Competing Classifiers

CS5154/6054

Yizong Cheng

10/25/2022

Introduction to

Information Retrieval

CS276: Information Retrieval and Web Search

Christopher Manning and Pandu Nayak

Lecture 13: Decision trees and machine learning on documents

Some slides borrowed from John Canny

The discriminative alternative: Logistic Regression and Support vector machines

- Directly predict class conditional on words:
- (Binary) Logistic Regression:

$$\log \frac{P(C|d)}{P(\overline{C}|d)} = \alpha + \sum_{w \in d} \beta_w \times w$$

- Tune parameters β_w to optimize conditional likelihood or "margin" (SVM) in predicting classes
- What a statistician would probably tell you to use if you said you had a categorical decision problem (like text categorization)

LogR/SVM Performance

- Early results with LogR were disappointing, because people didn't understand the means to *regularize* (smooth) LogR to cope with **sparse textual features**
- Done right, LogR clearly outperforms NB in text categorization and batch filtering studies
- SVMs were seen as the best general text classification method in the period c. 1997– 2005
- LogR seems as good as SVMs (Tong & Oles 2001)
- But now challenged by:
    - Neural net methods (improve word similarity models)
    - Ensemble methods, e.g. **random forests, boosting**

Ensemble Methods

Are like **Crowdsourced machine learning algorithms**:

- Take a collection of simple or *weak* learners
- Combine their results to make a single, better learner

Types:

- **Bagging:** train learners in parallel on different samples of the data, then combine by voting (discrete output) or by averaging (continuous output).
- **Stacking:** feed output of first-level model(s) as features into a second-stage learner like logistic regression.
- **Boosting:** train subsequent learners on the filtered/weighted output of earlier learners so they fix the stuff that the earlier learners got wrong

Random Forests

Grow K trees on datasets sampled from the original dataset with replacement (bootstrap samples), p = number of features.

- Draw K bootstrap samples of size N (size of original dataset)
- Grow each Decision Tree, by selecting a random set of m out of p features at each node, and choosing the best feature to split on.
    - Typically m might be e.g. sqrt(p)
- **Runtime:** Aggregate the predictions of the trees (most popular vote) to produce the final class.

Random Forests

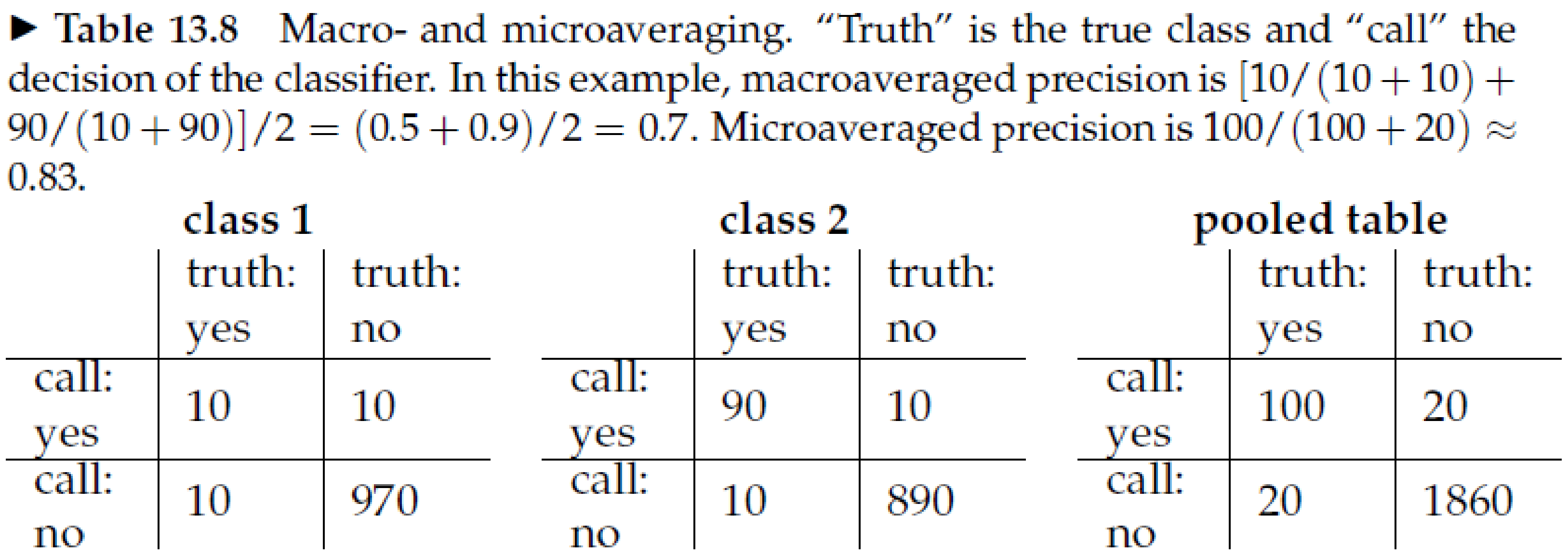
Principles: we want to take a vote between different learners so we don’t want the models to be too similar. These two criteria ensure diversity in the individual trees:

- Data bagging: Draw K bootstrap samples of size N:
  - Each tree is trained on different data.
- Feature bagging: Grow a Decision Tree, by selecting a random set of m out of p features at each node, and choosing the best feature to split on.
  - Corresponding nodes in different trees (usually) can’t use the same feature to split.

Random Forests

- **Very popular in practice,** at one point the most popular classifier for dense data (<= a few thousand features)
- **Easy to implement** (train a lot of trees).
- **Parallelizes easily** (but not necessarily efficiently). Good match for MapReduce.
- **Now not quite state-of-the-art accuracy** – Gradient-boosted trees (less features) and Deep NNs (vision, speech, language, …) generally do better
- **Needs many passes over the data** – at least the max depth of the trees. (<< boosted trees though)
- **Easy to overfit** – need to balance accuracy/fit tradeoff.

Macroaveraging and Microaveraging



▶ **Table 13.8** Macro- and microaveraging. "Truth" is the true class and "call" the decision of the classifier. In this example, macroaveraged precision is $[10/(10+10)+90/(10+90)]/2 = (0.5+0.9)/2 = 0.7$. Microaveraged precision is $100/(100+20) \approx 0.83$.

class 1

	truth: yes	truth: no
call: yes	10	10
call: no	10	970

class 2

	truth: yes	truth: no
call: yes	90	10
call: no	10	890

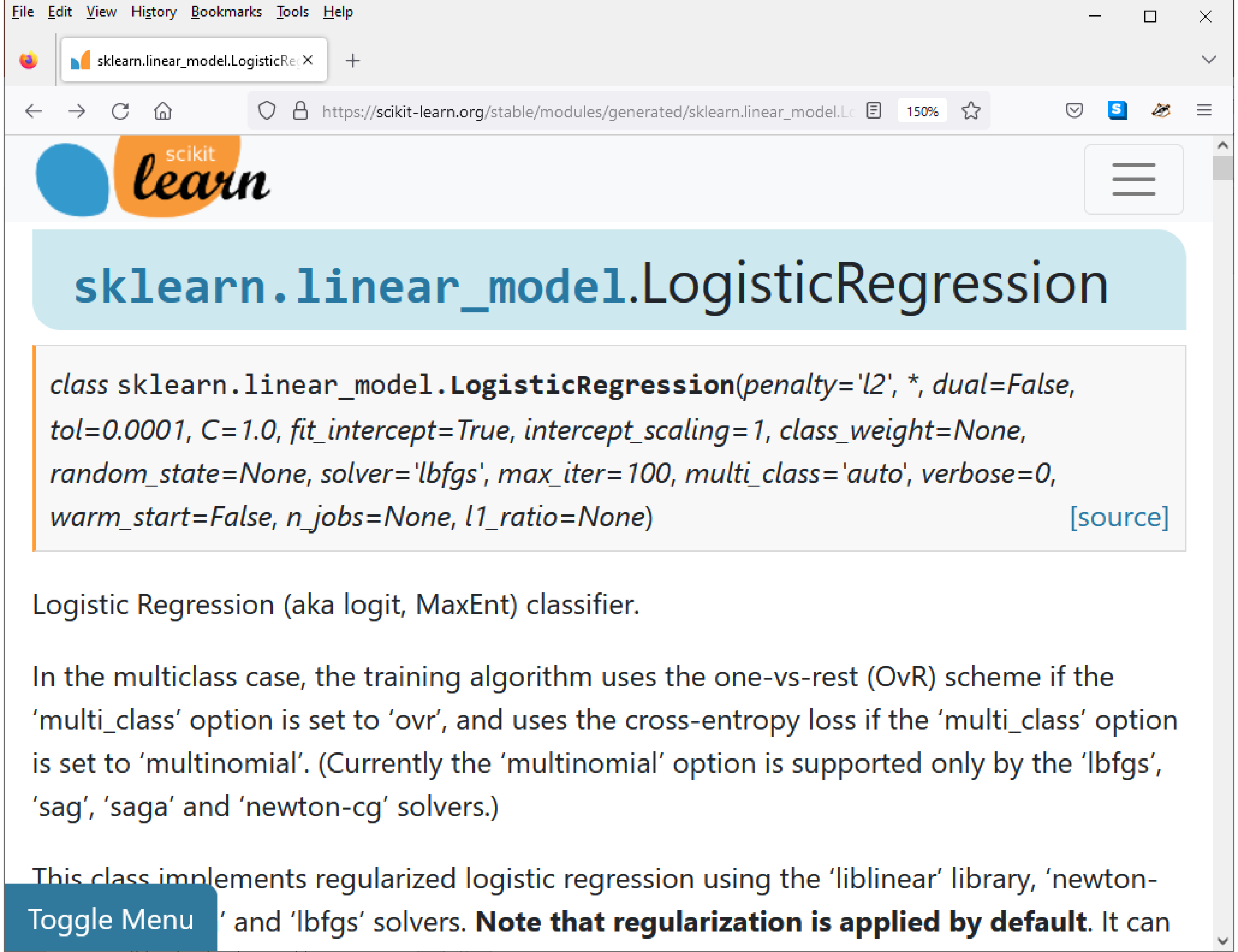
pooled table

	truth: yes	truth: no
call: yes	100	20
call: no	20	1860

▶ **Table 13.9** Text classification effectiveness numbers on Reuters-21578 for F_1 (in percent). Results from Li and Yang (2003) (a), Joachims (1998) (b: kNN) and Dumais et al. (1998) (b: NB, Rocchio, trees, SVM).

(a)	NB	Rocchio	kNN		SVM
micro-avg-L (90 classes)	80	85	86		89
macro-avg (90 classes)	47	59	60		60

(b)	NB	Rocchio	kNN	trees	SVM
earn	96	93	97	98	98
acq	88	65	92	90	94
money-fx	57	47	78	66	75
grain	79	68	82	85	95
crude	80	70	86	85	89
trade	64	65	77	73	76
interest	65	63	74	67	78
ship	85	49	79	74	86
wheat	70	69	77	93	92
corn	65	48	78	92	90
micro-avg (top 10)	82	65	82	88	92
micro-avg-D (118 classes)	75	62	n/a	n/a	87


File Edit View History Bookmarks Tools Help
sklearn.linear_model.LogisticRe
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.L
150%
scikit learn
Toggle Menu

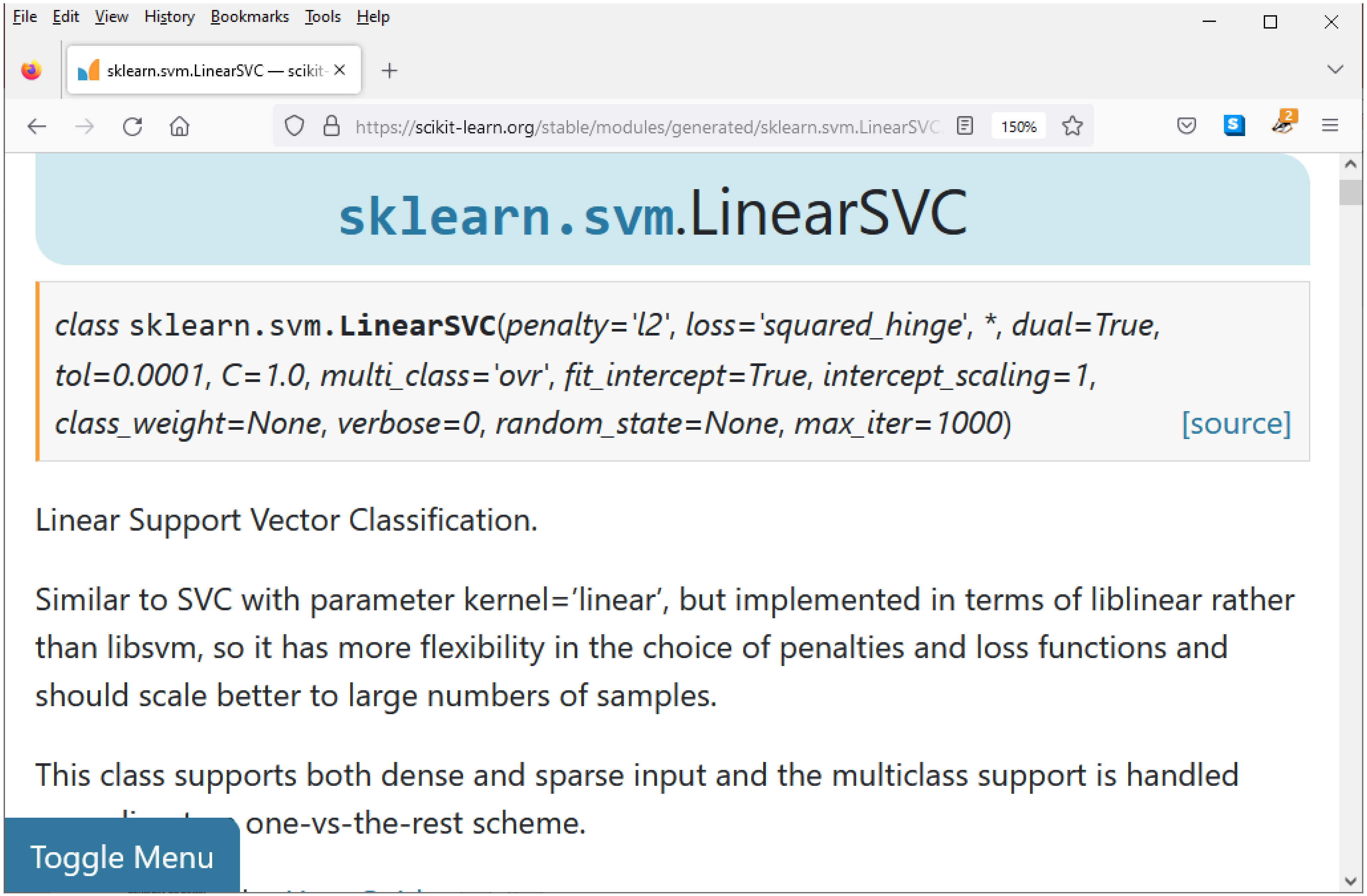
sklearn.linear_model.LogisticRegression

class sklearn.linear_model.**LogisticRegression**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-
' and 'lbfgs' solvers. **Note that regularization is applied by default**. It can


File Edit View History Bookmarks Tools Help
sklearn.svm.LinearSVC — scikit-
https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC
150%

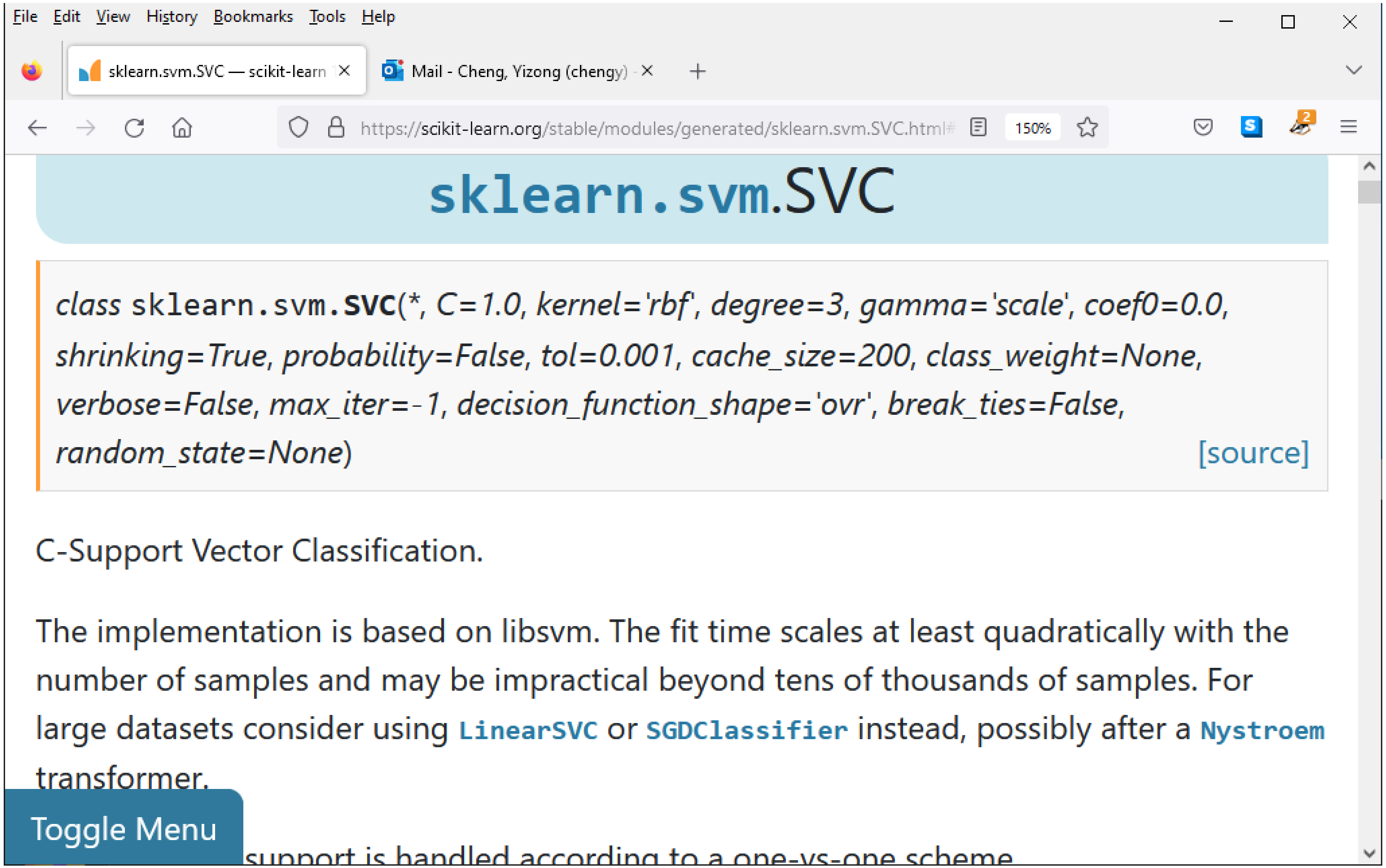
sklearn.svm.LinearSVC

class sklearn.svm.**LinearSVC**(*penalty='l2', loss='squared_hinge', *, dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000*) [source]

Linear Support Vector Classification.

Similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

This class supports both dense and sparse input and the multiclass support is handled one-vs-the-rest scheme.


File Edit View History Bookmarks Tools Help
sklearn.svm.SVC — scikit-learn
Mail - Cheng, Yizong (chengy)
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#
150%
Toggle Menu

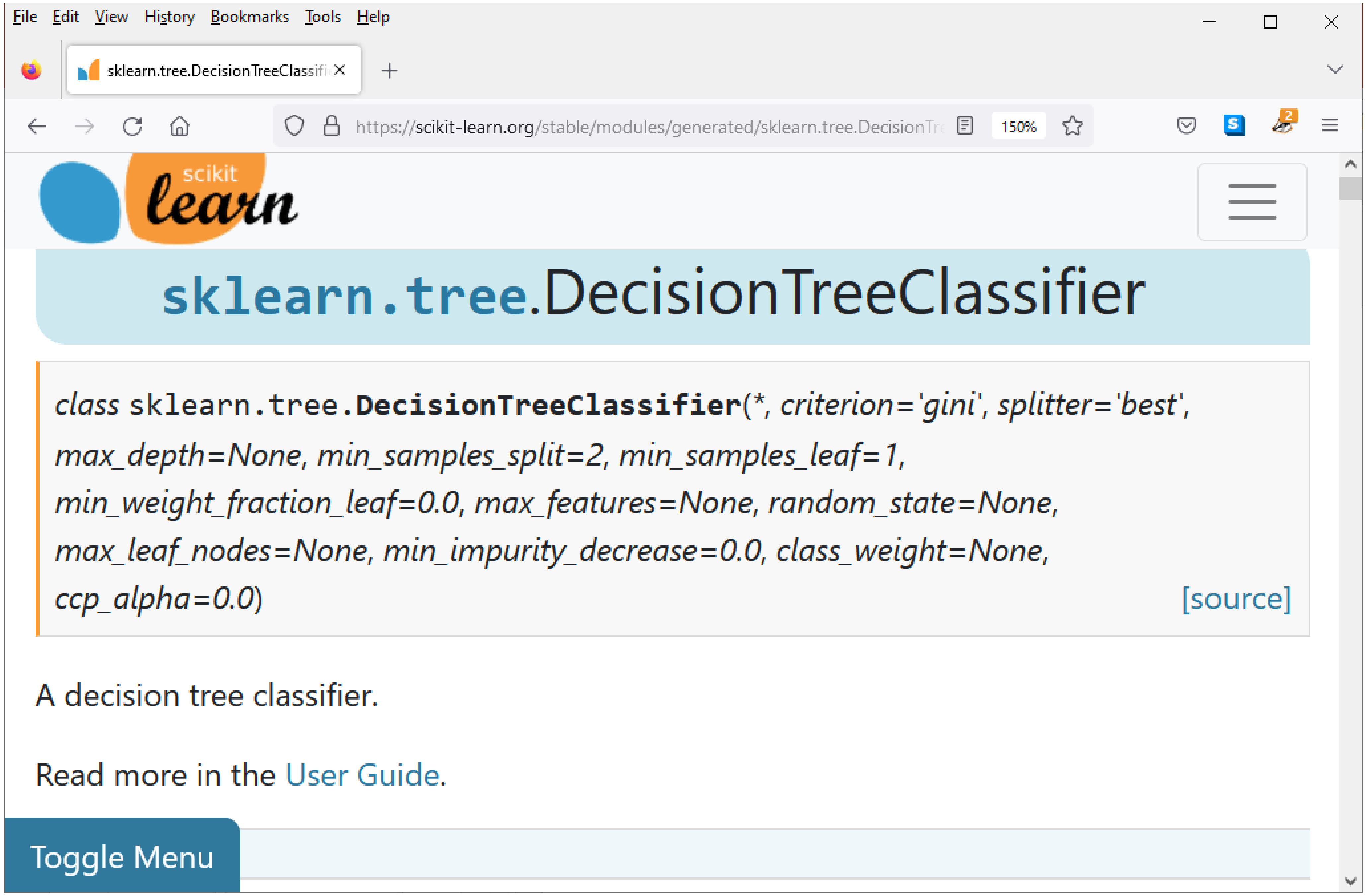
sklearn.svm.SVC

class sklearn.svm.**SVC**(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `LinearSVC` or `SGDClassifier` instead, possibly after a `Nystroem` transformer.

support is handled according to a one-vs-one scheme

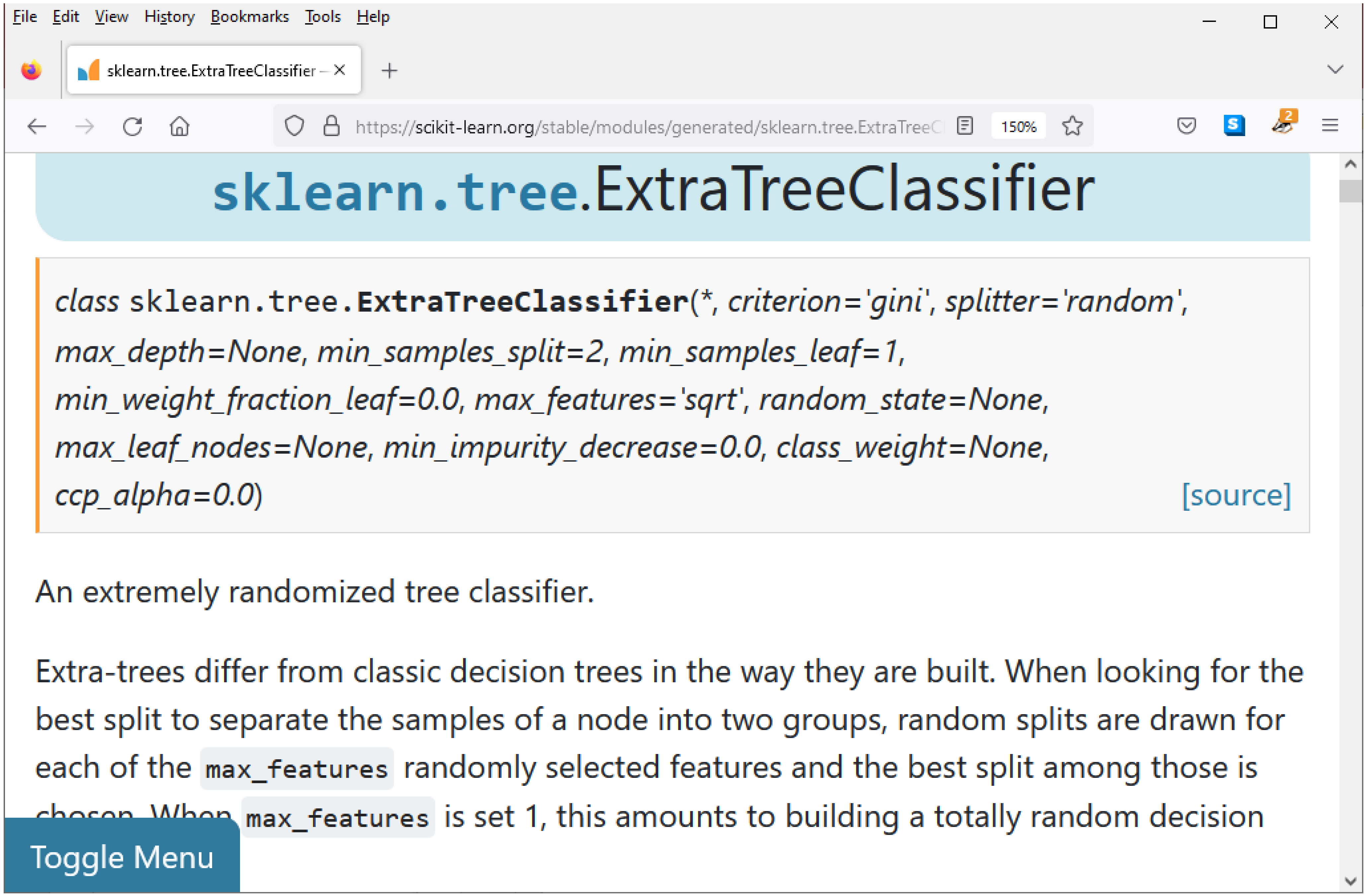

File Edit View History Bookmarks Tools Help
sklearn.tree.DecisionTreeClassifi
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTr
150%
scikit learn
Toggle Menu

sklearn.tree.DecisionTreeClassifier

class sklearn.tree.**DecisionTreeClassifier**(**, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

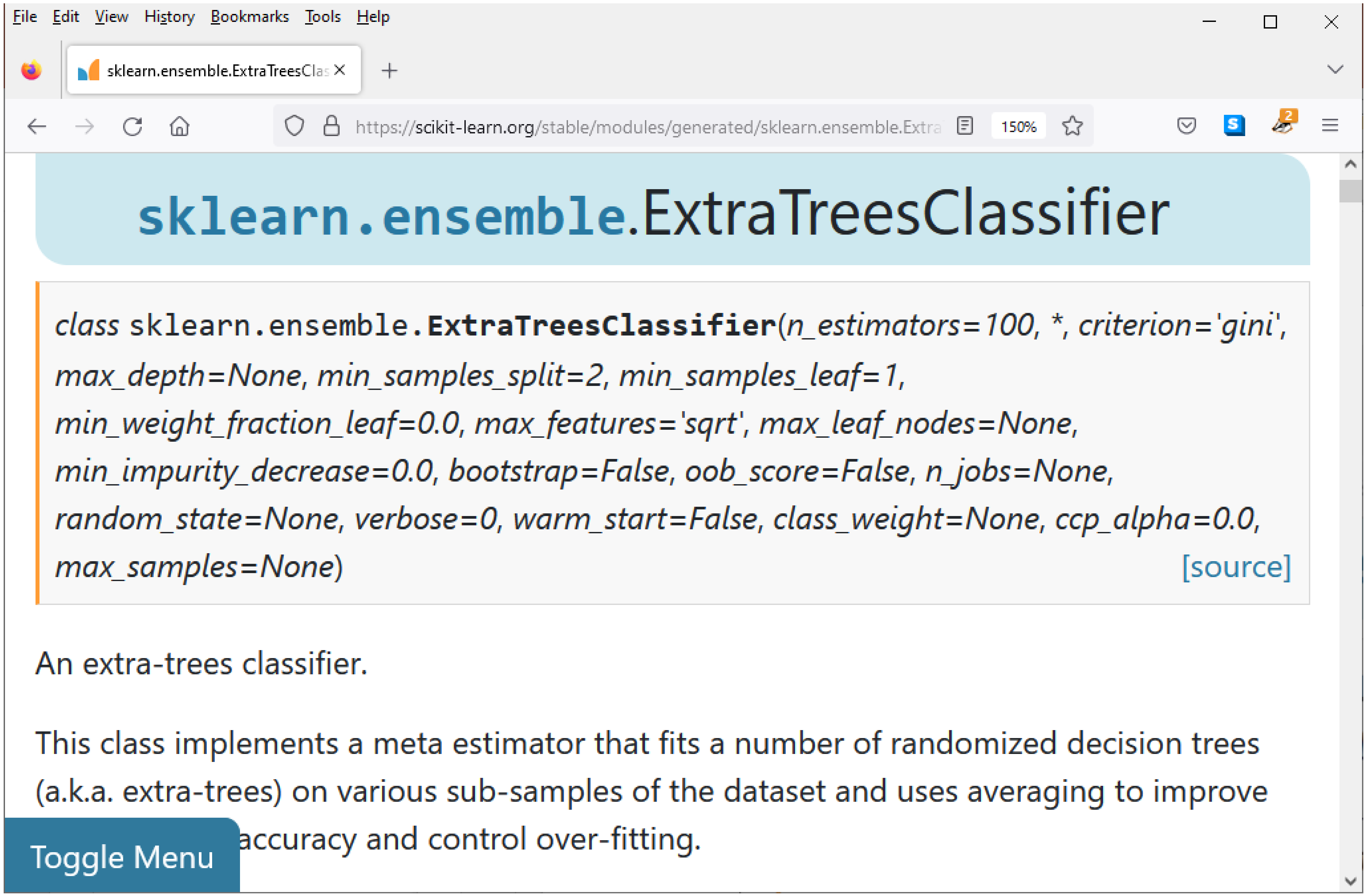

File Edit View History Bookmarks Tools Help
sklearn.tree.ExtraTreeClassifier –
https://scikit-learn.org/stable/modules/generated/sklearn.tree.ExtraTreeC
150%
Toggle Menu

sklearn.tree.ExtraTreeClassifier

class `sklearn.tree.`**`ExtraTreeClassifier`**(**, criterion='gini', splitter='random', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0*) [source]

An extremely randomized tree classifier.

Extra-trees differ from classic decision trees in the way they are built. When looking for the best split to separate the samples of a node into two groups, random splits are drawn for each of the `max_features` randomly selected features and the best split among those is chosen. When `max_features` is set 1, this amounts to building a totally random decision

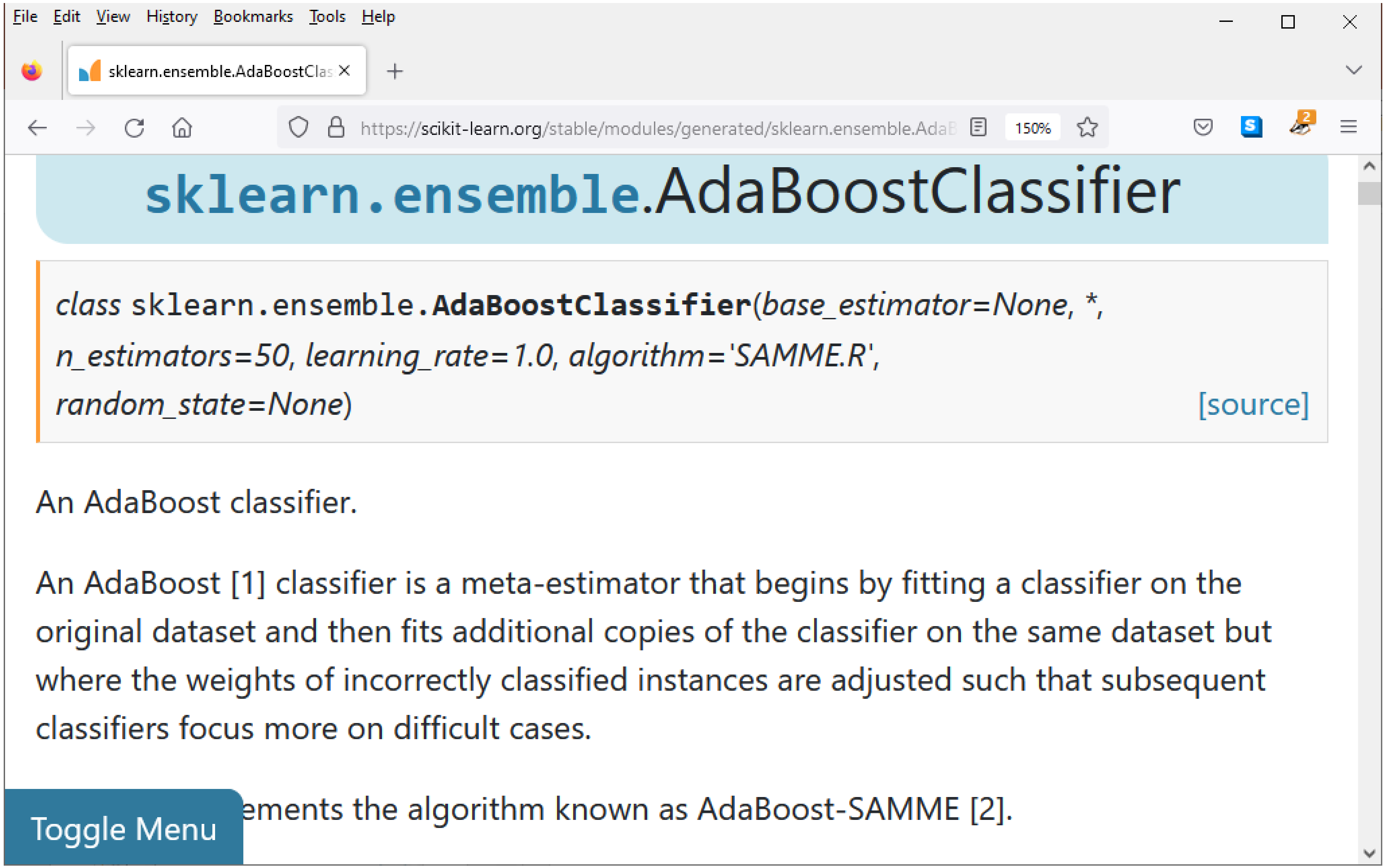

File Edit View History Bookmarks Tools Help
sklearn.ensemble.ExtraTreesClas
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Extra
150%
Toggle Menu

sklearn.ensemble.ExtraTreesClassifier

class `sklearn.ensemble.`**`ExtraTreesClassifier`**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=False, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

An extra-trees classifier.

This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve accuracy and control over-fitting.


File Edit View History Bookmarks Tools Help
sklearn.ensemble.AdaBoostClas
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaB
150%
Toggle Menu

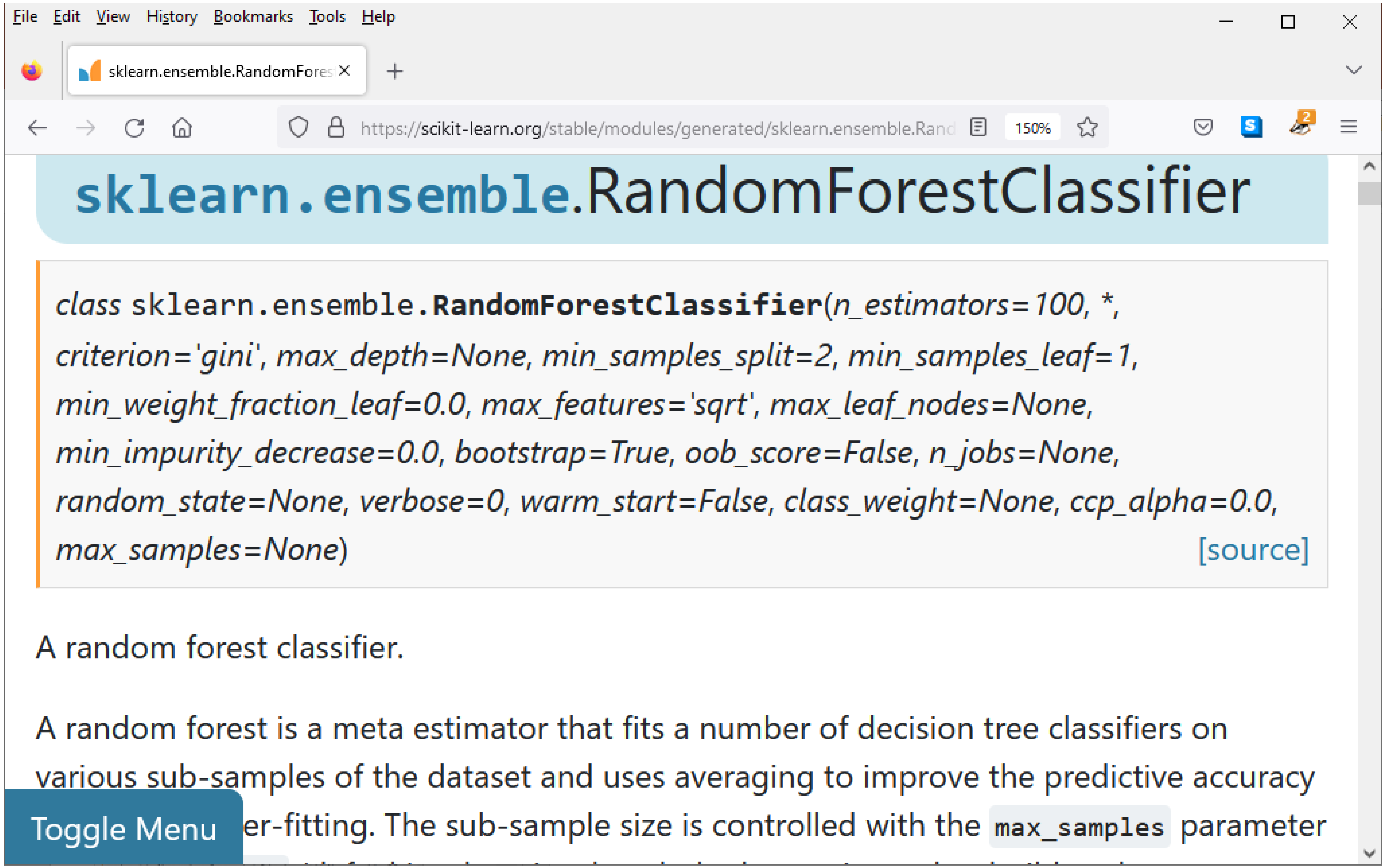
sklearn.ensemble.AdaBoostClassifier

class sklearn.ensemble.**AdaBoostClassifier**(*base_estimator=None*, *, *n_estimators=50*, *learning_rate=1.0*, *algorithm='SAMME.R'*, *random_state=None*) [source]

An AdaBoost classifier.

An AdaBoost [1] classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

ements the algorithm known as AdaBoost-SAMME [2].

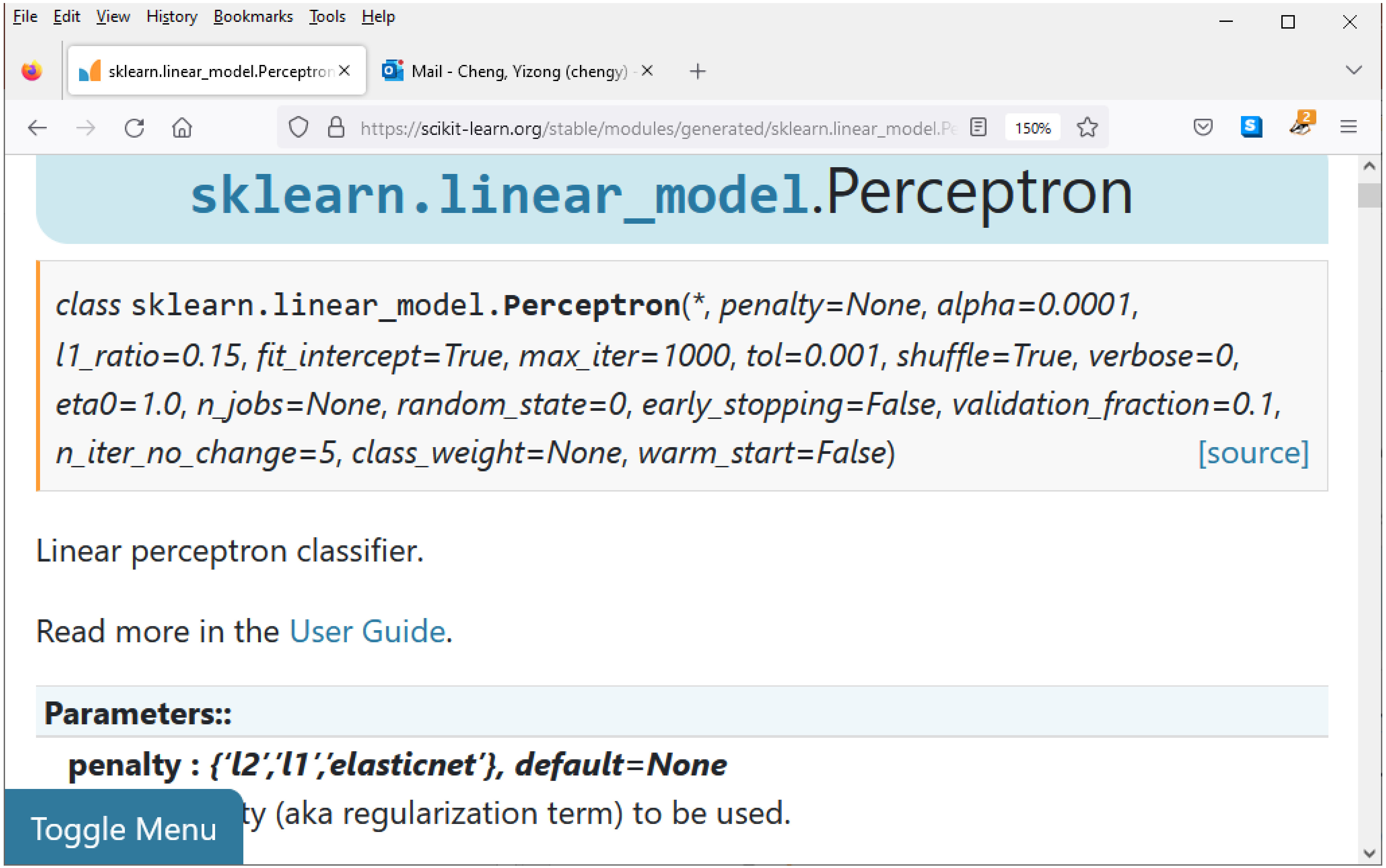

File Edit View History Bookmarks Tools Help
sklearn.ensemble.RandomFores
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Rand
150%
Toggle Menu

sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.**RandomForestClassifier**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy er-fitting. The sub-sample size is controlled with the `max_samples` parameter


File Edit View History Bookmarks Tools Help
sklearn.linear_model.Perceptron
Mail - Cheng, Yizong (chengy)
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Pe
150%
Toggle Menu

sklearn.linear_model.Perceptron

class `sklearn.linear_model.`**`Perceptron`**(**, penalty=None, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, eta0=1.0, n_jobs=None, random_state=0, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False*) [source]

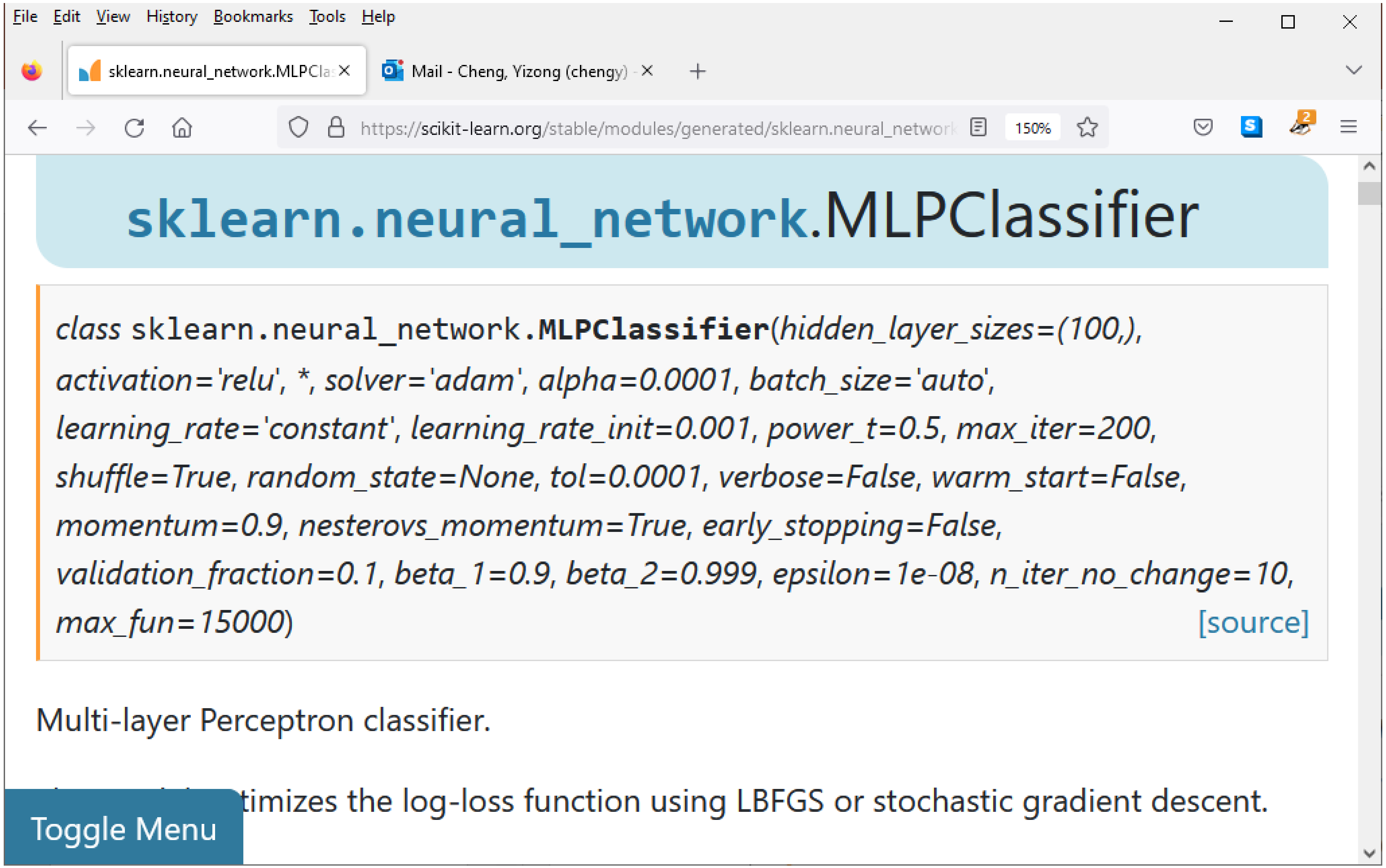
Linear perceptron classifier.

Read more in the User Guide.

Parameters::

penalty : *{'l2','l1','elasticnet'}, default=None*

ty (aka regularization term) to be used.

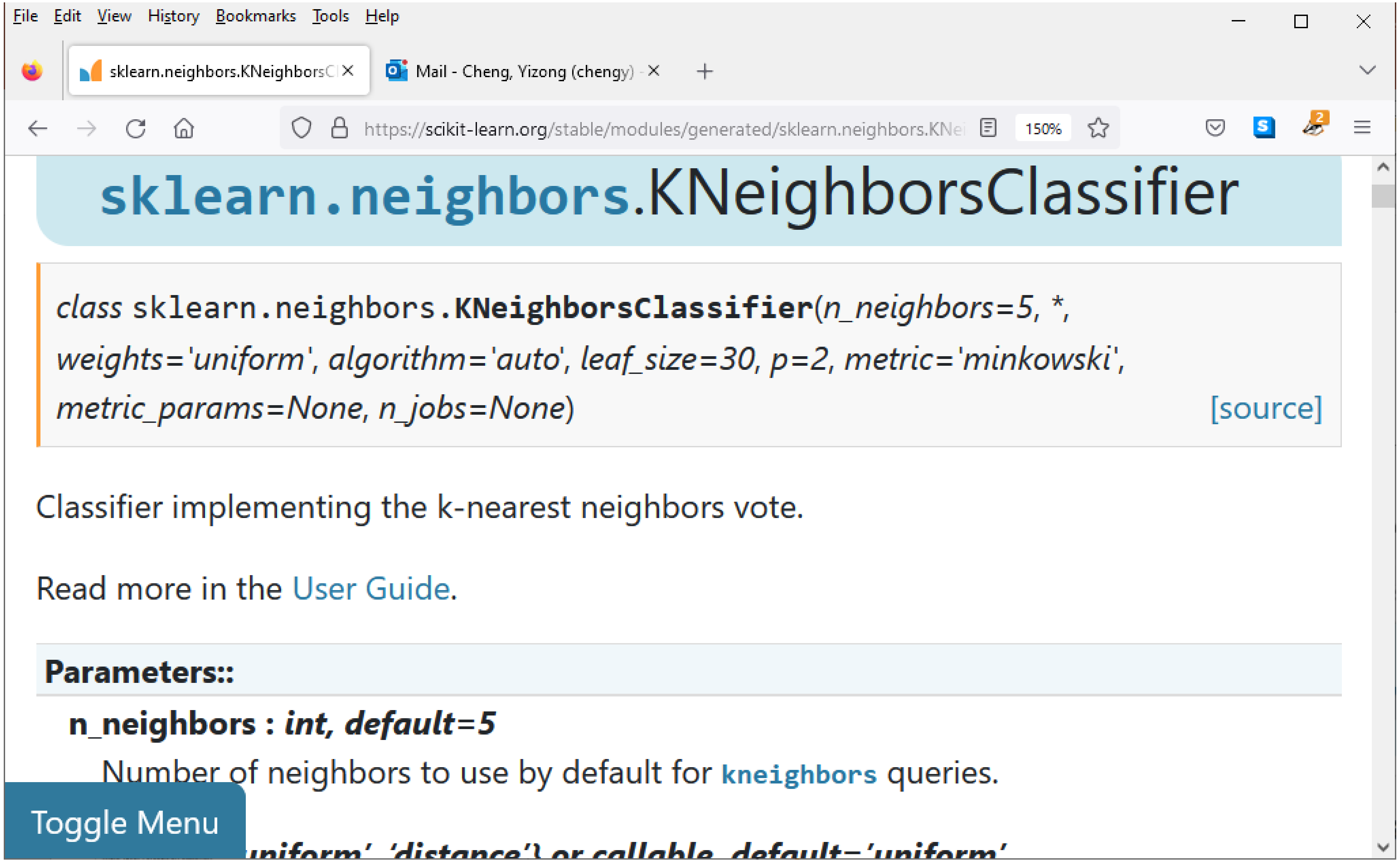

File Edit View History Bookmarks Tools Help
sklearn.neural_network.MLPCla
Mail - Cheng, Yizong (chengy)
https://scikit-learn.org/stable/modules/generated/sklearn.neural_network
150%
Toggle Menu

sklearn.neural_network.MLPClassifier

class `sklearn.neural_network.`**`MLPClassifier`**(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) [source]

Multi-layer Perceptron classifier.

timizes the log-loss function using LBFGS or stochastic gradient descent.


File Edit View History Bookmarks Tools Help
sklearn.neighbors.KNeighborsCl
Mail - Cheng, Yizong (chengy) -
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNe
150%
Toggle Menu

sklearn.neighbors.KNeighborsClassifier

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5*, ***, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*) [source]

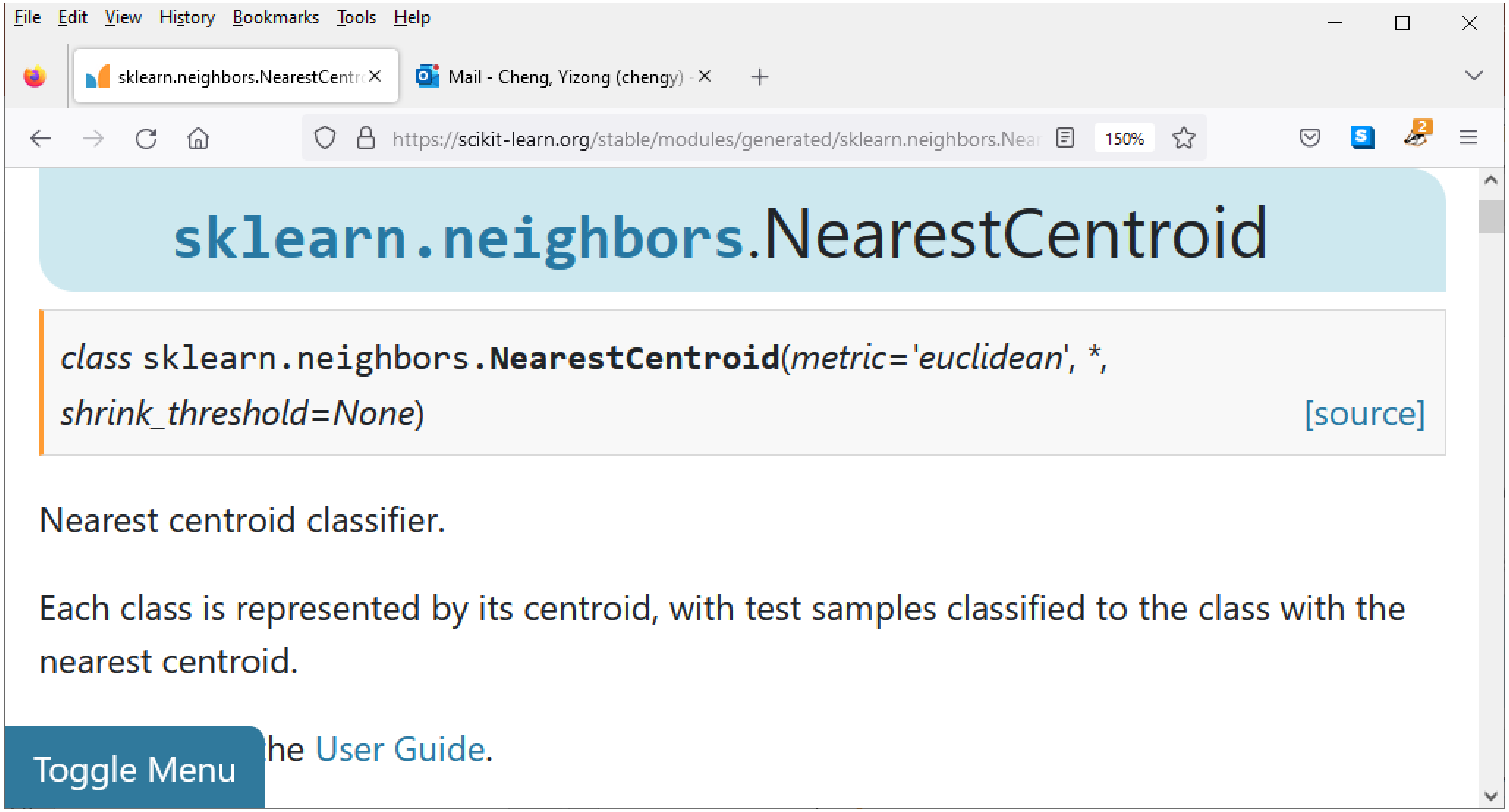
Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters::

n_neighbors : *int, default=5*

Number of neighbors to use by default for `kneighbors` queries.



File Edit View History Bookmarks Tools Help
sklearn.neighbors.NearestCentr
Mail - Cheng, Yizong (chengy)
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.Near
150%

sklearn.neighbors.NearestCentroid

class `sklearn.neighbors.`**`NearestCentroid`**(*metric='euclidean'*, ***, *shrink_threshold=None*) [source]

Nearest centroid classifier.

Each class is represented by its centroid, with test samples classified to the class with the nearest centroid.

he User Guide.

Assignment 14

```
# IR15A.py CS5154/6054 cheng 2022
# Comparing classifiers on documents as binary, count, or tfidf vector
# on two random segments of bible.txt from the first third and last third
# 100 test documents are at the center of 1000 training documents
# Only the four in IIR Chapters 13 and 14 are implemented
# you need to add the ten others imported, too
# Usage: python IR15A.py
```

```
import numpy as np
import random
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

```
f = open("bible.txt", "r")
docs = f.readlines()
f.close()
N =len(docs)
N1 = N // 3 - 1100
c0 = random.randrange(N1)
c1 = N - 1100 - random.randrange(N1)
print('Random segments at -', c0, c1)

trainX = np.concatenate([docs[c0:c0+500],
docs[c0+600:c0+1100],
docs[c1:c1+500], docs[c1+600:c1+1100]])
y = np.concatenate([np.zeros(1000, dtype=np.int16),
np.ones(1000, dtype=np.int16)])
testX = np.concatenate([docs[c0+500:c0+600],
docs[c1+500:c1+600]])
testY = np.concatenate([np.zeros(100, dtype=np.int16),
np.ones(100, dtype=np.int16)])
```

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

# documents as count vectors
cv = CountVectorizer(max_df=0.4, min_df=4)
X1 = cv.fit_transform(trainX).toarray()
T1 = cv.transform(testX).toarray()

# documents as tfidf vectors
cv = TfidfVectorizer(max_df=0.4, min_df=4)
X2 = cv.fit_transform(trainX).toarray()
T2 = cv.transform(testX).toarray()
```

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

model = MultinomialNB()
model.fit(X0, y)
A0 = accuracy_score(testY, model.predict(T0))
model.fit(X1, y)
A1 = accuracy_score(testY, model.predict(T1))
print ('MultinomialNB -', A0, A1)
```

```
model = KNeighborsClassifier()
model.fit(X0, y)
A0 = accuracy_score(testY, model.predict(T0))
model.fit(X1, y)
A1 = accuracy_score(testY, model.predict(T1))
model.fit(X2, y)
A2 = accuracy_score(testY, model.predict(T2))
print ('KNN -', A0, A1, A2)

model = NearestCentroid()
model.fit(X0, y)
A0 = accuracy_score(testY, model.predict(T0))
model.fit(X1, y)
A1 = accuracy_score(testY, model.predict(T1))
model.fit(X2, y)
A2 = accuracy_score(testY, model.predict(T2))
print ('Rocchio -', A0, A1, A2)
```

Introduction to

Information Retrieval

CS276: Information Retrieval and Web Search

Text Classification 1

Chris Manning and Pandu Nayak

Remember: Vector Space Representation

- Each document is a vector, one component for each term (= word).
- Normally normalize vectors to unit length.
- High-dimensional vector space:
  - Terms are axes
  - 10,000+ dimensions, or even 100,000+
  - Docs are vectors in this space

- How can we do classification in this space?

14 *Vector space classification*

CONTIGUITY HYPOTHESIS

The basic hypothesis in using the vector space model for classification is the *contiguity hypothesis*.

Contiguity hypothesis. Documents in the same class form a contiguous region and regions of different classes do not overlap.

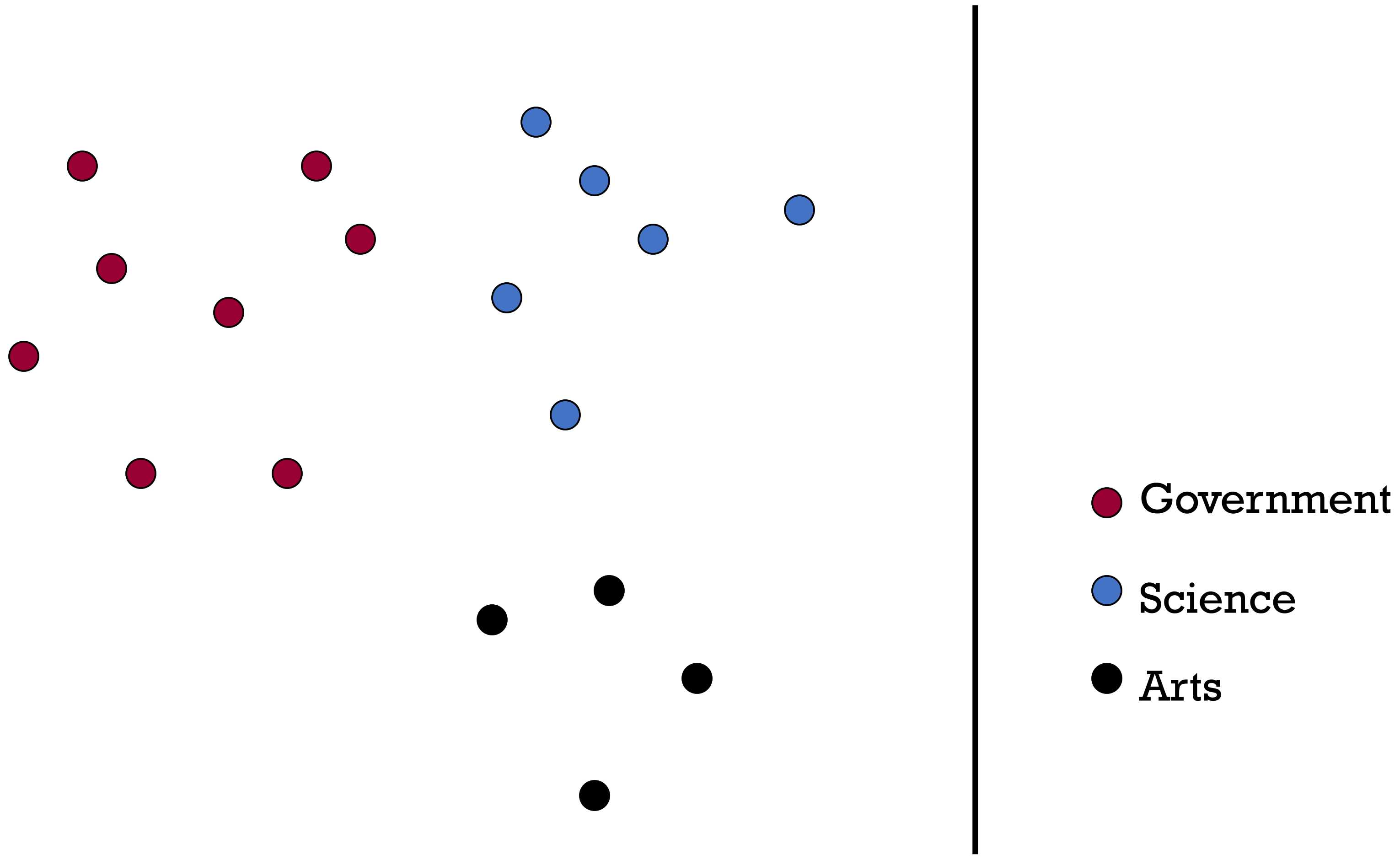
14.2 Rocchio classification

14.3 *k* nearest neighbor

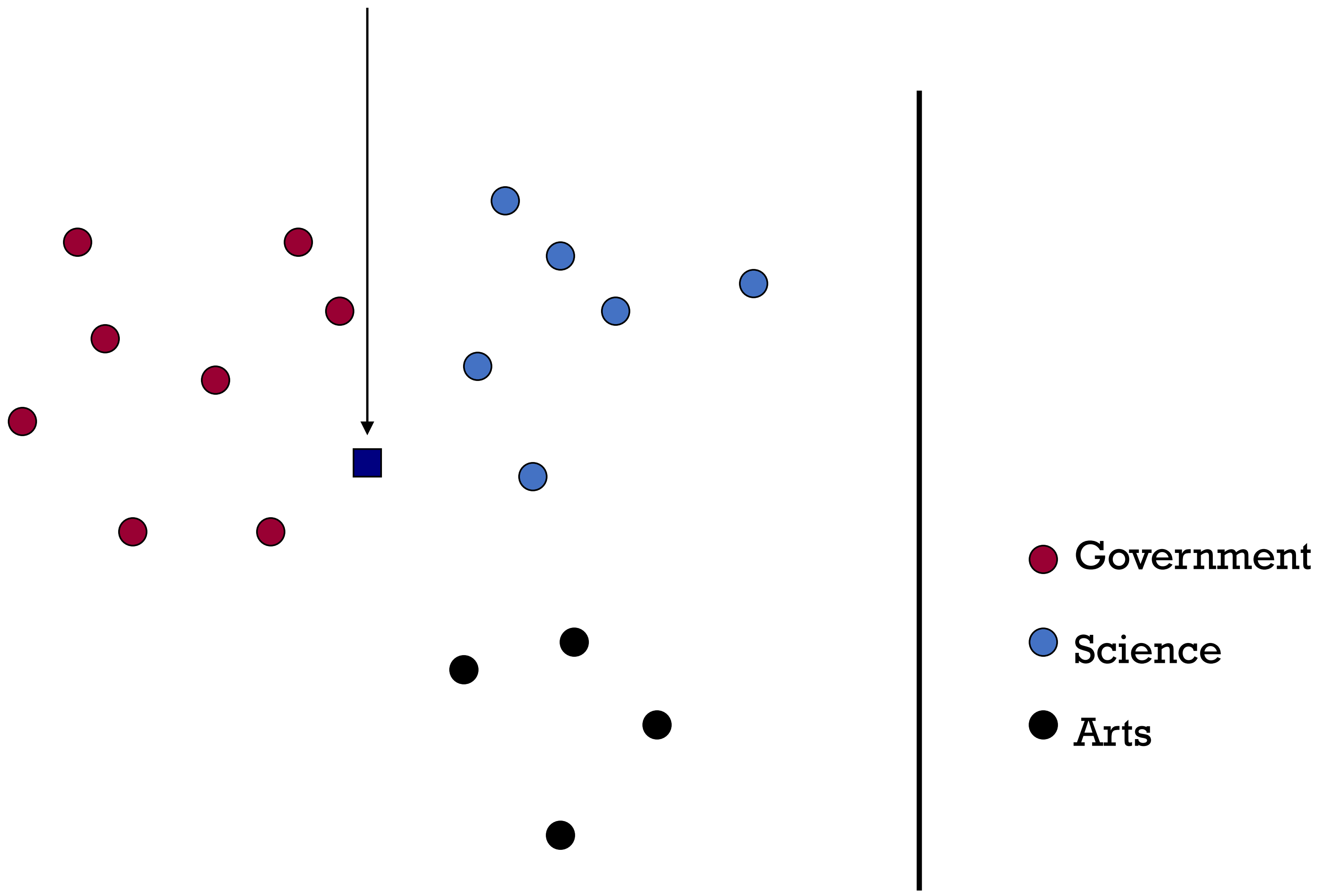
Classification Using Vector Spaces

- In vector space classification, training set corresponds to a labeled set of points (equivalently, vectors)
- Premise 1: Documents in the same class form a contiguous region of space
- Premise 2: Documents from different classes don't overlap (much)
- Learning a classifier: build surfaces to delineate classes in the space

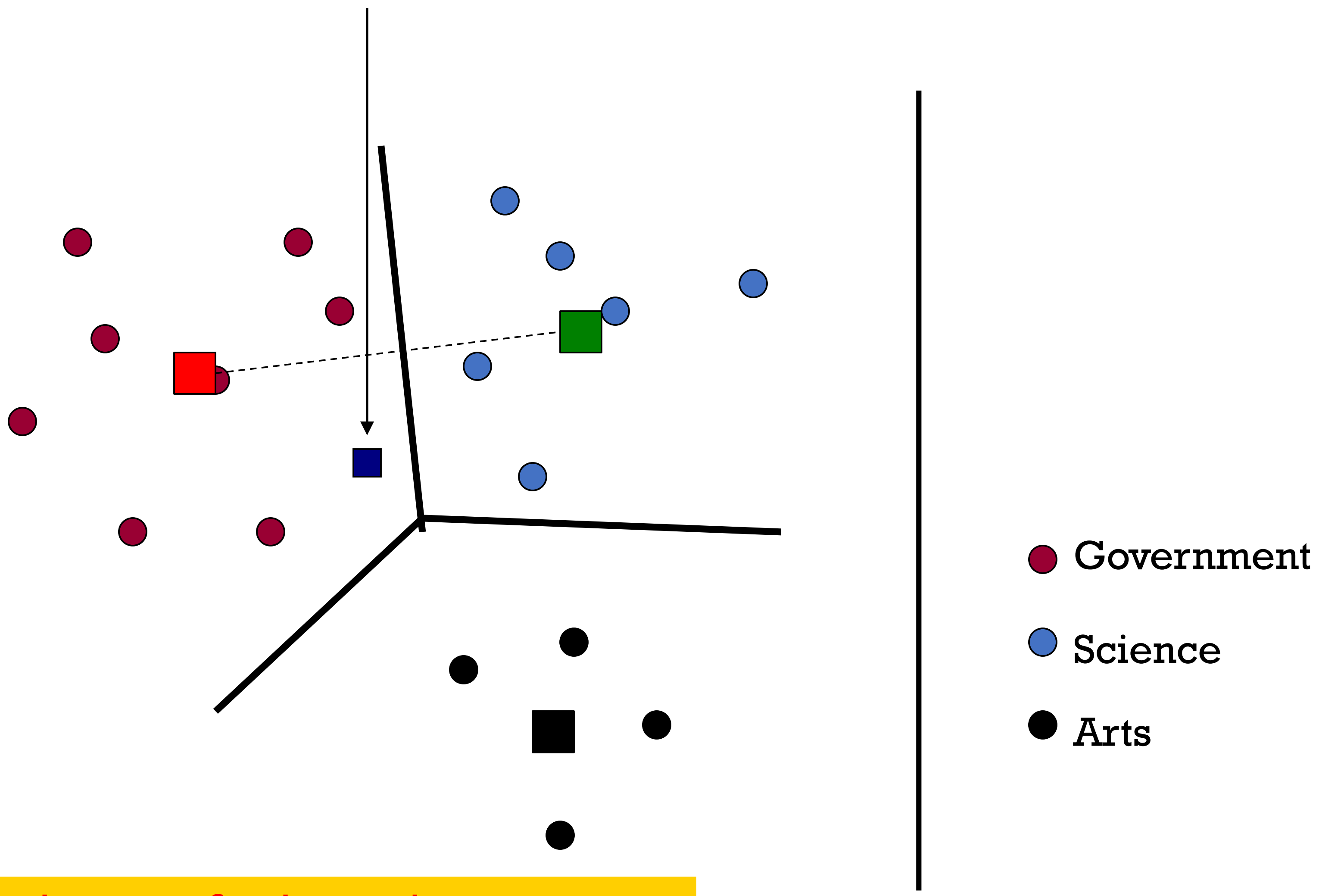
Documents in a Vector Space



Test Document of what class?



Test Document = Government



Our focus: how to find good separators

Definition of centroid

$$\vec{\mu}(c) = \frac{1}{|D_c|}\sum_{d \in D_c} \vec{v}(d)$$

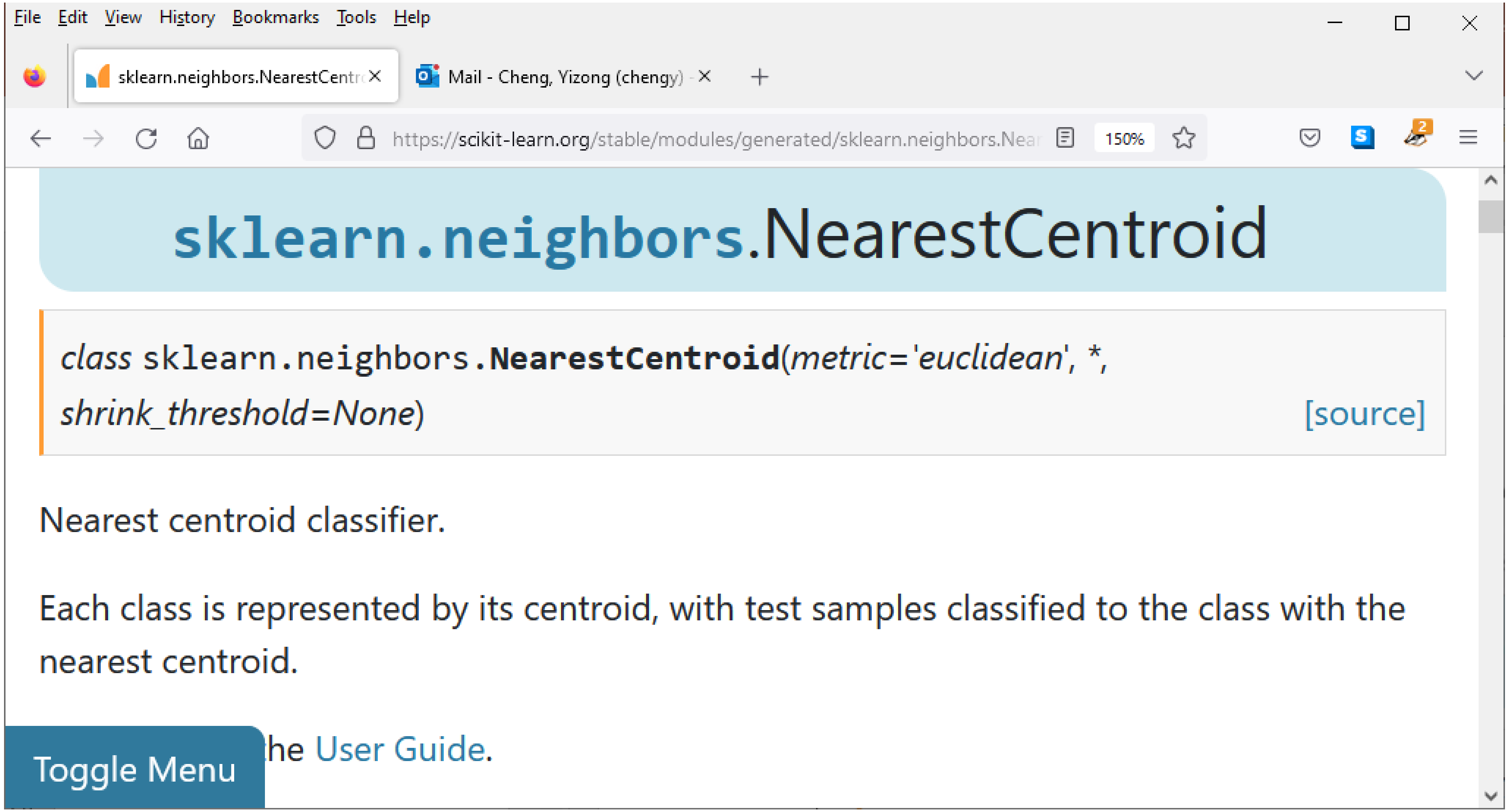
- Where D_c is the set of all documents that belong to class *c* and *v*(*d*) is the vector space representation of *d.*

- *Note that centroid will in general* not *be a unit vector even when the inputs are unit vectors.*

Rocchio classification

- Rocchio forms a simple representative for each class: the centroid/prototype
- Classification: nearest prototype/centroid
- It does not guarantee that classifications are consistent with the given training data

Why not?


File Edit View History Bookmarks Tools Help
sklearn.neighbors.NearestCentr
Mail - Cheng, Yizong (chengy)
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.Near
150%
Toggle Menu

sklearn.neighbors.NearestCentroid

class sklearn.neighbors.**NearestCentroid**(*metric='euclidean'*, *, *shrink_threshold=None*) [source]

Nearest centroid classifier.

Each class is represented by its centroid, with test samples classified to the class with the nearest centroid.

he User Guide.

Two-class Rocchio as a linear classifier

- Line or hyperplane defined by:

$$\overset{M}{\underset{i=1}{\text{å}}} w_i d_i = q$$

- For Rocchio, set:

$$\vec{w} = \vec{\mu}(c_1) - \vec{\mu}(c_2)$$

$$\theta = 0.5 \times (|\vec{\mu}(c_1)|^2 - |\vec{\mu}(c_2)|^2)$$

Rocchio classification

- A simple form of Fisher’s linear discriminant
- Little used outside text classification
  - It has been used quite effectively for text classification
  - But in general worse than Naïve Bayes
- Again, cheap to train and test documents

```
TRAINROCCHIO($\mathbb{C}, \mathbb{D}$)
1  for each $c_j \in \mathbb{C}$
2  do $D_j \leftarrow \{d : \langle d, c_j \rangle \in \mathbb{D}\}$
3       $\vec{\mu}_j \leftarrow \frac{1}{|D_j|} \sum_{d \in D_j} \vec{v}(d)$
4  return $\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}$

APPLYROCCHIO($\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}, d$)
1  return $\arg\min_j |\vec{\mu}_j - \vec{v}(d)|$
```

▶ **Figure 14.4** Rocchio classification: Training and testing.

▶ **Table 13.1** Data for parameter estimation examples.

	docID	words in document	in $c = China$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

	term weights					
vector	Chinese	Japan	Tokyo	Macao	Beijing	Shanghai
$\vec{d}_1$	0	0	0	0	1.0	0
$\vec{d}_2$	0	0	0	0	0	1.0
$\vec{d}_3$	0	0	0	1.0	0	0
$\vec{d}_4$	0	0.71	0.71	0	0	0
$\vec{d}_5$	0	0.71	0.71	0	0	0
$\vec{\mu}_c$	0	0	0	0.33	0.33	0.33
$\vec{\mu}_{\overline{c}}$	0	0.71	0.71	0	0	0

▶ **Table 14.1** Vectors and class centroids for the data in Table 13.1.

Example 14.1: Table 14.1 shows the tf-idf vector representations of the five documents in Table 13.1 (page 261), using the formula $(1 + \log_{10} \mathrm{tf}_{t,d}) \log_{10}(4/\mathrm{df}_t)$ if $\mathrm{tf}_{t,d} > 0$ (Equation (6.14), page 127). The two class centroids are $\mu_c = 1/3 \cdot (\vec{d}_1 + \vec{d}_2 + \vec{d}_3)$ and $\mu_{\overline{c}} = 1/1 \cdot (\vec{d}_4)$. The distances of the test document from the centroids are $|\mu_c - \vec{d}_5| \approx 1.15$ and $|\mu_{\overline{c}} - \vec{d}_5| = 0.0$. Thus, Rocchio assigns d_5 to $\overline{c}$.

The separating hyperplane in this case has the following parameters:

$$\begin{aligned} \vec{w} &\approx (0 \; -0.71 \; -0.71 \; 1/3 \; 1/3 \; 1/3)^T \\ b &= -1/3 \end{aligned}$$

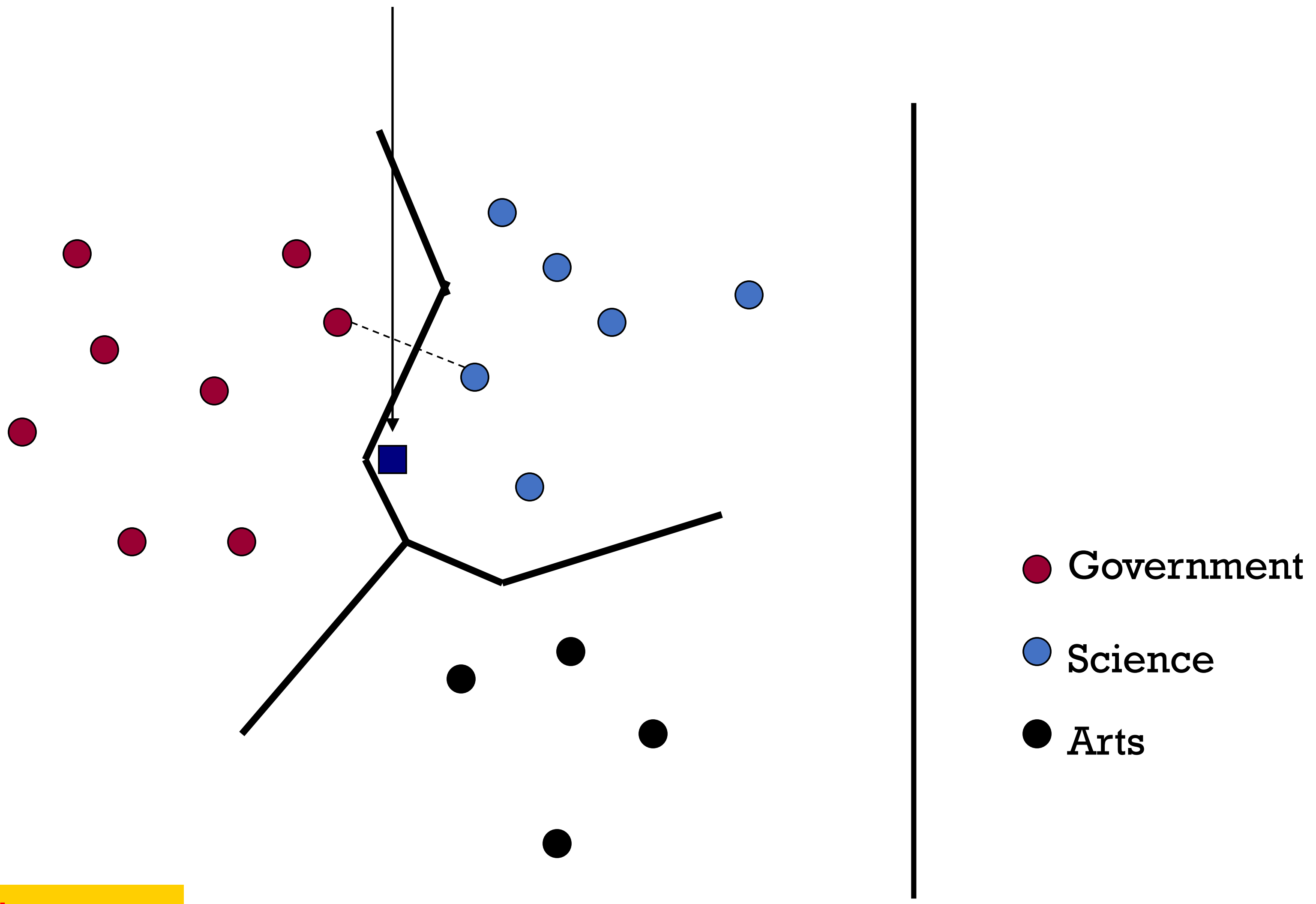
See Exercise 14.15 for how to compute $\vec{w}$ and b. We can easily verify that this hyperplane separates the documents as desired: $\vec{w}^T\vec{d}_1 \approx 0 \cdot 0 + -0.71 \cdot 0 + -0.71 \cdot 0 + 1/3 \cdot 0 + 1/3 \cdot 1.0 + 1/3 \cdot 0 = 1/3 > b$ (and, similarly, $\vec{w}^T\vec{d}_i > b$ for $i = 2$ and $i = 3$) and $\vec{w}^T\vec{d}_4 = -1 < b$. Thus, documents in c are above the hyperplane ($\vec{w}^T\vec{d} > b$) and documents in $\overline{c}$ are below the hyperplane ($\vec{w}^T\vec{d} < b$).

k Nearest Neighbor Classification

- kNN = *k* Nearest Neighbor

- To classify a document *d*:
- Define *k*-neighborhood as the *k* nearest neighbors of *d*
- Pick the majority class label in the *k*-neighborhood
- For larger *k* can roughly estimate P(c|*d*) as #(c)/k

Test Document = Science



Voronoi diagram

Nearest-Neighbor Learning

- Learning: just store the labeled training examples *D*
- Testing instance *x (under 1NN)*:
  - Compute similarity between *x* and all examples in *D*.
  - Assign *x* the category of the most similar example in *D*.
- Does not compute anything beyond storing the examples
- Also called:
  - Case-based learning
  - Memory-based learning
  - Lazy learning
- Rationale of kNN: contiguity hypothesis

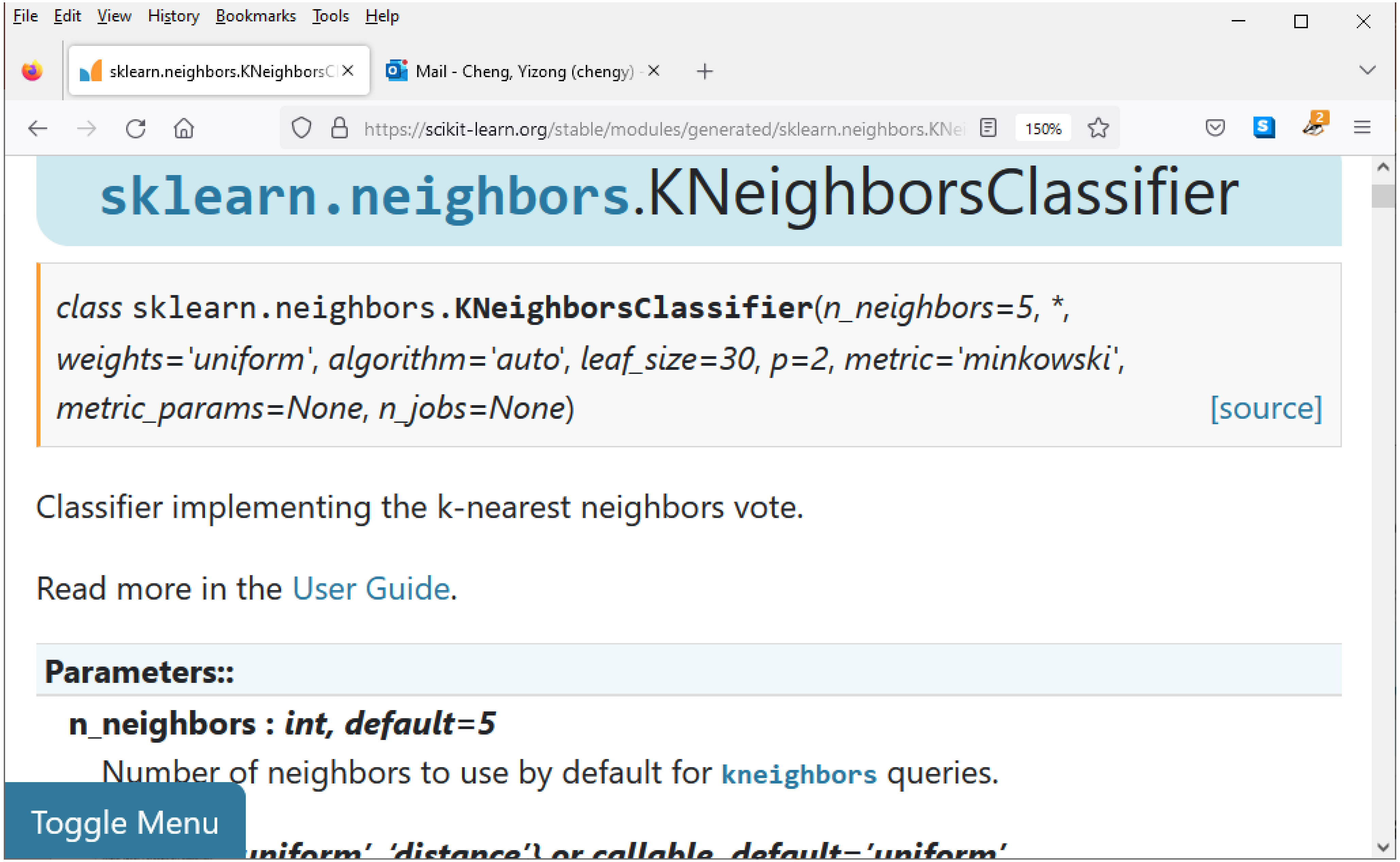
```
TRAIN-KNN($\mathbb{C}, \mathbb{D}$)
1  $\mathbb{D}' \leftarrow$ PREPROCESS($\mathbb{D}$)
2  $k \leftarrow$ SELECT-K($\mathbb{C}, \mathbb{D}'$)
3  return $\mathbb{D}', k$

APPLY-KNN($\mathbb{C}, \mathbb{D}', k, d$)
1  $S_k \leftarrow$ COMPUTENEARESTNEIGHBORS($\mathbb{D}', k, d$)
2  for each $c_j \in \mathbb{C}$
3  do $p_j \leftarrow |S_k \cap c_j|/k$
4  return $\arg\max_j p_j$
```

▶ **Figure 14.7** kNN training (with preprocessing) and testing. p_j is an estimate for $P(c_j|S_k) = P(c_j|d)$. c_j denotes the set of all documents in the class c_j.

	term weights					
vector	Chinese	Japan	Tokyo	Macao	Beijing	Shanghai
$\vec{d}_1$	0	0	0	0	1.0	0
$\vec{d}_2$	0	0	0	0	0	1.0
$\vec{d}_3$	0	0	0	1.0	0	0
$\vec{d}_4$	0	0.71	0.71	0	0	0
$\vec{d}_5$	0	0.71	0.71	0	0	0
$\vec{\mu}_c$	0	0	0	0.33	0.33	0.33
$\vec{\mu}_{\overline{c}}$	0	0.71	0.71	0	0	0

▶ **Table 14.1** Vectors and class centroids for the data in Table 13.1.

Example 14.2: The distances of the test document from the four training documents in Table 14.1 are $|\vec{d}_1 - \vec{d}_5| = |\vec{d}_2 - \vec{d}_5| = |\vec{d}_3 - \vec{d}_5| \approx 1.41$ and $|\vec{d}_4 - \vec{d}_5| = 0.0$. d_5's nearest neighbor is therefore d_4 and 1NN assigns d_5 to d_4's class, $\overline{c}$.


File Edit View History Bookmarks Tools Help
sklearn.neighbors.KNeighborsC
Mail - Cheng, Yizong (chengy)
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNe
150%
Toggle Menu

sklearn.neighbors.KNeighborsClassifier

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5*, ***, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters::

n_neighbors : *int, default=5*

Number of neighbors to use by default for `kneighbors` queries.

uniform', 'distance'} or callable, default='uniform'

k Nearest Neighbor

- Using only the closest example (1NN) is subject to errors due to:
  - A single atypical example.
  - Noise (i.e., an error) in the category label of a single training example.
- More robust: find the *k* examples and return the majority category of these *k*
- *k* is typically odd to avoid ties; 3 and 5 are most common

kNN: Discussion

- No feature selection necessary
- No training necessary
- Scales well with large number of classes
  - Don't need to train *n* classifiers for *n* classes
- Classes can influence each other
  - Small changes to one class can have ripple effect
- Done naively, very expensive at test time
- In most cases it's more accurate than NB or Rocchio
  - As the amount of data goes to infinity, it has to be a great classifier! – it's "Bayes optimal"