.

true class \ assigned class	*money-fx*	*trade*	*interest*	*wheat*	*corn*	*grain*
money-fx	95	0	10	0	0	0
trade	1	1	90	0	1	0
interest	13	0	0	0	0	0
wheat	0	0	1	34	3	7
corn	1	0	2	13	26	5
grain	0	0	2	14	5	10

► **Table 14.5** A confusion matrix for Reuters-21578. For example, 14 documents from *grain* were incorrectly assigned to *wheat*. Adapted from Picca et al. (2006).


File Edit View History Bookmarks Tools Help
sklearn.metrics.confusion_matri
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusi
150%
scikit learn
sklearn.metrics.confusion_matrix
sklearn.metrics.confusion_matrix(y_true, y_pred, *, labels=None, sample_weight=None, normalize=None) [source]
Compute confusion matrix to evaluate the accuracy of a classification.
By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j.
Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$.
Toggle Menu
n the User Guide.

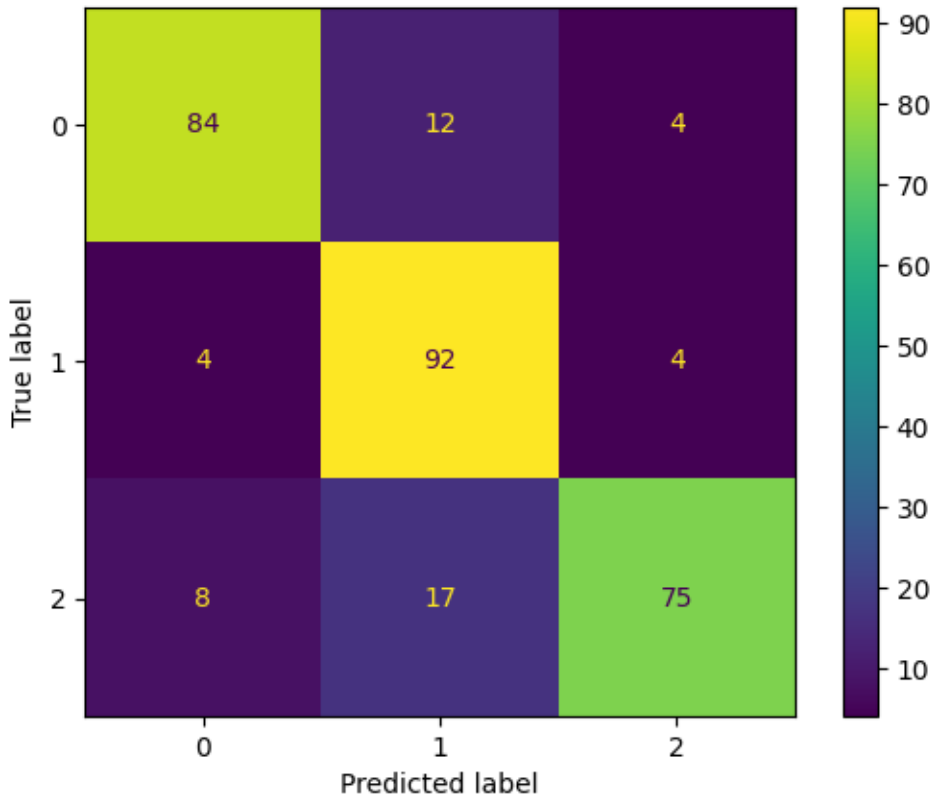

File Edit View History Bookmarks Tools Help
sklearn.metrics.ConfusionMatri
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.Confus
150%
sklearn.metrics.ConfusionMatrixDisplay
class sklearn.metrics.ConfusionMatrixDisplay(confusion_matrix, *, display_labels=None)
[source]
Confusion Matrix visualization.
It is recommend to use from_estimator or from_predictions to create a ConfusionMatrixDisplay. All parameters are stored as attributes.
Read more in the User Guide.
Parameters::
natrix : ndarray of shape (n_classes, n_classes)
Toggle Menu

```
f = open("bible.txt", "r")
docs = f.readlines()
f.close()
N =len(docs)
mid = 1100 + (N - 3 * 1100) // 2
trainX = np.concatenate([docs[0:1000], docs[mid:mid+1000], docs[N-1000:N]])
y = np.concatenate([np.zeros(1000, dtype=np.int16), np.ones(1000, dtype=np.int16), np.full(1000, 2, dtype=np.int16)])
testX = np.concatenate([docs[1000:1100], docs[mid+1000:mid+1100], docs[N-1100:N-1000]])
testY = np.concatenate([np.zeros(100, dtype=np.int16), np.ones(100, dtype=np.int16), np.full(100, 2, dtype=np.int16)])

 cv = CountVectorizer(binary=True, max_df=0.4, min_df=4)
 X0 = cv.fit_transform(trainX).toarray()
 T0 = cv.transform(testX).toarray()

 model = BernoulliNB()
 model.fit(X0, y)
 pred = model.predict(T0)
 print(pred)
 A0 = accuracy_score(testY, pred)
 print ('BernoulliNB -', A0)
```

```
[0 2 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 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 2 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 1 2 0 1 0 0 1
 0 1 0 0 0 0 1 1 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 2 1 1 1 1 1 1 1 2 2 0 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 0 2 2 2 2 2 2 1 2 2 2 1 2
 2 1 2 2 2 2 2 2 2 2 0 2 2 2 2 0 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 0 2 2 2
 2 1 2 0 2 2 0 2 2 1 2 2 0 2 1 2 2 2 2 1 1 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2
 2 1 0 2]
BernoulliNB - 0.8366666666666667
```

```
cm = confusion_matrix(testY, pred)
print(cm)
disp = ConfusionMatrixDisplay(cm, display_labels=model.classes_)
disp.plot()
plt.show()
```

```
[[84 12  4]
 [ 4 92  4]
 [ 8 17 75]]
```



15 *Support vector machines and machine learning on documents*

15.4 Machine learning methods in ad hoc information retrieval

15.4.2 Result ranking by machine learning

REGRESSION

ORDINAL REGRESSION

However, approaching IR result ranking like this is not necessarily the right way to think about the problem. Statisticians normally first divide problems into classification problems (where a categorical variable is predicted) versus *regression* problems (where a real number is predicted). In between is the specialized field of *ordinal regression* where a ranking is predicted. Machine learning for ad hoc retrieval is most properly thought of as an ordinal regression problem, where the goal is to rank a set of documents for a query, given training data of the same sort. This formulation gives some additional power, since documents can be evaluated relative to other candidate documents for the same query, rather than having to be mapped to a global scale of goodness, while also weakening the problem space, since just a ranking is required rather than an absolute measure of relevance. Issues of ranking are especially germane in web search, where the ranking at the very top of the results list is exceedingly important, whereas decisions of relevance of a document to a query may be much less important. Such

By hypothesis, one of d_i and d_j has been judged more relevant. If d_i is judged more relevant than d_j, denoted $d_i \prec d_j$ (d_i should precede d_j in the results ordering), then we will assign the vector $\Phi(d_i, d_j, q)$ the class $y_{ijq} = +1$; otherwise -1. The goal then is to build a classifier which will return

$$\vec{w}^T \Phi(d_i, d_j, q) > 0 \quad \text{iff} \quad d_i \prec d_j \tag{15.19}$$

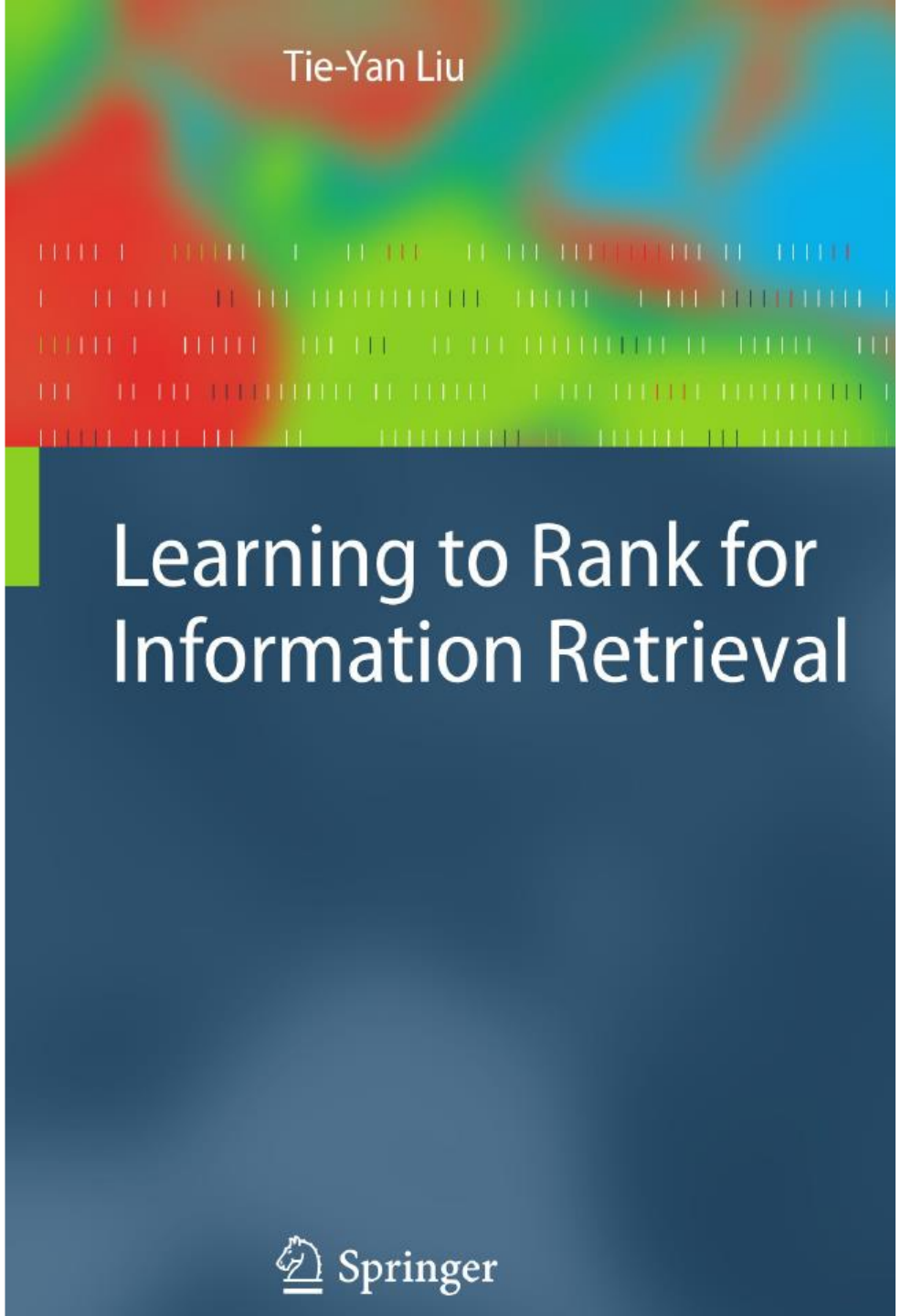
This SVM learning task is formalized in a manner much like the other examples that we saw before:

(15.20) Find $\vec{w}$, and $\xi_{i,j} \geq 0$ such that:

- $\frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i,j}\xi_{i,j}$ is minimized
- and for all $\{\Phi(d_i, d_j, q) : d_i \prec d_j\}$, $\vec{w}^T\Phi(d_i, d_j, q) \geq 1 - \xi_{i,j}$

We can leave out y_{ijq} in the statement of the constraint, since we only need to consider the constraint for document pairs ordered in one direction, since $\prec$ is antisymmetric. These constraints are then solved, as before, to give a linear classifier which can rank pairs of documents. This approach has been used to build ranking functions which outperform standard hand-built ranking functions in IR evaluations on standard data sets; see the references for papers that present such results.

T.-Y. Liu, *Learning to Rank for Information Retrieval*,
DOI 10.1007/978-3-642-14267-3_1, © Springer-Verlag Berlin Heidelberg 2011



Introduction to

Information Retrieval

CS276: Information Retrieval and Web Search
Christopher Manning and Pandu Nayak

Lecture 14: Learning to Rank (with GBDTs)

Borrows slides/pictures from Schigehiko Schamoni

“Learning to rank”

- Classification probably isn’t the right way to think about approaching ad hoc IR:
  - Classification problems: Map to an unordered set of classes
  - Regression problems: Map to a real value
  - Ordinal regression (or “ranking”) problems: Map to an *ordered* set of classes
    - A fairly obscure sub-branch of statistics, but what we want here
- This formulation gives extra power:
  - Relations between relevance levels are modeled
  - **Documents are good versus other documents for a query given collection**; not an absolute scale of goodness