

Speechresearch/classifiers

API Documentation

December 8, 2010

Contents

Contents	1
1 Package classifiers	4
1.1 Modules	4
1.2 Variables	4
2 Module classifiers.data_splitter	5
2.1 Functions	5
2.2 Variables	5
2.3 Class bluedata	5
2.3.1 Methods	5
2.3.2 Properties	6
2.4 Class bluedata_groups	6
2.4.1 Methods	6
2.4.2 Properties	7
2.4.3 Class Variables	7
3 Module classifiers.g_class_centers	8
3.1 Functions	8
3.2 Variables	8
4 Module classifiers.g_kmeans	9
4.1 Functions	9
4.2 Variables	9
4.3 Class cluster_descriptor	9
4.3.1 Methods	9
5 Module classifiers.l_class_show	10
5.1 Functions	10
5.2 Variables	10
6 Module classifiers.l_classifier_guts	11
6.1 Functions	11
6.2 Variables	12
6.3 Class l_classifier_desc	12
6.3.1 Methods	12
6.3.2 Properties	14

6.4	Class <code>l_classifier</code>	14
6.4.1	Methods	14
6.4.2	Properties	15
6.4.3	Class Variables	16
7	Module <code>classifiers.lx_classifier</code>	17
7.1	Functions	17
7.2	Variables	17
8	Module <code>classifiers.multivariate</code>	18
8.1	Functions	18
8.2	Variables	18
9	Module <code>classifiers.q_classiboost</code>	19
9.1	Functions	19
9.2	Variables	19
10	Module <code>classifiers.q_classifier_mm</code>	20
10.1	Functions	20
10.2	Variables	20
11	Module <code>classifiers.q_classifier_r</code>	21
11.1	Functions	21
11.2	Variables	26
11.3	Class <code>datum_c</code>	26
11.3.1	Methods	26
11.3.2	Properties	27
11.3.3	Instance Variables	27
11.4	Class <code>datum_tr</code>	27
11.4.1	Methods	27
11.4.2	Properties	28
11.4.3	Instance Variables	28
11.5	Class <code>grouper_c</code>	28
11.5.1	Methods	29
11.5.2	Properties	29
11.6	Class <code>model_template</code>	29
11.6.1	Methods	30
11.6.2	Properties	30
11.7	Class <code>qmodel</code>	30
11.7.1	Methods	30
11.7.2	Properties	31
11.8	Class <code>lmodel</code>	31
11.8.1	Methods	32
11.8.2	Properties	32
11.9	Class <code>classifier_desc</code>	32
11.9.1	Methods	33
11.9.2	Properties	34
11.10	Class <code>classifier</code>	34
11.10.1	Methods	35
11.10.2	Properties	36
11.10.3	Class Variables	36
11.11	Class <code>evaluate_match_w_rare</code>	36

11.11.1 Methods	37
11.11.2 Properties	37
11.11.3 Class Variables	37
12 Module classifiers.qd_classifier_guts	38
12.1 Functions	38
12.2 Variables	38
12.3 Class qd_classifier_desc	38
12.3.1 Methods	39
12.3.2 Properties	40
12.4 Class qd_classifier	40
12.4.1 Methods	40
12.4.2 Properties	41
12.4.3 Class Variables	41
13 Module classifiers.read_classified	42
13.1 Functions	42
13.2 Variables	43
14 Module classifiers.sim_cen_gauss_class	44
14.1 Functions	44
14.2 Variables	45
15 Script script-l_classifier	46
15.1 Variables	47
16 Script script-qd_classifier	48
16.1 Variables	48
17 Script script-qdg_classifier	49
17.1 Functions	49
17.2 Variables	49
Index	51

1 Package classifiers

1.1 Modules

- **data_splitter** (*Section 2, p. 5*)
- **g_class_centers**: This computes centers for classes.
(*Section 3, p. 8*)
- **g_kmeans** (*Section 4, p. 9*)
- **Lclass_show**: Reads in the output of l.classifier.py and prepares it for display.
(*Section 5, p. 10*)
- **Lclassifier_guts**: A classifier that assumes that P is linear in position.
(*Section 6, p. 11*)
- **lx_classifier** (*Section 7, p. 17*)
- **multivariate** (*Section 8, p. 18*)
- **q_classiboost**: Reads the *.fiat file produced by q.classifier.py that contains the decisions of the individual classifiers in the forest.
(*Section 9, p. 19*)
- **q_classifier_mm**: Subroutines for q.classifier.py that implement a multi-mu model.
(*Section 10, p. 20*)
- **q_classifier_r**: This is a support module, used by many types of classifiers.
(*Section 11, p. 21*)
- **qd_classifier_guts**: A classifier that assumes that $\log(P)$ is a quadratic form.
(*Section 12, p. 38*)
- **read_classified**: This module reads the outputs of qd.classifier.py, specifically classified.fiat and classes.chunk.
(*Section 13, p. 42*)
- **sim_cen_gauss.class**: Simple centered Gaussian Classifier.
(*Section 14, p. 44*)

1.2 Variables

Name	Description
__package__	Value: 'classifiers'

2 Module `classifiers.data_splitter`

Note: The split into training set and test set is completely deterministic, depending only on each datum's identifier, the total number of data. (If you use a single instance of `bluedata` or `bluedata_groups` to make several splits, then each split will be different. But if you created a new instance, you'd get the exact same sequence of splits.)

2.1 Functions

<code>Hash(x)</code>

2.2 Variables

Name	Description
<code>--package--</code>	Value: <code>'classifiers'</code>

2.3 Class `bluedata`

object `classifiers.data_splitter.bluedata`

This is an extension of `gmisc.lib.blue_data_selector.bluedata` which is aware of the classids of data. It tries to split the data to take the same fraction of each class.

2.3.1 Methods

<code>__init__(self, data, blueness=2.0)</code>

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Parameters

`data`: (type=*a sequence of `datum_tr` objects.*)

Overrides: `object.__init__`

<code>split(self, n, seed=0)</code>

Split the stored data into a group of `n` and the remainder. Successive calls to `split()` will tend to (as nearly as possible) put different data into the first list.

Parameters

`n`: (type=*int or float*)

Return Value

tuple(list of length `n`(or nearly), list of remainder)

<code>__len__(self)</code>

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

2.3.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

2.4 Class `bluedata_groups`**2.4.1 Methods**

<code>__init__(self, d, gr)</code>
A 'grouper' function takes a DUID (a unique i.d. string for a datum) and returns the name of the data group to which it belongs. This group name is used in constructing the training and test sets.
Parameters
d: a list of data (<i>type=list(datum_c)</i>)
Return Value
A dictionary full of lists; each list corresponds to one group. (<i>type=dictops.dict_of_lists</i>)
Overrides: <code>object.__init__</code>

<code>__len__(self)</code>

split(*self*, *n*, *seed*=0)

Take groups until you have accumulated approximately *n* data. The trick is to make all classifiers as identical as possible by getting the same balance of classes in the training set. We are willing to sacrifice some data to do that.

Parameters

n: how many data in the testing set
(*type*=float or int)

Return Value

(testing set, training set, names in training set)
(*type*=tuple(list(*datum_tr*), list(*datum_tr*), list(*str*)))

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

2.4.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

2.4.3 Class Variables

Name	Description
HUGE	Value: 1e+30

3 Module `classifiers.g_class_centers`

This computes centers for classes. You give it a `classified.fiat` file on the argument list, as obtained from `*_classifier.py`, and it produces a list of files, class names, and the feature vector at the center of each class.

To define the center, it takes the component-by-component median of the feature vector, including only data that are correctly classified.

3.1 Functions

<code>process(<i>f</i>)</code>

<code>format(<i>v</i>)</code>

3.2 Variables

Name	Description
CLIP	Value: 0.15
<code>--package--</code>	Value: 'classifiers'

4 Module `classifiers.g_kmeans`

4.1 Functions

```
find_which_cluster(datum, cpos, distfcn)
```

```
find_center(i, membership, data, clip)
```

```
kmeans(data, Ncl, distfcn=None, Nit=None, clip=None)
```

```
read_data(fd)
```

Reads in feature vectors, each with a uid as a comment.

```
euclid(a, b)
```

4.2 Variables

Name	Description
HUGE	Value: 1e+30
Clip	Value: None
__package__	Value: 'classifiers'

4.3 Class `cluster_descriptor`

4.3.1 Methods

```
__init__(self, center, membership, size)
```

5 Module `classifiers.lclass_show`

Reads in the output of `lclassifier.py` and prepares it for display. NOTE: this only makes sense for two classes.

5.1 Functions

<code>ssv(<i>a</i>)</code>

<code>doit(<i>fn</i>)</code>

5.2 Variables

Name	Description
<code>--package--</code>	Value: <code>'classifiers'</code>

6 Module `classifiers.l.classifier_guts`

A classifier that assumes that P is linear in position. This is known as a (linear) logistic discriminant analysis:

Notes:

- A useful general reference is: `@inbook{webb:spr:logistic, author = {Andrew Webb}, title = {Statistical Pattern Recognition}, pages = {124–132}, year = {1999}, publisher = {Arnold}, address = {London, New York}, note = {ISBN 0 340 74164 3} }`
- This code was described in an appendix to "Dimensions of durational variation in speech", by Anastassia Loukina, Greg Kochanski, Burton Rosner, Chilin Shih, and Elinor Keane, submitted 2010 to J. Acoustical Society of America.

6.1 Functions

```
ref_from_data(data)
```

```
ref_from_models(models)
```

```
go_auto(fd, coverage=3, ftest=0.25, n_per_dim=10, ftrim=None,
        verbose=True, modify_class=None)
```

```
test_build1(classifier_choice)
```

```
test_build1n(classifier_choice)
```

```
test_build1t(classifier_choice)
```

```
test_build32(classifier_choice)
```

```
test_build2(classifier_choice)
```

```
test_4_2(classifier_choice)
```

```
test_2_bias(classifier_choice)
```

```
test_2_scale(classifier_choice)
```

```
test_var()
```

```
calibrate_var()
```

```
test()
```

6.2 Variables

Name	Description
COVERAGE	Value: 3
FTEST	Value: 0.25
N_PER_DIM	Value: 10
PSYCO	Value: False
--package--	Value: 'classifiers'

6.3 Class `Lclassifier_desc`



This class is metadata and helper functions for a linear discriminant classifier.

6.3.1 Methods

```
__init__(self, data=None, evaluator=None, models=None, ftrim=None)
```

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)

```
np(self)
```

The number of parameters required to define each class.

Return Value

The number of parameters required to define one class.

(*type=int*)

Overrides: `classifiers.q_classifier_r_classifier_desc.np`

npt(*self*)

The total number of parameters required to define all classes.

Return Value

The number of parameters required to define the classifier.

(*type=int*)

Overrides: `classifiers.q_classifier_r.classifier_desc.npt`

unpack(*self*, *prmvec*, *trainingset_name=None*, *uid=None*)

Produce a classifier from a parameter vector.

Parameters

prmvec: a vector of parameters that describe a classifier model.

Return Value

the classifier.

(*type=the corresponding subclass of classifier.*)

Overrides: `classifiers.q_classifier_r.classifier_desc.unpack`

start(*self*, *data*)

Starting position for Markov Chain Monte Carlo.

Overrides: `classifiers.q_classifier_r.classifier_desc.start`

typename(*self*)**Return Value**

a string that names the subclass - what kind of `classifier_desc` is it? @rtype str

Overrides: `classifiers.q_classifier_r.classifier_desc.typename` `exitit`(inherited documentation)

__repr__(*self*)

`str(x)`

Overrides: `object.__repr__` `exitit`(inherited documentation)

__str__(*self*)

`str(x)`

Overrides: `object.__str__` `exitit`(inherited documentation)

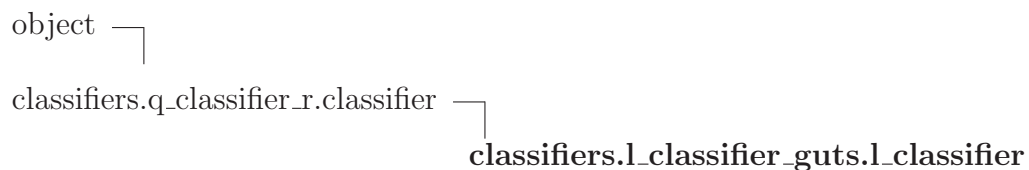
Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,
`__setattr__()`, `__sizeof__()`, `__subclasshook__()`

6.3.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

6.4 Class `Lclassifier`



This is a linear discriminant classifier.

6.4.1 Methods

`__init__(self, cdesc=None, models=None, trainingset_name=None, uid=None)`
`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Parameters

- `models`: a dictionary containing a probabilistic model for each class
- `info`: not used in the internal operation of the classifier, but it is stuff that is important to write out.

Overrides: `object.__init__` extit(inherited documentation)

`P(self, datum, whichclass=None)`

Determine the probability of being in each class. If `whichclass=None`, then it returns a list of tuples `[(classname,P), ...]` for all classes. Otherwise, it returns the probability of the specified class.

`__repr__(self)`

`str(x)`

Overrides: `object.__repr__` extit(inherited documentation)

__str__(*self*)

str(*x*)

Overrides: object.__str__ exitit(inherited documentation)

add_info(*self, k, v*)

add_model(*self, classname, model*)

Add a class to an existing classifier.

bestc(*self, datum*)

Determine the best class for a datum.

list_classes(*self*)

list_wrong_classifications(*self, classdata*)

logP(*self, datum, whichclass=None*)

logPv(*self, datum*)

Determine the probability of being in each class. If *whichclass=None*, then it returns a list of tuples [(classname,P), ...] for all classes.

logPw(*self, datum, whichclass, constrain*)

Determine the probability of *datum* being in *whichclass*.

writer(*self, dcw*)

Write this classifier out (usually to a data file).

Parameters

dcw: (*type=chunkio.chunk_w*)

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,
`__setattr__()`, `__sizeof__()`, `__subclasshook__()`

6.4.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>--class--</code>	

6.4.3 Class Variables

Name	Description
HUGE	Value: 1e+30

7 Module *classifiers.lx_classifier*

7.1 Functions

```
go_cross(trfd, tsfd, n_per_dim=6, ftrim=0.0)
```

7.2 Variables

Name	Description
N_PER_DIM	Value: 6
..package..	Value: 'classifiers'

8 Module `classifiers.multivariate`

8.1 Functions

`meanvar(dataset, N, modelchoice=<class gmisclib.multivariate.q.quadratic at 0x1ed3290>)`

Given a dataset, this produces a bunch of MCMC estimates of plausible means and inverse covariance matrices for the dataset. Modelchoice is a particular size model, e.g. an instance of `quadratic`. The output is in the form [(mean, inv_covar), ...].

`test_quadratic()`

`test_diag_quadratic()`

`test_multimu()`

`test_multimu_diag()`

8.2 Variables

Name	Description
ERGCOVER	Value: 3.0
HUGE	Value: 1e+30
FromName	Value: {'quadratic': MQ.quadratic, 'diag_quadratic': MQ.diag_qua...
__package__	Value: 'classifiers'

9 Module *classifiers.q_classiboost*

Reads the *.fiat file produced by q_classifier.py that contains the decisions of the individual classifiers in the forest. Returns the majority decision.

9.1 Functions

index (<i>individuals</i>)

Takes a list of decisions on an individual utterance from an individual classifier (as from *.fiat from q_classifier.py) and indexes them by the utterance.

vote (<i>individuals</i>)

9.2 Variables

Name	Description
DEFAULTS	Value: {}
--package--	Value: 'classifiers'

10 Module `classifiers.q_classifier_mm`

Subroutines for `q_classifier.py` that implement a multi-mu model. That is, one where all the classes share a common covariance matrix.

10.1 Functions

<code>count_the_classes(<i>data</i>)</code>

How many examples of each class do we have?

<code>build_classifiers(<i>data</i>, <i>N</i>, <i>modelchoice</i>=M.multi_mu)</code>
--

Builds a set of quadratic classifiers. Data is a list of <code>datum_tr</code> . It returns a list: [(<code>evaluation_result</code> , <code>classifier_description</code>), ...] .

10.2 Variables

Name	Description
HUGE	Value: 1e30

11 Module *classifiers.q_classifier_r*

This is a support module, used by many types of classifiers.

11.1 Functions

Hash (<i>dl</i>)
<hr/>
Parameters <i>dl</i> : A list of data (<i>type=list(datum_c)</i>)
Return Value a hash of the UID's of data items. (<i>type=str</i>)

Hash1 (<i>l</i>)
<hr/>
Parameters <i>l</i> : A list of data
Return Value a hash of a vector. (<i>type=str</i>)

prior (<i>training</i>)
<hr/>
This computes the probability of correct classification, assuming you can't see the feature vector. It is used to compute P(chance). It assumes that you choose class C with probability 1 if P(C) is the biggest among all the classes.

max_correct (<i>training, testing</i>)
<hr/>
This is a hard, conservative upper limit for the probability of correct classification.

read_data(*fd*, *commentarray*=None)

Reads in feature vectors where the first element is the true class. This is the main data input for `l_classifier`, `qdg_classifier` and `qd_classifier`.

Parameters

fd: (type=*file*)
commentarray: (type=*a list or None*)

Return Value

`list(datum_tr)`

get_dim(*fd*)

This function takes a list of data (type `datum_tr`) and makes sure that they all have the same length feature vector. If so, it reports the length (dimension) of the feature vector.

Parameters

fd: input data
 (type=`list(datum_tr)`)

Return Value

length of vectors
 (type=`int`)

compute_cross_class(*training*, *testing*, *modelchoice*=None, *n_per_dim*=None, *builder*=None, *classout*=None, *trainingset_name*=None, *modify_class*=None, *verbose*=True)

Build classifiers based on the training set, and test them on the testing set. Modelchoice here is the completed class object, not a closure.

compute_self_class(*d*, *coverage*=None, *f_test*=None, *modelchoice*=None, *n_per_dim*=None, *modify_class*=None, *builder*=None, *classout*=None, *verbose*=True)

modelchoice here is expected to take one argument– the data.

list_groups(*d*, *gr*)

```
compute_group_class(dg, modelchoice=None, n_per_dim=None,
builder=None, classout=None, ftest=None, grouper=None, coverage=None,
modify_class=None, verbose=True)
```

This function makes sure that the training set and testing set come from different groups. The 'grouper' returns a group name, when given a datum. Modelchoice is expected to take one argument, the training set.

Parameters

grouper: function returning a group name for each datum
(type=function from datum_tr to str)

```
qzmodel(ndim)
```

```
lzmodel(ndim)
```

```
forest_build(data, N, modelchoice=None, trainingset_name=None)
```

Build a forest of classifiers.

Parameters

data: data to train the classifiers on.
(type=datum_c)

N: How many to build.
(type=int,)

modelchoice: what kind of classifier to build
(type=subclass of model_template)

trainingset name: (stored for later use).
(type=str)

evaluate_match(*cl, data*)

This can be passed into a classifier descriptor as the evaluate argument. It returns the number of exact matches between the classified data and the input, true classification.

Parameters

cl: some classifier that has a `bestc()` method.
(*type=typically a subclass of classifier.*)

data: list of classes that describe data points.
(*type=Typically a subclass of datum_c.*)

Return Value

the number of classification errors made
(*type=int*)

evaluate_Bayes(*cl, data, constrain=0.0*)

Evaluates the negative log of the probability that the classifier would assign to the datum being in the observed class (i.e. whatever class is specified in the `datum.tr`). Obviously, you want this to be a relatively small number.

Parameters

cl: a classifier
(*type=classifier*)

data: data
(*type=list(datum_c)*)

Return Value

the log of the probability of being in the observed class.

If `cdesc.ftrim` is not `None`, we assume that some of the data in each class are dubious, and should be ignored if they are sufficiently improbable. We modify the probability scores of data that is among the worst (`cl.cdesc.ftrim[0]` fraction), and limit those scores to be no larger than `cl.cdesc.ftrim[1]` larger than the best score. This lets you limit by score or limit by fraction or any mixture in between. If `cl.cdesc.ftrim` is `None`, then no limiting or trimming is done.

(*type=float*)

default_writer(*summary, out, classout, wrong, fname='classes.chunk'*)

This writes out classifiers to a data file.

Attention: `out` needs to be a list, not an iterator, because we use it twice.

count_classes(*data*)

Count how many instances there are of each class.

Parameters

data: (*type=datum_c @rtype map from str to int*)

list_classes(*data*)

List the names of the classes in a dataset, with the most populous classes first.

Parameters

data: (*type=datum_c @rtype list(str)*)

name_of_evaluator(*e*)

Used to get the name of an evaluator, to write it to a file header.

Parameters

e: (*type=function, preferable with `--name--` attribute.*)

Return Value

str

evaluator_from_name(*nm*)

Maps a name to a function that will evaluate how well a classifier performs.

Parameters

nm: a printable name
(*type=str*)

Return Value

a function

default_modify_class(*qc, training_counts, testing_counts*)

Modifies a classifier so it isn't so dominated by the most frequent classes.

Parameters

training_counts: how many data are there in each class in the training set
(*type=map str to int*)

testing_counts: how many data are there in each class in the testing set
(*type=map str to int*)

qc: (*type=classifier*)

11.2 Variables

Name	Description
ERGCOVER	Value: 4.0
D	Value: False
CONSTRAIN	Value: 1e-06
<code>--package--</code>	Value: 'classifiers'

11.3 Class `datum_c`

object —
 `classifiers.q_classifier_r.datum_c`

Known Subclasses: `classifiers.q_classifier_r.datum_tr`

This is an unclassified datum, either in the test or training set.

11.3.1 Methods

<code>--init__(self, vector, uid=None)</code> <hr/> <code>x.__init__(...)</code> initializes x; see <code>x.__class__.__doc__</code> for signature Parameters vector: a vector of numbers uid: an arbitrary identifier for a datum (<i>type=</i> <code>str</code>) Overrides: <code>object.__init__</code>
--

<code>--repr__(self)</code> <code>repr(x)</code> Overrides: <code>object.__repr__</code> <code>exitit</code> (inherited documentation)

<code>--cmp__(self, other)</code>
--

Inherited from `object`

`--delattr__()`, `--format__()`, `--getattr__()`, `--hash__()`, `--new__()`, `--reduce__()`, `--reduce_ex__()`,
`--setattr__()`, `--sizeof__()`, `--str__()`, `--subclasshook__()`

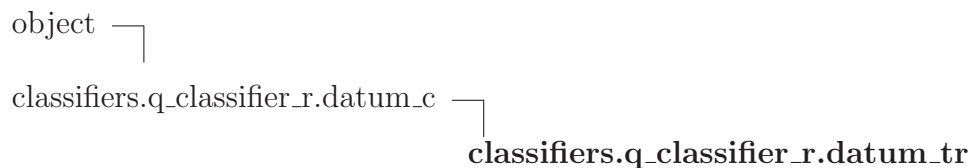
11.3.2 Properties

Name	Description
<i>Inherited from object</i> <code>__class__</code>	

11.3.3 Instance Variables

Name	Description
<code>uid</code>	
<code>value</code>	

11.4 Class `datum_tr`



This is a datum where we know the true class, presumably in the training set.

11.4.1 Methods

<code>__init__(self, vector, classid, uid=None)</code>
<code>x.__init__(...)</code> initializes <code>x</code> ; see <code>x.__class__.__doc__</code> for signature
Parameters
vector: a vector of numbers
uid: an arbitrary identifier for a datum (<i>type=</i> <code>str</code>)
classid: is the true class, (<i>type=</i> <code>str</code>)
Overrides: <code>object.__init__</code>

<code>__str__(self)</code>
<code>repr(x)</code>
Overrides: <code>object.__str__</code> <code>__extit__</code> (inherited documentation)

<pre> __repr__(self) repr(x) Overrides: object.__repr__ extit(inherited documentation) </pre>

<pre> __cmp__(self, other) </pre>

Inherited from object

```

__delattr__(), __format__(), __getattr__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(),
__setattr__(), __sizeof__(), __subclasshook__()

```

11.4.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

11.4.3 Instance Variables

Name	Description
classid	
uid	
value	

11.5 Class grouper_c

```

object ┌
      │
      └─ classifiers.q_classifier_r.grouper_c

```

A 'grouper' function takes a DUID (a unique i.d. string for a datum) and returns the name of the data group to which it belongs. This group name is used in constructing the training and test sets.

11.5.1 Methods

```
__init__(self, pattern='^([^\s]*)/', which=1)
```

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `object.__init__` extit(inherited documentation)

```
__call__(self, x)
```

Parameters

`x`: a datum

(*type=datum_c*)

Return Value

group name

(*type=str*)

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

11.5.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

11.6 Class model_template

```

object ┌
      │
      └─ classifiers.q_classifier_r.model_template

```

Known Subclasses: `classifiers.q_classifier_r.lmodel`, `classifiers.q_classifier_r.qmodel`

This class describes how to compute the relative probability that a datum is a member of a particular class. It also knows how to package up all necessary information for storage in a file.

logp(*self*, *datum*)

Overrides: classifiers.q_classifier_r.model_template.logp

tochunk(*self*, *chunkwriter*)

Overrides: classifiers.q_classifier_r.model_template.tochunk

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

fromchunk(*chunk*)

This the inverse of qmodel.tochunk(), except the group start is already read.

Overrides: classifiers.q_classifier_r.model_template.fromchunk

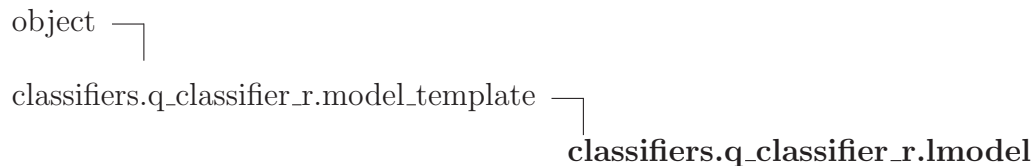
Inherited from object

__delattr__(), __format__(), __getattr__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(),
__repr__(), __setattr__(), __sizeof__(), __subclasshook__()

11.7.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

11.8 Class lmodel



11.8.1 Methods

`__init__(self, direction, offset, reference_pt)`

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `object.__init__` extit(inherited documentation)

`logp(self, datum)`

Overrides: `classifiers.q_classifier_r.model_template.logp`

`__str__(self)`

`str(x)`

Overrides: `object.__str__` extit(inherited documentation)

`tochunk(self, chunkwriter)`

Overrides: `classifiers.q_classifier_r.model_template.tochunk`

`fromchunk(chunk)`

This the inverse of `lmodel.tochunk()`, except the group start is already read.

Overrides: `classifiers.q_classifier_r.model_template.fromchunk`

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__subclasshook__()`

11.8.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

11.9 Class `classifier_desc`



Known Subclasses: `classifiers.l_classifier_guts.l_classifier_desc`

This is a thing that describes and generates `classifiers`.

11.9.1 Methods

`__init__(self, list_of_classes, fvdim, evaluator=None, ftrim=None)`

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `object.__init__` extit(inherited documentation)

`np(self)`

Return Value

The number of parameters required to define one class.

(*type=int*)

`npt(self)`

Return Value

The number of parameters required to define the classifier.

(*type=int*)

`unpack(self, prmvec, trainingset_name=None, uid=None)`

Produce a classifier from a parameter vector.

Parameters

`prmvec`: a vector of parameters that describe a classifier model.

(*type=numpy.ndarray*)

Return Value

the classifier.

(*type=the corresponding subclass of `classifier`.*)

`start(self, data)`

Starting position for Markov Chain Monte Carlo.

`__str__(self)`

`str(x)`

Overrides: `object.__str__` extit(inherited documentation)

<pre>__repr__(self) str(x) Overrides: object.__repr__ extit(inherited documentation)</pre>
--

<pre>typename(self)</pre> <hr/> <p>Return Value a string that names the subclass - what kind of <code>classifier_desc</code> is it? @rtype str</p>
--

Inherited from object

```
__delattr__(), __format__(), __getattr__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(),
__setattr__(), __sizeof__(), __subclasshook__()
```

11.9.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

11.10 Class classifier

```
object ┌
      │
      └─ classifiers.q_classifier_r.classifier
```

Known Subclasses: `classifiers.l_classifier_guts.l_classifier`

This is the base class for all kinds of classifiers.

11.10.1 Methods

__init__(*self*, *typename*, *models*, *info*=None, *trainingset_name*=None, *wid*=None, *cdesc*=None)

x.**__init__**(...) initializes *x*; see *x*.**__class__**.**__doc__** for signature

Parameters

models: a dictionary containing a probabilistic model for each class
(*type*=dict(*str*: subclass of *model_template*))

info: not used in the internal operation of the classifier, but it is stuff that is important to write out.
(*type*=dict(*str*: whatever))

Overrides: object.**__init__**

add_model(*self*, *classname*, *model*)

Add a class to an existing classifier.

list_classes(*self*)

add_info(*self*, *k*, *v*)

P(*self*, *datum*, *whichclass*=None)

Determine the probability of being in each class. If *whichclass*=None, then it returns a list of tuples [(classname,P), ...] for all classes. Otherwise, it returns the probability of the specified class.

logP(*self*, *datum*, *whichclass*=None)

logPw(*self*, *datum*, *whichclass*, *constrain*)

Determine the probability of *datum* being in *whichclass*.

logPv(*self*, *datum*)

Determine the probability of being in each class. If *whichclass*=None, then it returns a list of tuples [(classname,P), ...] for all classes.

bestc(*self*, *datum*)

Determine the best class for a datum.

```
__str__(self)
str(x)
Overrides: object.__str__ exitit(inherited documentation)
```

```
__repr__(self)
str(x)
Overrides: object.__repr__ exitit(inherited documentation)
```

```
list_wrong_classifications(self, classdata)
```

```
writer(self, dcw)
-----
Write this classifier out (usually to a data file).
Parameters
  dcw: (type=chunkio.chunk_w)
```

Inherited from `object`

```
__delattr__(), __format__(), __getattr__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(),
__setattr__(), __sizeof__(), __subclasshook__()
```

11.10.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

11.10.3 Class Variables

Name	Description
HUGE	Value: 1e+30

11.11 Class `evaluate_match_w_rare`

```
object ┌
      │
      └─ classifiers.q_classifier_r.evaluate_match_w_rare
```

This is called in the same way as `evaluate_match` or `evaluate_Bayes`. It pretends to be a

function, except that you can weight the values of different classes when you construct the class.

11.11.1 Methods

`__init__(self)`

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)

`__call__(self, clsfr, data)`

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

11.11.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

11.11.3 Class Variables

Name	Description
<code>__name__</code>	Value: <code>'evaluate_match_w_rare'</code>

12 Module `classifiers.qd_classifier_guts`

A classifier that assumes that $\log(P)$ is a quadratic form.

12.1 Functions

```
go_auto(fd, coverage=3, ftest=0.25, n_per_dim=10, ftrim=None,
evnm=None, verbose=True, modify_class=None)
```

```
test_build3(classifier_choice)
```

```
test_build1q(classifier_choice)
```

```
test_build1tq(classifier_choice)
```

```
test()
```

12.2 Variables

Name	Description
COVERAGE	Value: 3
FTEST	Value: 0.25
N_PER_DIM	Value: 10
__package__	Value: 'classifiers'

12.3 Class `qd_classifier_desc`

object 

`g_classifiers.q_classifier.r.classifier_desc`  `classifiers.qd_classifier_guts.qd_classifier_desc`

12.3.1 Methods

`__init__(self, data=None, evaluator=None, models=None, ftrim=None)`

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)

`np(self)`

Return Value

The number of parameters required to define one class.

(*type=int*)

Overrides: `g_classifiers.q_classifier_r.classifier_desc.np` `exitit`(inherited documentation)

`npt(self)`

The total number of parameters required to define all classes.

Return Value

The number of parameters required to define the classifier.

(*type=int*)

Overrides: `g_classifiers.q_classifier_r.classifier_desc.npt`

`unpack(self, prmvec, trainingset_name=None, uid=None)`

Produce a classifier from a parameter vector.

Parameters

`prmvec`: a vector of parameters that describe a classifier model.

Return Value

the classifier.

(*type=the corresponding subclass of `classifier`.*)

Overrides: `g_classifiers.q_classifier_r.classifier_desc.unpack`

`start(self, data)`

Starting position for Markov Chain Monte Carlo.

Overrides: `g_classifiers.q_classifier_r.classifier_desc.start`

```
typename(self)
```

Return Value

a string that names the subclass - what kind of `classifier_desc` is it? @rtype str

Overrides: `g_classifiers.q_classifier_r.classifier_desc.typename` extit(inherited documentation)

Inherited from `g_classifiers.q_classifier_r.classifier_desc`

```
--repr--(), --str--()
```

Inherited from `object`

```
--delattr--(), --format--(), --getattr--(), --hash--(), --new--(), --reduce--(), --reduce_ex--(),
--setattr--(), --sizeof--(), --subclasshook--()
```

12.3.2 Properties

Name	Description
<i>Inherited from <code>object</code></i>	
<code>--class--</code>	

12.4 Class `qd_classifier`

```
object └─
```

```
g_classifiers.q_classifier_r.classifier └─ classifiers.qd_classifier_guts.qd_classifier
```

12.4.1 Methods

```
__init__(self, cdesc=None, models=None, trainingset_name=None, uid=None)
```

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Parameters

`models`: a dictionary containing a probabilistic model for each class

`info`: not used in the internal operation of the classifier, but it is stuff that is important to write out.

Overrides: `object.__init__` extit(inherited documentation)

Inherited from `g_classifiers.q_classifier_r.classifier`

`P()`, `__repr__()`, `__str__()`, `add_info()`, `add_model()`, `bestc()`, `list_classes()`, `list_wrong_classifications()`, `logP()`, `logPv()`, `logPw()`, `writer()`

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__setattr__()`, `__sizeof__()`, `__subclasshook__()`

12.4.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

12.4.3 Class Variables

Name	Description
<i>Inherited from <code>g_classifiers.q_classifier_r.classifier</code></i>	
HUGE	

13 Module `classifiers.read_classified`

This module reads the outputs of `qd_classifier.py`, specifically `classified.fiat` and `classes.chunk`.

13.1 Functions

`read_classified(f)`

Read `classified.fiat`, as produced by `l_classifier` or `qd_classifier`. This unpacks the two columns that are packed into ASCII representations and turns them back into python objects.

Parameters

`f`: file or filename

(*type=file or str @rtype (dict, list(dict), list(str))*)

Return Value

Much like `fiatio.read`, it returns a tuple of header information, a list of dictionaries (each dictionary corresponding to a line) and a list of comment strings.

`model_fromchunk(chunk)`

A factory function to read in an arbitrary classifier model.

Parameters

`chunk`: (*type=chunkio.chunk*)

`read_classifier(chunk)`

Read a classifier in from a `gmisclib.chunkio.chunk`. This can yield either a quadratic or a linear classifier, depending what's available in the data file. (This is not normally called directly by the user.)

Parameters

`chunk`: loosely, a data file. More precisely, a source of tokens.

(*type=chunkio.chunk*)

Return Value

a single classifier, containing models for several classes.

(*type=l_classifier or qd_classifier*)

<p>read_classes(<i>f</i>)</p> <hr/> <p>Read <code>classes.chunk</code>, as produced by <code>l_classifier</code> or <code>qd_classifier</code>. It converts the header information to ints or floats as appropriate. It expects to read in a forest of equivalent classifiers, and it returns a list of them. This is the normal API for reading <code>classes.chunk</code>.</p> <p>Parameters</p> <p><code>f</code>: a filename or a file to read</p> <p><i>(type=</i><code>str</code><i> or </i><code>file</code><i>)</i></p> <p>Return Value</p> <p><code>tuple(dict(str:str), list(some_kind_of_classifier))</code></p>
--

<p>read_classes_header(<i>dc</i>)</p> <hr/> <p>Read in the header info. This is part of the normal API.</p> <p>Parameters</p> <p><code>dc</code>: a datachunk containing the <code>classes.chunk</code> file produced by a classifier run.</p> <p><i>(type=</i><code>chunkio.datachunk</code><i>)</i></p> <p>Return Value</p> <p>a dictionary containing the header information. See the docstring for the <code>l_classifier</code> script for typical contents. Numbers are converted to <code>float</code> or <code>int</code> as appropriate.</p> <p><i>(type=</i><code>dict(str: str, int, or float)</code><i>)</i></p>

13.2 Variables

Name	Description
<code>__package__</code>	Value: <code>'classifiers'</code>

14 Module `classifiers.sim_cen_gauss_class`

Simple centered Gaussian Classifier. It's designed for a case where there is a lot of data and where the data is a multivariate Gaussian.

14.1 Functions

logdet(*cov*)

Parameters

cov: a 2-dimensional matrix
(*type=numpy.ndarray*)

Return Value

the log of the determinant of a matrix.
(*type=float*)

typ_trace(*covlist*)

This is used with `boost_diag()`. It returns a typical value for the the trace of the covariance matrix, averaged over all matrices.

Parameters

covlist: a dictionary mapping names to covariance matrices.
(*type=dict(str:numpy.ndarray)*)

Return Value

trace of covariance
(*type=float*)

boost_diag(*covlist*, *diagboost*)

This raises all the eigenvalues a little bit, so that processing can proceed even if the covariance is singular. To first order, the fudge gets subtracted later on in the processing.

Parameters

covlist: a dictionary mapping names to covariance matrices.
(*type=dict(str:numpy.ndarray)*)

diagboost: how much to boost
(*type=float*)

go_auto(*fd*, *Mu*=False, *coverage*=3, *ftest*=0.25, *cb*=None, *cfid_file*=None)

Parameters

- cb**: a function that builds the classifier. E.g. `n_zero_cov_builder`.
(*type*=`list(QCR.classifier)`)
- Mu**: should the class means be allowed to be nonzero? Not here; Mu must be false.
(*type*=`bool`)
- cfid_file**: Pathname where the definition of the classifier should be written. None => don't write.
(*type*=`str`)

n_zero_cov_builder(*d*, *N*, *modelchoice*=None, *trainingset_name*=None, *diagboost*=1e-06)

n_zero_cov_builder_drop(*d*, *N*, *modelchoice*=None, *trainingset_name*=None, *diagboost*=1e-06)

Needs to return a list, not an iterator.

n_mu_cov_builder(*d*, *N*, *modelchoice*=None, *trainingset_name*=None, *diagboost*=1e-06)

test1()

14.2 Variables

Name	Description
COVERAGE	Value: 3
FTEST	Value: 0.25
Fudge	Value: 1e-06
PSYCO	Value: False
<code>__package__</code>	Value: 'classifiers'

15 Script `script-l_classifier`

A classifier that assumes that P is linear in position. This is known as a (linear) logistic discriminant analysis:

This is a script that can be run from the Linux command line. Usage: `l_classifier [flags] < input_data >log_file` This script also produces two files: `classes.chunk` and `classified.fiat`.

Flags:

- `-test` Run some tests.
- `-D` Print extra debug information. Repeated `-D` flags increase verbosity.
- `-quiet` Print less.
- `-c 0.NN` Ignore the specified fraction ($0 \leq 0.NN < 1$) of the worst classifications. See `q_classifier_r.evaluate_Bayes` for details. When building the classifiers, if $0.NN > 0$, this essentially says that "nothing is extremely improbable, because there's a $0.NN$ chance that it is just a mistake." This makes the classifier boundaries less sensitive to points on the outskirts of regions.
- `-ftest 0.NN` Use a fraction $0 < 0.NN < 1$ of the data for the test set; the remainder is used for training the classifiers.
- `-coverage N.N` This script generates a group of classifiers for a particular test-set/training-set pair, but then it samples a new test set and repeats until an average datum appears in a test set $N.N$ times.
- `-nperdim N.N` This controls how many classifiers are generated per test-set/training-set pair. The number is $N.N$ times the number of dimensions in the feature vector.
- `-i filename` Take input from the specified file instead of the standard input.

The input data is a multicolumn ASCII file with one line per measurement to be classified. Columns are separated by whitespace and are:

- 1: The correct class (i.e. the 'gold standard'). Obviously, the classification problem gets more difficult as the number of distinct classes gets larger.
- 2 - $N+1$: The various components of the feature vector to be used for classification. All lines must have the same number of components.
- *: An optional hash mark followed by an arbitrary identifier for that measurement. (If no identifier is supplied, it will be called "Line:NN" based on the line number in this input file.) Identifiers don't affect the computation, but they do let you connect values in the output files to feature vectors in the input file.

The standard output contains miscellaneous progress information and lines (that are prefixed with "WRONG") that list incorrect classifications. However, comprehensive classification information can be found in `classes.chunk`. This provides a list of all the classifiers that were

generated, and contains enough information to reconstruct the classifiers so that they could be applied to another set of data. (`classes.chunk` is in a format readable by `chunkio.py`.) It is recommended that it be read in by `read_classified.read_classes_header` (if you just want the top few lines of header information), or `read_classified.read_classes` for the full description.

The header contains information on classifier performance. It contains attribute/value pairs as follows:

- **Pcorrect** Average fraction of correct classification
- **PSigma** The standard deviation (among the classifiers) of fraction of correct classification.
- **total 3660** Total number of classifications (i.e. number of data times number of classifiers).
- **nok 8 K 0.994231955051 KSigma**
- **Chance** The overall probability of accidentally making a correct classification. (This is an average over all classifiers.)
- **ChSigma** The standard deviation (among the classifiers) of **Chance**.
- **Perfection** How well a perfect classifier could perform on that test set (normally 1.0).
- **PerfectionSigma** The standard deviation of **Perfection** across all classifiers (normally 0).
- **N_per_dim, Ftest, Coverage** Parameters set by command line flags
- **classifier_type** "linear_discriminant_classifier" for this program.

Notes:

- A useful general reference is: `@inbook{webb:spr:logistic, author = {Andrew Webb}, title = {Statistical Pattern Recognition}, pages = {124–132}, year = {1999}, publisher = {Arnold}, address = {London, New York}, note = {ISBN 0 340 74164 3} }`
- This code was described in an appendix to "Dimensions of durational variation in speech", by Anastassia Loukina, Greg Kochanski, Burton Rosner, Chilin Shih, and Elinor Keane, submitted 2010 to J. Acoustical Society of America.

15.1 Variables

Name	Description
<code>__package__</code>	Value: None

16 Script *script-qd_classifier*

A classifier that assumes that $\log(P)$ is a quadratic form.

16.1 Variables

Name	Description
--package--	Value: None

17 Script `script-qdg_classifier`

A linear or quadratic classifier, but the test and training sets are defined by the `-group` flag. The idea is that you can define groups and a group is treated as a unit when the data is split into the test and training set.

Why would you want to do this? For instance, if you are building classifiers to separate languages, and you have several subjects per language, it is possible that your subject is learning peculiarities of the individual subjects, rather than properties of the language.

Without grouping, half of subject A's data might be in the training set and half in the test set. The classifier may learn A's properties and then use that knowledge in the test. Thus, without grouping, the classifier can succeed without generalizing from subject to subject.

With grouping, the classifier needs to learn the language from (e.g.) subjects A, B, C and then extrapolate that knowledge onto other subjects from the same language (e.g. D, E, F). Thus, the classifier is forced to learn general properties of the language, not specific properties of the individual.

To use this, give the `-group PATTERN NUM` switch. `PATTERN` is a **regular expression**, possibly including parenthesized regions (`re.match.group`), and `NUM` is an integer that selects which region is used as the group name. `NUM=0` means the entire regular expression, `NUM=1` means the first parenthesized region, etc.

See the usage notes and flags for `l_classifier`.

Notes:

- Classifying 2300 entries into 5 groups, based on a 3-dimensional feature vector takes about 40 minutes (in 2010, on a single processor).
- This code was described in an appendix to "Dimensions of durational variation in speech", by Anastassia Loukina, Greg Kochanski, Burton Rosner, Chilin Shih, and Elinor Keane, submitted 2010 to J. Acoustical Society of America.

17.1 Functions

```
go_group_q(fd, n_per_dim=10, ftrim=None, grouper=None, coverage=3,
ftest=0.25, verbose=True, modify_class=None)
```

```
go_group_l(fd, n_per_dim=10, ftrim=None, grouper=None, coverage=3,
ftest=0.25, verbose=True, modify_class=None)
```

17.2 Variables

Name	Description
__package__	Value: None

Index

- classifiers (*package*), 4
 - classifiers.data_splitter (*module*), 5–7
 - classifiers.data_splitter.bluedata (*class*), 5–6
 - classifiers.data_splitter.bluedata_groups (*class*), 6–7
 - classifiers.data_splitter.Hash (*function*), 5
 - classifiers.g_class_centers (*module*), 8
 - classifiers.g_class_centers.format (*function*), 8
 - classifiers.g_class_centers.process (*function*), 8
 - classifiers.g_kmeans (*module*), 9
 - classifiers.g_kmeans.cluster_descriptor (*class*), 9
 - classifiers.g_kmeans.euclid (*function*), 9
 - classifiers.g_kmeans.find_center (*function*), 9
 - classifiers.g_kmeans.find_which_cluster (*function*), 9
 - classifiers.g_kmeans.kmeans (*function*), 9
 - classifiers.g_kmeans.read_data (*function*), 9
 - classifiers.l_class_show (*module*), 10
 - classifiers.l_class_show.doit (*function*), 10
 - classifiers.l_class_show.ssv (*function*), 10
 - classifiers.l_classifier_guts (*module*), 11–16
 - classifiers.l_classifier_guts.calibrate_var (*function*), 12
 - classifiers.l_classifier_guts.go_auto (*function*), 11
 - classifiers.l_classifier_guts.l_classifier (*class*), 14–16
 - classifiers.l_classifier_guts.l_classifier_desc (*class*), 12–14
 - classifiers.l_classifier_guts.ref_from_data (*function*), 11
 - classifiers.l_classifier_guts.ref_from_models (*function*), 11
 - classifiers.l_classifier_guts.test (*function*), 12
 - classifiers.l_classifier_guts.test_2_bias (*function*), 11
 - classifiers.l_classifier_guts.test_2_scale (*function*), 11
 - classifiers.l_classifier_guts.test_4_2 (*function*), 11
 - classifiers.l_classifier_guts.test_build1 (*function*), 11
 - classifiers.l_classifier_guts.test_build1n (*function*), 11
 - classifiers.l_classifier_guts.test_build1t (*function*), 11
 - classifiers.l_classifier_guts.test_build2 (*function*), 11
 - classifiers.l_classifier_guts.test_build32 (*function*), 11
 - classifiers.l_classifier_guts.test_var (*function*), 11
- classifiers.lx_classifier (*module*), 17
 - classifiers.lx_classifier.go_cross (*function*), 17
- classifiers.multivariate (*module*), 18
 - classifiers.multivariate.meanvar (*function*), 18
 - classifiers.multivariate.test_diag_quadratic (*function*), 18
 - classifiers.multivariate.test_multimu (*function*), 18
 - classifiers.multivariate.test_multimu_diag (*function*), 18
 - classifiers.multivariate.test_quadratic (*function*), 18
- classifiers.q_classiboost (*module*), 19
 - classifiers.q_classiboost.index (*function*), 19
 - classifiers.q_classiboost.vote (*function*), 19
- classifiers.q_classifier_mm (*module*), 20
 - classifiers.q_classifier_mm.build_classifiers (*function*), 20
 - classifiers.q_classifier_mm.count_the_classes

- (function)*, 20
- classifiers.q_classifier_r (*module*), 21–37
 - classifiers.q_classifier_r.classifier (*class*), 34–36
 - classifiers.q_classifier_r.classifier_desc (*class*), 32–34
 - classifiers.q_classifier_r.compute_cross_class (*function*), 22
 - classifiers.q_classifier_r.compute_group_class (*function*), 22
 - classifiers.q_classifier_r.compute_self_class (*function*), 22
 - classifiers.q_classifier_r.count_classes (*function*), 24
 - classifiers.q_classifier_r.datum_c (*class*), 26–27
 - classifiers.q_classifier_r.datum_tr (*class*), 27–28
 - classifiers.q_classifier_r.default_modify_class (*function*), 25
 - classifiers.q_classifier_r.default_writer (*function*), 24
 - classifiers.q_classifier_r.evaluate_Bayes (*function*), 24
 - classifiers.q_classifier_r.evaluate_match (*function*), 23
 - classifiers.q_classifier_r.evaluate_match_w_rare (*class*), 36–37
 - classifiers.q_classifier_r.evaluator_from_name (*function*), 25
 - classifiers.q_classifier_r.forest_build (*function*), 23
 - classifiers.q_classifier_r.get_dim (*function*), 22
 - classifiers.q_classifier_r.grouper_c (*class*), 28–29
 - classifiers.q_classifier_r.Hash (*function*), 21
 - classifiers.q_classifier_r.Hash1 (*function*), 21
 - classifiers.q_classifier_r.list_classes (*function*), 25
 - classifiers.q_classifier_r.list_groups (*function*), 22
 - classifiers.q_classifier_r.lmodel (*class*), 31–32
 - classifiers.q_classifier_r.lzmodel (*function*), 23
 - classifiers.q_classifier_r.max_correct (*function*), 21
 - classifiers.q_classifier_r.model_template (*class*), 29–30
 - classifiers.q_classifier_r.name_of_evaluator (*function*), 25
 - classifiers.q_classifier_r.prior (*function*), 21
 - classifiers.q_classifier_r.qmodel (*class*), 30–31
 - classifiers.q_classifier_r.qzmodel (*function*), 23
 - classifiers.q_classifier_r.read_data (*function*), 21
- classifiers.qd_classifier_guts (*module*), 38–41
 - classifiers.qd_classifier_guts.go_auto (*function*), 38
 - classifiers.qd_classifier_guts.qd_classifier (*class*), 40–41
 - classifiers.qd_classifier_guts.qd_classifier_desc (*class*), 38–40
 - classifiers.qd_classifier_guts.test (*function*), 38
 - classifiers.qd_classifier_guts.test_build1q (*function*), 38
 - classifiers.qd_classifier_guts.test_build1tq (*function*), 38
 - classifiers.qd_classifier_guts.test_build3 (*function*), 38
- classifiers.read_classified (*module*), 42–43
 - classifiers.read_classified.model_fromchunk (*function*), 42
 - classifiers.read_classified.read_classes (*function*), 42
 - classifiers.read_classified.read_classes_header (*function*), 43
 - classifiers.read_classified.read_classified (*function*), 42
 - classifiers.read_classified.read_classifier (*function*), 42

- tion*), 42
- classifiers.sim_cen_gauss_class (*module*), 44–45
- classifiers.sim_cen_gauss_class.boost_diag (*function*), 44
- classifiers.sim_cen_gauss_class.go_auto (*function*), 44
- classifiers.sim_cen_gauss_class.logdet (*function*), 44
- classifiers.sim_cen_gauss_class.n_mu_cov_builder (*function*), 45
- classifiers.sim_cen_gauss_class.n_zero_cov_builder (*function*), 45
- classifiers.sim_cen_gauss_class.n_zero_cov_builder_drop (*function*), 45
- classifiers.sim_cen_gauss_class.test1 (*function*), 45
- classifiers.sim_cen_gauss_class.typ_trace (*function*), 44
- script-l_classifier (*script*), 46–47
- script-qd_classifier (*script*), 48
- script-qdg_classifier (*script*), 49–50
 - script-qdg_classifier.go_group_l (*function*), 49
 - script-qdg_classifier.go_group_q (*function*), 49