graphlab.SArray

class `graphlab.` `SArray`(*data=list(), dtype=None, ignore_cast_failure=False*)

An immutable, homogeneously typed array object backed by persistent storage.

SArray is scaled to hold data that are much larger than the machine's main memory. It fully supports missing values and random access. The data backing an SArray is located on the same machine as the GraphLab Server process. Each column in an `SFrame` is an SArray.

Parameters: **data** : list | numpy.ndarray | pandas.Series | string

The input data. If this is a list, numpy.ndarray, or pandas.Series, the data in the list is converted and stored in an SArray. Alternatively if this is a string, it is interpreted as a path (or url) to a text file. Each line of the text file is loaded as a separate row. If `data` is a directory where an SArray was previously saved, this is loaded as an SArray read directly out of that directory.

dtype : {None, int, float, str, list, array.array, dict, datetime.datetime, graphlab.Image}, optional

The data type of the SArray. If not specified (None), we attempt to infer it from the input. If it is a numpy array or a Pandas series, the dtype of the array/series is used. If it is a list, the dtype is inferred from the inner list. If it is a URL or path to a text file, we default the dtype to str.

ignore_cast_failure : bool, optional

If True, ignores casting failures but warns when elements cannot be casted into the specified dtype.

Notes

- If `data` is pandas.Series, the index will be ignored.
- The datetime is based on the Boost datetime format (see http://www.boost.org/doc/libs/1_48_0/doc/html/date_time/date_time_io.html for details)

Examples

SArray can be constructed in various ways:

Construct an SArray from list.

```
>>> from graphlab import SArray
>>> sa = SArray(data=[1,2,3,4,5], dtype=int)
```

Construct an SArray from numpy.ndarray.

```
>>> sa = SArray(data=numpy.asarray([1,2,3,4,5]), dtype=int)
or:
>>> sa = SArray(numpy.asarray([1,2,3,4,5]), int)
```

Construct an SArray from pandas.Series.

```
>>> sa = SArray(data=pd.Series([1,2,3,4,5]), dtype=int)
or:
>>> sa = SArray(pd.Series([1,2,3,4,5]), int)
```

If the type is not specified, automatic inference is attempted:

```
>>> SArray(data=[1,2,3,4,5]).dtype()
int
>>> SArray(data=[1,2,3,4,5.0]).dtype()
float
```

The SArray supports standard datatypes such as: integer, float and string. It also supports three higher level datatypes: float arrays, dict and list (array of arbitrary types).

Create an SArray from a list of strings:

```
>>> sa = SArray(data=['a','b'])
```

Create an SArray from a list of float arrays;

```
>>> sa = SArray([[1,2,3], [3,4,5]])
```

Create an SArray from a list of lists:

```
>>> sa = SArray(data=[['a', 1, {'work': 3}], [2, 2.0]])
```

Create an SArray from a list of dictionaries:

```
>>> sa = SArray(data=[{'a':1, 'b': 2}, {'b':2, 'c': 1}])
```

Create an SArray from a list of datetime objects:

```
>>> sa = SArray(data=[datetime.datetime(2011, 10, 20, 9, 30, 10)])
```

Construct an SArray from local text file. (Only works for local server).

```
>>> sa = SArray('/tmp/a_to_z.txt.gz')
```

Construct an SArray from a text file downloaded from a URL.

```
>>> sa = SArray('http://s3-us-west-2.amazonaws.com/testdatasets/a_to_z.txt.gz')
```

Numeric Operators

SArrays support a large number of vectorized operations on numeric types. For instance:

```
>>> sa = SArray([1,1,1,1,1])
>>> sb = SArray([2,2,2,2,2])
>>> sc = sa + sb
>>> sc
dtype: int
Rows: 5
[3, 3, 3, 3, 3]
>>> sc + 2
dtype: int
Rows: 5
[5, 5, 5, 5, 5]
```

Operators which are supported include all numeric operators (+,-,*,/), as well as comparison operators (>, >=, <, <=), and logical operators (&, |).

For instance:

```
>>> sa = SArray([1,2,3,4,5])
>>> (sa >= 2) & (sa <= 4)
dtype: int
Rows: 5
[0, 1, 1, 1, 0]
```

The numeric operators (+,-,*,/) also work on array types:

```
>>> sa = SArray(data=[[1.0,1.0], [2.0,2.0]])
>>> sa + 1
dtype: list
Rows: 2
[array('f', [2.0, 2.0]), array('f', [3.0, 3.0])]
>>> sa + sa
dtype: list
Rows: 2
[array('f', [2.0, 2.0]), array('f', [4.0, 4.0])]
```

The addition operator (+) can also be used for string concatenation:

```
>>> sa = SArray(data=['a','b'])
>>> sa + "x"
dtype: str
Rows: 2
['ax', 'bx']
```

This can be useful for performing type interpretation of lists or dictionaries stored as strings:

```
>>> sa = SArray(data=['a,b','c,d'])
>>> ("[" + sa + "]").astype(list) # adding brackets make it look like a list
dtype: list
Rows: 2
[['a', 'b'], ['c', 'd']]
```

All comparison operations and boolean operators are supported and emit binary SArrays.

```
>>> sa = SArray([1,2,3,4,5])
>>> sa >= 2
dtype: int
Rows: 3
[0, 1, 1, 1, 1]
>>> (sa >= 2) & (sa <= 4)
dtype: int
Rows: 3
[0, 1, 1, 1, 0]
```

Element Access and Slicing SArrays can be accessed by integer keys just like a regular python list. Such operations may not be fast on large datasets so looping over an SArray should be avoided.

```
>>> sa = SArray([1,2,3,4,5])
>>> sa[0]
1
>>> sa[2]
3
>>> sa[5]
IndexError: SFrame index out of range
```

Negative indices can be used to access elements from the tail of the array

```
>>> sa[-1] # returns the last element
5
>>> sa[-2] # returns the second to last element
4
```

The SArray also supports the full range of python slicing operators:

```
>>> sa[1000:] # Returns an SArray containing rows 1000 to the end
>>> sa[:1000] # Returns an SArray containing rows 0 to row 999 inclusive
>>> sa[0:1000:2] # Returns an SArray containing rows 0 to row 1000 in steps of 2
>>> sa[-100:] # Returns an SArray containing last 100 rows
>>> sa[-100:len(sa):2] # Returns an SArray containing last 100 rows in steps of 2
```

Logical Filter

An SArray can be filtered using

```
>>> array[binary_filter]
```

where array and binary_filter are SArrays of the same length. The result is a new SArray which contains only elements of 'array' where its matching row in the binary_filter is non zero.

This permits the use of boolean operators that can be used to perform logical filtering operations. For instance:

```
>>> sa = SArray([1,2,3,4,5])
>>> sa[(sa >= 2) & (sa <= 4)]
dtype: int
Rows: 3
[2, 3, 4]
```

This can also be used more generally to provide filtering capability which is otherwise not expressible with simple boolean functions. For instance:

```
>>> sa = SArray([1,2,3,4,5])
>>> sa[sa.apply(lambda x: math.log(x) <= 1)]
dtype: int
Rows: 3
[1, 2]
```

This is equivalent to

```
>>> sa.filter(lambda x: math.log(x) <= 1)
dtype: int
Rows: 3
[1, 2]
```

Iteration

The SArray is also iterable, but not efficiently since this involves a streaming transmission of data from the server to the client. This should not be used for large data.

```
>>> sa = SArray([1,2,3,4,5])
>>> [i + 1 for i in sa]
[2, 3, 4, 5, 6]
```

This can be used to convert an SArray to a list:

```
>>> sa = SArray([1,2,3,4,5])
>>> l = list(sa)
>>> l
[1, 2, 3, 4, 5]
```

Methods

`SArray.all` ()	Return True if every element of the S
`SArray.any` ()	Return True if any element of the SA
`SArray.append` (other)	Append an SArray to the current SAr
`SArray.apply` (fn[, dtype, skip_undefined, seed])	Transform each element of the SArr
`SArray.argmax` ()	Get the index of the maximum num
`SArray.argmin` ()	Get the index of the minimum nume
`SArray.astype` (dtype[, undefined_on_failure])	Create a new SArray with all values c
`SArray.clip` ([lower, upper])	Create a new SArray with each value
`SArray.clip_lower` (threshold)	Create new SArray with all values cli
`SArray.clip_upper` (threshold)	Create new SArray with all values cli
`SArray.contains` (item)	Performs an element-wise substring
`SArray.cumulative_max` ()	Return the cumulative maximum val
`SArray.cumulative_mean` ()	Return the cumulative mean of the e
`SArray.cumulative_min` ()	Return the cumulative minimum val
`SArray.cumulative_std` ()	Return the cumulative standard devi
`SArray.cumulative_sum` ()	Return the cumulative sum of the el
`SArray.cumulative_var` ()	Return the cumulative variance of th
`SArray.date_range` (start_time, end_time, freq)	Returns a new SArray that represent
`SArray.datetime_to_str` ([str_format])	Create a new SArray with all the valu
`SArray.dict_has_all_keys` (keys)	Create a boolean SArray by checking
`SArray.dict_has_any_keys` (keys)	Create a boolean SArray by checking
`SArray.dict_keys` ()	Create an SArray that contains all th
`SArray.dict_trim_by_keys` (keys[, exclude])	Filter an SArray of dictionary type by
`SArray.dict_trim_by_values` ([lower, upper])	Filter dictionary values to a given rar

`SArray.dict_values`()	Create an SArray that contains all th
`SArray.dropna`()	Create new SArray containing only t
`SArray.dtype`()	The data type of the SArray.
`SArray.fillna`(value)	Create new SArray with all missing v
`SArray.filter`(fn[, skip_undefined, seed])	Filter this SArray by a function.
`SArray.from_avro`(filename)	Construct an SArray from an Avro fil
`SArray.from_const`(value, size)	Constructs an SArray of size with a c
`SArray.from_sequence`([start])	Create an SArray from sequence
`SArray.head`([n])	Returns an SArray which contains th
`SArray.item_length`()	Length of each element in the curre
`SArray.max`()	Get maximum numeric value in SArr
`SArray.mean`()	Mean of all the values in the SArray,
`SArray.min`()	Get minimum numeric value in SArr
`SArray.nnz`()	Number of non-zero elements in th
`SArray.num_missing`()	Number of missing elements in the
`SArray.pixel_array_to_image`(width, height, ...)	Create a new SArray with all the valu
`SArray.rolling_count`(window_start, window_end)	Count the number of non-NULL val
`SArray.rolling_max`(window_start, window_end)	Calculate a new SArray of the maxin
`SArray.rolling_mean`(window_start, window_end)	Calculate a new SArray of the mean
`SArray.rolling_min`(window_start, window_end)	Calculate a new SArray of the minim
`SArray.rolling_stdv`(window_start, window_end)	Calculate a new SArray of the stand
`SArray.rolling_sum`(window_start, window_end)	Calculate a new SArray of the sum c
`SArray.rolling_var`(window_start, window_end)	Calculate a new SArray of the varian
`SArray.sample`(fraction[, seed])	Create an SArray which contains a s
`SArray.save`(filename[, format])	Saves the SArray to file.
`SArray.show`([view])	Visualize the SArray with GraphLab (
`SArray.size`()	The size of the SArray.
`SArray.sketch_summary`([background, ...])	Summary statistics that can be calcu
`SArray.sort`([ascending])	Sort all values in this SArray.

`SArray.split_datetime`([column_name_prefix, ...])	Splits an SArray of datetime type to r
`SArray.std`([ddof])	Standard deviation of all the values i
`SArray.str_to_datetime`([str_format])	Create a new SArray with all the valu
`SArray.subslice`([start, stop, step])	This returns an SArray with each eler
`SArray.sum`()	Sum of all values in this SArray.
`SArray.tail`([n])	Get an SArray that contains the last r
`SArray.to_numpy`()	Converts this SArray to a numpy arra
`SArray.topk_index`([topk, reverse])	Create an SArray indicating which el
`SArray.unique`()	Get all unique values in the current S
`SArray.unpack`([column_name_prefix, ...])	Convert an SArray of list, array, or di
`SArray.var`([ddof])	Variance of all the values in the SArra
`SArray.vector_slice`(start[, end])	If this SArray contains vectors or lists