Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

`>>> x+2` → `7`	Sum of two variables
`>>> x-2` → `3`	Subtraction of two variables
`>>> x*2` → `10`	Multiplication of two variables
`>>> x**2` → `25`	Exponentiation of a variable
`>>> x%2` → `1`	Remainder of a variable
`>>> x/float(2)` → `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Operations — Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespace from ends

Lists — Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements — Index starts at 0

Subset

`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item

Slice

`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list

Subset Lists of Lists

`>>> my_list2[1][0]`	my_list[list][itemOfList]
`>>> my_list2[1][:2]`	

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis

scikit-learn — Machine learning

NumPy — Scientific computing

matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays — Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements — Index starts at 0

Subset

`>>> my_array[1]` → `2`	Select item at index 1

Slice

`>>> my_array[0:2]` → `array([1, 2])`	Select items at index 0 and 1

Subset 2D Numpy arrays

`>>> my_2darray[:,0]` → `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
 array([False, False, False,  True], dtype=bool)
>>> my_array * 2
 array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
 array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

Data Science Cheat Sheet
NumPy

KEY
We'll use shorthand in this cheat sheet
`arr` - A numpy Array object

IMPORTS
Import these to start
`import numpy as np`

IMPORTING/EXPORTING
- `np.loadtxt('file.txt')` - From a text file
- `np.genfromtxt('file.csv',delimiter=',')` - From a CSV file
- `np.savetxt('file.txt',arr,delimiter=' ')` - Writes to a text file
- `np.savetxt('file.csv',arr,delimiter=',')` - Writes to a CSV file

CREATING ARRAYS
- `np.array([1,2,3])` - One dimensional array
- `np.array([(1,2,3),(4,5,6)])` - Two dimensional array
- `np.zeros(3)` - 1D array of length `3` all values `0`
- `np.ones((3,4))` - 3x4 array with all values `1`
- `np.eye(5)` - 5x5 array of `0` with `1` on diagonal (Identity matrix)
- `np.linspace(0,100,6)` - Array of `6` evenly divided values from `0` to `100`
- `np.arange(0,10,3)` - Array of values from `0` to less than `10` with step `3` (eg `[0,3,6,9]`)
- `np.full((2,3),8)` - 2x3 array with all values `8`
- `np.random.rand(4,5)` - 4x5 array of random floats between `0-1`
- `np.random.rand(6,7)*100` - 6x7 array of random floats between `0-100`
- `np.random.randint(5,size=(2,3))` - 2x3 array with random ints between `0-4`

INSPECTING PROPERTIES
- `arr.size` - Returns number of elements in `arr`
- `arr.shape` - Returns dimensions of `arr` (rows, columns)
- `arr.dtype` - Returns type of elements in `arr`
- `arr.astype(dtype)` - Convert `arr` elements to type `dtype`
- `arr.tolist()` - Convert `arr` to a Python list
- `np.info(np.eye)` - View documentation for `np.eye`

COPYING/SORTING/RESHAPING
- `np.copy(arr)` - Copies `arr` to new memory
- `arr.view(dtype)` - Creates view of `arr` elements with type `dtype`
- `arr.sort()` - Sorts `arr`
- `arr.sort(axis=0)` - Sorts specific axis of `arr`
- `two_d_arr.flatten()` - Flattens 2D array `two_d_arr` to 1D
- `arr.T` - Transposes `arr` (rows become columns and vice versa)
- `arr.reshape(3,4)` - Reshapes `arr` to `3` rows, `4` columns without changing data
- `arr.resize((5,6))` - Changes `arr` shape to `5x6` and fills new values with `0`

ADDING/REMOVING ELEMENTS
- `np.append(arr,values)` - Appends `values` to end of `arr`
- `np.insert(arr,2,values)` - Inserts `values` into `arr` before index `2`
- `np.delete(arr,3,axis=0)` - Deletes row on index `3` of `arr`
- `np.delete(arr,4,axis=1)` - Deletes column on index `4` of `arr`

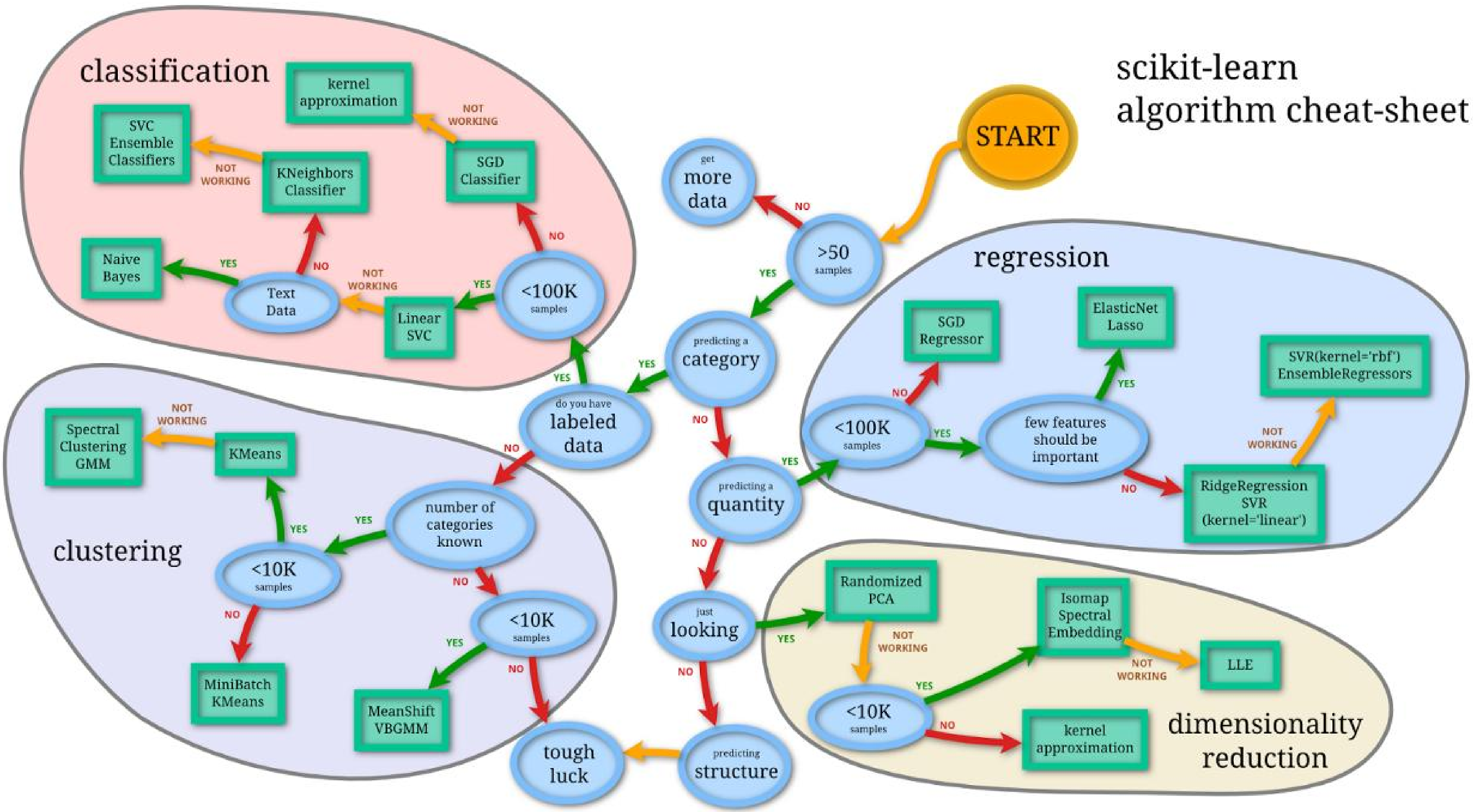
COMBINING/SPLITTING
- `np.concatenate((arr1,arr2),axis=0)` - Adds `arr2` as rows to the end of `arr1`
- `np.concatenate((arr1,arr2),axis=1)` - Adds `arr2` as columns to end of `arr1`
- `np.split(arr,3)` - Splits `arr` into `3` sub-arrays
- `np.hsplit(arr,5)` - Splits `arr` horizontally on the `5`th index

INDEXING/SLICING/SUBSETTING
- `arr[5]` - Returns the element at index `5`
- `arr[2,5]` - Returns the 2D array element on index `[2][5]`
- `arr[1]=4` - Assigns array element on index `1` the value `4`
- `arr[1,3]=10` - Assigns array element on index `[1][3]` the value `10`
- `arr[0:3]` - Returns the elements at indices `0,1,2` (On a 2D array: returns rows `0,1,2`)
- `arr[0:3,4]` - Returns the elements on rows `0,1,2` at column `4`
- `arr[:2]` - Returns the elements at indices `0,1` (On a 2D array: returns rows `0,1`)
- `arr[:,1]` - Returns the elements at index `1` on all rows
- `arr<5` - Returns an array with boolean values
- `(arr1<3) & (arr2>5)` - Returns an array with boolean values
- `~arr` - Inverts a boolean array
- `arr[arr<5]` - Returns array elements smaller than `5`

SCALAR MATH
- `np.add(arr,1)` - Add `1` to each array element
- `np.subtract(arr,2)` - Subtract `2` from each array element
- `np.multiply(arr,3)` - Multiply each array element by `3`
- `np.divide(arr,4)` - Divide each array element by `4` (returns `np.nan` for division by zero)
- `np.power(arr,5)` - Raise each array element to the `5`th power

VECTOR MATH
- `np.add(arr1,arr2)` - Elementwise add `arr2` to `arr1`
- `np.subtract(arr1,arr2)` - Elementwise subtract `arr2` from `arr1`
- `np.multiply(arr1,arr2)` - Elementwise multiply `arr1` by `arr2`
- `np.divide(arr1,arr2)` - Elementwise divide `arr1` by `arr2`
- `np.power(arr1,arr2)` - Elementwise raise `arr1` raised to the power of `arr2`
- `np.array_equal(arr1,arr2)` - Returns `True` if the arrays have the same elements and shape
- `np.sqrt(arr)` - Square root of each element in the array
- `np.sin(arr)` - Sine of each element in the array
- `np.log(arr)` - Natural log of each element in the array
- `np.abs(arr)` - Absolute value of each element in the array
- `np.ceil(arr)` - Rounds up to the nearest int
- `np.floor(arr)` - Rounds down to the nearest int
- `np.round(arr)` - Rounds to the nearest int

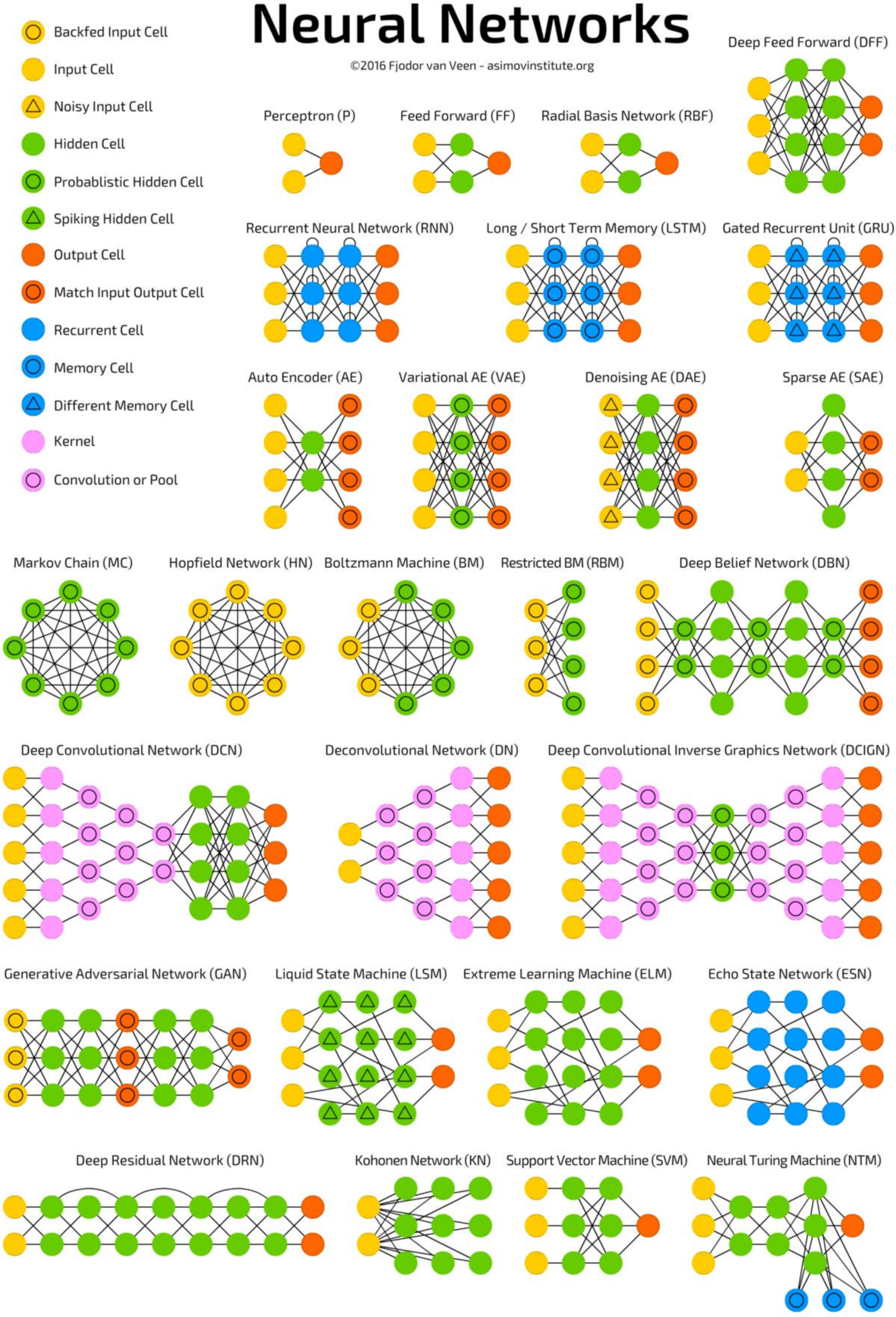
STATISTICS
- `np.mean(arr,axis=0)` - Returns mean along specific axis
- `arr.sum()` - Returns sum of `arr`
- `arr.min()` - Returns minimum value of `arr`
- `arr.max(axis=0)` - Returns maximum value of specific axis
- `np.var(arr)` - Returns the variance of array
- `np.std(arr,axis=1)` - Returns the standard deviation of specific axis
- `arr.corrcoef()` - Returns correlation coefficient of array



scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
classification
<100K samples
SGD Classifier
kernel approximation
Linear SVC
Text Data
Naive Bayes
KNeighbors Classifier
SVC Ensemble Classifiers
clustering
number of categories known
<10K samples
KMeans
Spectral Clustering GMM
MiniBatch KMeans
<10K samples
MeanShift VBGMM
regression
<100K samples
SGD Regressor
few features should be important
ElasticNet Lasso
RidgeRegression SVR (kernel='linear')
SVR(kernel='rbf') EnsembleRegressors
dimensionality reduction
Randomized PCA
<10K samples
Isomap Spectral Embedding
LLE
kernel approximation
YES
NO
NOT WORKING

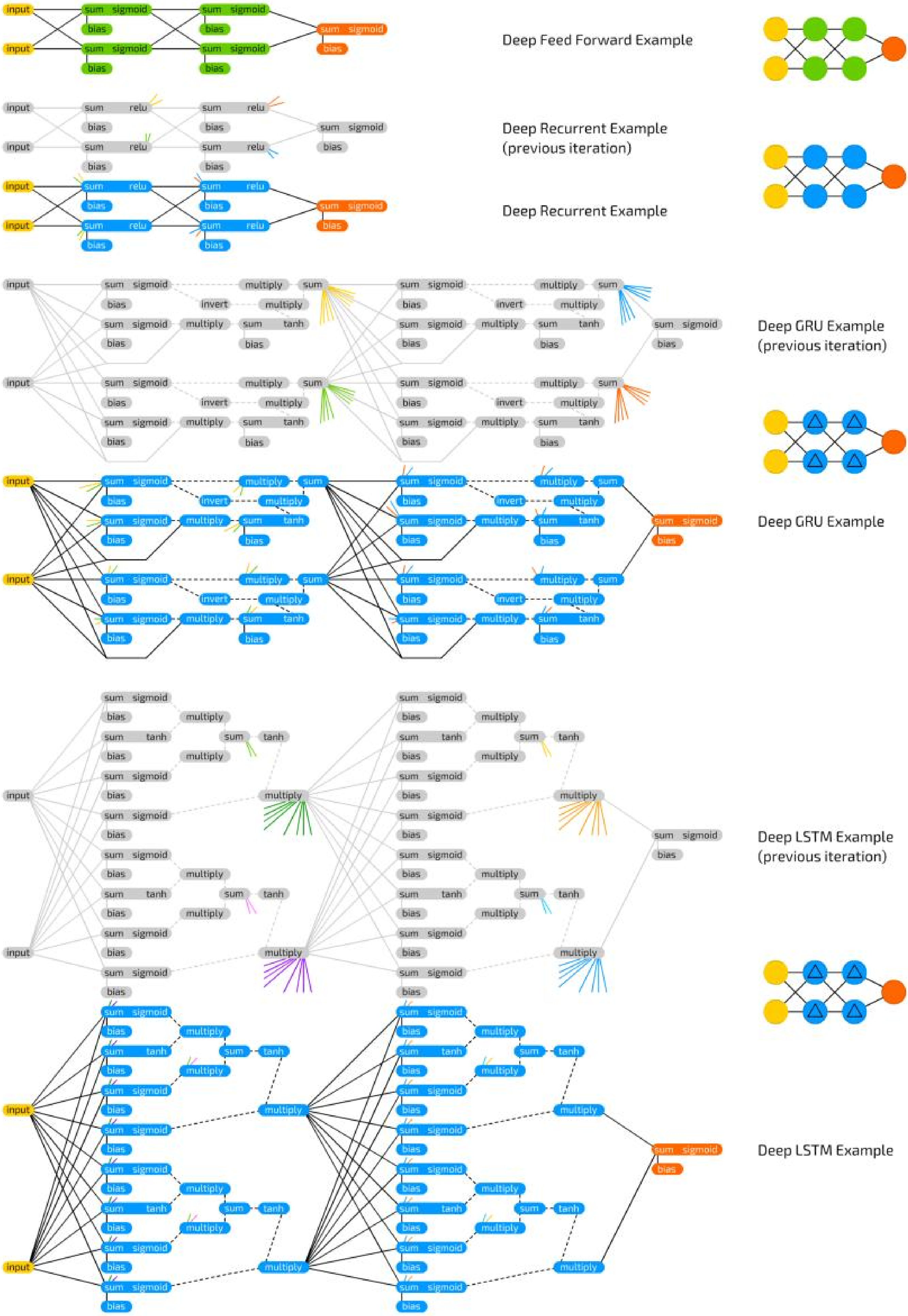
A mostly complete chart of

Neural Networks



An informative chart to build

Neural Network Graphs



Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science **Interactively** at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.cross_validation import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Fit the model to the data

Unsupervised Learning

```
>>> k_means.fit(X_train)
>>> pca_model = pca.fit_transform(X_train)
```

Fit the model to the data

Fit to data, then transform it

Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

Predict labels

Predict labels

Estimate probability of a label

Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test)
```

Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Estimator score method

Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

`>>> x+2` `7`	Sum of two variables
`>>> x-2` `3`	Subtraction of two variables
`>>> x*2` `10`	Multiplication of two variables
`>>> x**2` `25`	Exponentiation of a variable
`>>> x%2` `1`	Remainder of a variable
`>>> x/float(2)` `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]` `>>> my_list2[1][:2]`	my_list[list][itemOfList]

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespace from ends

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis
Scikit-learn — Machine learning
NumPy — Scientific computing
matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset	
`>>> my_array[1]` `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
 array([False, False, False,  True], dtype=bool)
>>> my_array * 2
 array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
 array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

Python Cheat Sheet

JUST THE BASICS

Created By: Arianne Colton and Sean Chen

General

- Python is case sensitive
- Python index starts from 0
- Python uses whitespace (tabs or spaces) to indent code instead of using braces.

HELP

Help Home Page	`help()`
Function Help	`help(str.replace)`
Module Help	`help(re)`

MODULE (AKA LIBRARY)

Python module is simply a '.py' file

List Module Contents	`dir(module1)`
Load Module	`import module1` *
Call Function from Module	`module1.func1()`

* import statement creates a new namespace and executes all the statements in the associated .py file within that namespace. If you want to load the module's content into current namespace, use '`from module1 import *`'

Scalar Types

Check data type : `type(variable)`

SIX COMMONLY USED DATA TYPES

1. **int/long*** - Large int automatically converts to long
2. **float*** - 64 bits, there is no 'double' type
3. **bool*** - True or False
4. **str*** - ASCII valued in Python 2.x and Unicode in Python 3
   - String can be in single/double/triple quotes
   - String is a sequence of characters, thus can be treated like other sequences
   - Special character can be done via \ or preface with r

     ```
     str1 = r'this\f?ff'
     ```
   - String formatting can be done in a number of ways

     ```
     template = '%.2f %s haha $%d';
     str1 = template % (4.88, 'hola', 2)
     ```

Scalar Types

* str(), bool(), int() and float() are also explicit type cast functions.

5. **NoneType(None)** - Python 'null' value (ONLY one instance of None object exists)
   - None is not a reserved keyword but rather a unique instance of 'NoneType'
   - None is common default value for optional function arguments :

     ```
     def func1(a, b, c = None)
     ```
   - Common usage of None :

     ```
     if variable is None :
     ```
6. **datetime** - built-in python 'datetime' module provides 'datetime', 'date', 'time' types.
   - 'datetime' combines information stored in 'date' and 'time'

Create datetime from String	`dt1 = datetime.strptime('20091031', '%Y%m%d')`
Get 'date' object	`dt1.date()`
Get 'time' object	`dt1.time()`
Format datetime to String	`dt1.strftime('%m/%d/%Y %H:%M')`
Change Field Value	`dt2 = dt1.replace(minute = 0, second = 30)`
Get Difference	`diff = dt1 - dt2` # diff is a 'datetime.timedelta' object

Note : Most objects in Python are mutable except for 'strings' and 'tuples'

Data Structures

Note : All non-Get function call i.e. `list1.sort()` examples below are in-place (without creating a new object) operations unless noted otherwise.

TUPLE

One dimensional, fixed-length, **immutable** sequence of Python objects of ANY type.

Data Structures

Create Tuple	`tup1 = 4, 5, 6` or `tup1 = (6,7,8)`
Create Nested Tuple	`tup1 = (4,5,6), (7,8)`
Convert Sequence or Iterator to Tuple	`tuple([1, 0, 2])`
Concatenate Tuples	`tup1 + tup2`
Unpack Tuple	`a, b, c = tup1`

Application of Tuple

Swap variables	`b, a = a, b`

LIST

One dimensional, variable length, **mutable** (i.e. contents can be modified) sequence of Python objects of ANY type.

Create List	`list1 = [1, 'a', 3]` or `list1 = list(tup1)`
Concatenate Lists*	`list1 + list2` or `list1.extend(list2)`
Append to End of List	`list1.append('b')`
Insert to Specific Position	`list1.insert(posIdx, 'b')` **
Inverse of Insert	`valueAtIdx = list1.pop(posIdx)`
Remove First Value from List	`list1.remove('a')`
Check Membership	`3 in list1 => True` ***
Sort List	`list1.sort()`
Sort with User-Supplied Function	`list1.sort(key = len)` # sort by length

* List concatenation using '+' is expensive since a new list must be created and objects copied over. Thus, `extend()` is preferable.

** Insert is computationally expensive compared with append.

*** Checking that a list contains a value is lot slower than dicts and sets as Python makes a linear scan where others (based on hash tables) in constant time.

Built-in 'bisect module‡

- Implements binary search and insertion into a sorted list
- 'bisect.bisect' finds the location, where 'bisect.insort' actually inserts into that location.

‡ WARNING : bisect module functions do not check whether the list is sorted, doing so would be computationally expensive. Thus, using them in an unsorted list will succeed without error but may lead to incorrect results.

SLICING FOR SEQUENCE TYPES†

† Sequence types include 'str', 'array', 'tuple', 'list', etc.

Notation	`list1[start:stop]`
	`list1[start:stop:step]` (If step is used) §

Note :

- 'start' index is included, but 'stop' index is NOT.
- start/stop can be omitted in which they default to the start/end.

§ Application of 'step' :

Take every other element	`list1[::2]`
Reverse a string	`str1[::-1]`

DICT (HASH MAP)

Create Dict	`dict1 ={'key1' :'value1',2 :[3, 2]}`
Create Dict from Sequence	`dict(zip(keyList, valueList))`
Get/Set/Insert Element	`dict1['key1']`* `dict1['key1'] = 'newValue'`
Get with Default Value	`dict1.get('key1', defaultValue)` **
Check if Key Exists	`'key1' in dict1`
Delete Element	`del dict1['key1']`
Get Key List	`dict1.keys()` ***
Get Value List	`dict1.values()` ***
Update Values	`dict1.update(dict2)` # dict1 values are replaced by dict2

* 'KeyError' exception if the key does not exist.

** 'get()' by default (aka no 'defaultValue') will return 'None' if the key does not exist.

*** Returns the lists of keys and values in the same order. However, the order is not any particular order, aka it is most likely not sorted.

Valid dict key types

- Keys have to be immutable like scalar types (int, float, string) or tuples (all the objects in the tuple need to be immutable too)
- The technical term here is 'hashability', check whether an object is hashable with the `hash('this is string'), hash([1, 2])` - this would fail.

SET

- A set is an **unordered** collection of UNIQUE elements.
- You can think of them like dicts but keys only.

Create Set	`set([3, 6, 3])` or `{3, 6, 3}`
Test Subset	`set1.issubset(set2)`
Test Superset	`set1.issuperset(set2)`
Test sets have same content	`set1 == set2`

- **Set operations :**

Union(aka 'or')	`set1 | set2`
Intersection (aka 'and')	`set1 & set2`
Difference	`set1 - set2`
Symmetric Difference (aka 'xor')	`set1 ^ set2`

Numpy Cheat Sheet

Python Package

Numpy (Numerical Python)

What is NumPy?

Foundation package for scientific computing in Python

Why NumPy?

- Numpy **'ndarray'** is a much more efficient way of storing and manipulating **"numerical data"** than the built-in Python data structures.
- Libraries written in lower-level languages, such as C, can operate on data stored in Numpy **'ndarray'** without copying any data.

N-DIMENSIONAL ARRAY (NDARRAY)

What is NdArray?

Fast and space-efficient multidimensional array (container for homogeneous data) providing vectorized arithmetic operations

Create NdArray	`np.array(seq1)` # seq1 - is any sequence like object, i.e. [1, 2, 3]

Create Special NdArray	`1, np.zeros(10)`
	# one dimensional ndarray with 10 elements of value 0
	`2, np.ones(2, 3)`
	# two dimensional ndarray with 6 elements of value 1
	`3, np.empty(3, 4, 5) *`
	# three dimensional ndarray of uninitialized values
	`4, np.eye(N) or np.identity(N)`
	# creates N by N identity matrix

NdArray version of Python's `range`	`np.arange(1, 10)`
Get # of Dimension	`ndarray1.ndim`
Get Dimension Size	`dim1size, dim2size, .. = ndarray1.shape`
Get Data Type **	`ndarray1.dtype`
Explicit Casting	`ndarray2 = ndarray1.astype(np.int32) ***`

- * Cannot assume empty() will return all zeros. It could be garbage values.

- ** Default data type is **'np.float64'**. This is equivalent to Python's float type which is 8 bytes (64 bits); thus the name 'float64'.
- *** If casting were to fail for some reason, **'TypeError'** will be raised.

SLICING (INDEXING/SUBSETTING)

- Slicing (i.e. `ndarray1[2:6]`) is a **'view'** on the original array. **Data is NOT copied.** Any modifications (i.e. `ndarray1[2:6] = 8`) to the 'view' will be reflected in the original array.
- Instead of a 'view', explicit copy of slicing via :

```
ndarray1[2:6].copy()
```

- Multidimensional array indexing notation :

```
ndarray1[0][2] or ndarray1[0, 2]
```

*** Boolean indexing :**

```
ndarray1[(names == 'Bob') | (names == 'Will'), 2:]
# '2:' means select from 3rd column on
```

- Selecting data by boolean indexing **ALWAYS** creates a copy of the data.
- The 'and' and 'or' keywords do NOT work with boolean arrays. Use & and |.

*** Fancy indexing** (aka 'indexing using integer arrays')

Select a subset of rows in a particular order :

```
ndarray1[ [3, 8, 4] ]
ndarray1[ [-1, 6] ]
# negative indices select rows from the end
```

- Fancy indexing **ALWAYS** creates a copy of the data.

Numpy (Numerical Python)

Setting data with assignment :

```
ndarray1[ndarray1 < 0] = 0 *
```

- * If ndarray1 is two-dimensions, ndarray1 < 0 creates a two-dimensional boolean array.

COMMON OPERATIONS

1. Transposing

- A special form of reshaping which returns a **'view'** on the underlying data without copying anything.

```
ndarray1.transpose()     or
ndarray1.T               or
ndarray1.swapaxes(0, 1)
```

2. Vectorized wrappers (for functions that take scalar values)

- `math.sqrt()` works on only a `scalar`

```
np.sqrt(seq1) # any sequence (list, ndarray, etc) to return a ndarray
```

3. Vectorized expressions

- `np.where(cond, x, y)` is a vectorized version of the expression 'x if condition else y'

```
np.where([True, False], [1, 2], [2, 3]) => ndarray (1, 3)
```

- Common Usages :

```
np.where(matrixArray > 0, 1, -1)
=> a new array (same shape) of 1 or -1 values
np.where(cond, 1, 0).argmax() *
=> Find the first True element
```

- * `argmax()` can be used to find the index of the maximum element. Example usage is find the first element that has a "price > number" in an array of price data.

4. Aggregations/Reductions Methods (i.e. mean, sum, std)

Compute mean	`ndarray1.mean()` or `np.mean(ndarray1)`
Compute statistics over axis *	`ndarray1.mean(axis = 1)` `ndarray1.sum(axis = 0)`

- * axis = 0 means column axis, 1 is row axis.

5. Boolean arrays methods

Count # of 'Trues' in boolean array	`(ndarray1 > 0).sum()`
If at least one value is 'True'	`ndarray1.any()`
If all values are 'True'	`ndarray1.all()`

Note: These methods also work with non-boolean arrays, where non-zero elements evaluate to True.

6. Sorting

Inplace sorting	`ndarray1.sort()`
Return a sorted copy instead of inplace	`sorted1 = np.sort(ndarray1)`

7. Set methods

Return sorted unique values	`np.unique(ndarray1)`
Test membership of ndarray1 values in [2, 3, 6]	`resultBooleanArray = np.in1d(ndarray1, [2, 3, 6])`

- Other set methods : `intersect1d()`, `union1d()`, `setdiff1d()`, `setxor1d()`

8. Random number generation (np.random)

- Supplements the built-in Python random * with functions for efficiently generating whole arrays of sample values from many kinds of probability distributions.

```
samples = np.random.normal(size =(3, 3))
```

- * Python built-in random ONLY samples one value at a time.

Created by Arianne Colton and Sean Chen
www.datasciencefree.com
Based on content from 'Python for Data Analysis' by Wes McKinney
Updated: August 18, 2016

MACHINE LEARNING IN EMOJI

SUPERVISED · UNSUPERVISED · REINFORCEMENT

SUPERVISED human builds model based on input / output

UNSUPERVISED human input, machine output
human utilizes if satisfactory

REINFORCEMENT human input, machine output
human reward/punish, cycle continues

BASIC REGRESSION

LINEAR linear_model.LinearRegression()
Lots of numerical data

LOGISTIC linear_model.LogisticRegression()
Target variable is categorical
or

CLASSIFICATION

NEURAL NET neural_network.MLPClassifier()
Complex relationships. Prone to overfitting
Basically magic.

K-NN neighbors.KNeighborsClassifier()
Group membership based on proximity

DECISION TREE tree.DecisionTreeClassifier()
If/then/else. Non-contiguous data
Can also be regression

RANDOM FOREST ensemble.RandomForestClassifier()
Find best split randomly
Can also be regression

SVM svm.SVC() svm.LinearSVC()
Maximum margin classifier. Fundamental
Data Science algorithm

NAIVE BAYES GaussianNB() MultinomialNB() BernoulliNB()
Updating knowledge step by step with new info

CLUSTER ANALYSIS

K-MEANS cluster.KMeans()
Similar datum into groups
based on centroids

ANOMALY DETECTION covariance.EllipticalEnvelope()
Finding outliers
through grouping

FEATURE REDUCTION

T-DISTRIB STOCHASTIC NEIB EMBEDDING manifold.TSNE()
Visualize high dimensional data. Convert
similarity to joint probabilities

PRINCIPLE COMPONENT ANALYSIS decomposition.PCA()
Distill feature space into components that
describe greatest variance

CANONICAL CORRELATION ANALYSIS decomposition.CCA()
Making sense of cross-correlation
matrices

LINEAR DISCRIMINANT ANALYSIS lda.LDA()
Linear combination of features that
separates classes

OTHER IMPORTANT CONCEPTS

- BIAS VARIANCE TRADEOFF
- UNDERFITTING / OVERFITTING
- INERTIA
- ACCURACY FUNCTION (TP + TN) / (P + N)
- PRECISION FUNCTION TP / (TP + FP)
- SPECIFICITY FUNCTION TN / (FP + TN)
- SENSITIVITY FUNCTION TP / (TP + FN)

@emilyinamillion made this

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Python For Data Science *Cheat Sheet*

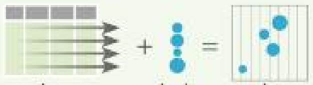
Bokeh

Learn Bokeh Interactively at www.DataCamp.com, taught by Bryan Van de Ven,core contributor

Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



data + *glyphs* = *plot*

The basic steps to creating plots with the `bokeh.plotting` interface are:

1. Prepare some data:
Python lists, NumPy arrays, Pandas DataFrames and other sequences of values
2. Create a new plot
3. Add renderers for your data, with visual customizations
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5]                              Step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example",          Step 2
               x_axis_label='x',
               y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2)       Step 3
>>> output_file("lines.html")                        Step 4
>>> show(p)                                          Step 5
```

1 Data

Also see Lists, NumPy & Pandas

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9,4,65, 'US'],
                                [32.4,4,66, 'Asia'],
                                [21.4,4,109, 'Europe']]),
                      columns=['mpg','cyl', 'hp', 'origin'],
                      index=['Toyota', 'Fiat', 'Volvo'])
```

```
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

2 Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
                x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

3 Renderers & Visual Customizations

Glyphs

Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
              fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
              color='blue', size=1)
```

Line Glyphs

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
                  pd.DataFrame([[3,4,5],[3,2,1]]),
                  color="blue")
```

Customized Glyphs

Also see Data

Selection and Non-Selection Glyphs

```
>>> p.circle('mpg', 'cyl', source=cds_df,
             selection_color='red',
             nonselection_alpha=0.1)
```

Hover Glyphs

```
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p.add_tools(hover)
```

Colormapping

```
>>> color_mapper = CategoricalColorMapper(
                        factors=['Europe', 'Asia', 'US'],
                        palette=['red', 'green', 'blue'])
>>> p.circle('mpg', 'cyl', source=cds_df,
             color=dict(field='origin',
                        transform=color_mapper),
                        legend='Origin'))
```

Rows & Columns Layout

Rows

```
>>> from bokeh.layouts import row
>>> layout = row(p1,p2, p3)
```

Columns

```
>>> from bokeh.layouts import columns
>>> layout = column(p1,p2,p3)
```

Nesting Rows & Columns

```
>>>layout = row(column(p1,p2), p3)
```

Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2],[p3]])
```

Tabbed Layout

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

Linked Plots

Also see Data

Linked Axes

```
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

Linked Brushing

```
>>> p4 = figure(plot_width = 100, tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200, tools='box_select,lasso_select')
>>> p5.circle('mpg', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

Legends

Legend Location

Inside Plot Area

```
>>> p.legend.location = 'bottom_left'
```

Outside Plot Area

```
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1])
>>> r2 = p2.line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One" , [p1, r1]),("Two" , [r2])], location=(0, -30))
>>> p.add_layout(legend, 'right')
```

Legend Orientation

```
>>> p.legend.orientation = "horizontal"
>>> p.legend.orientation = "vertical"
```

Legend Background & Border

```
>>> p.legend.border_line_color = "navy"
>>> p.legend.background_fill_color = "white"
```

4 Output

Output to HTML File

```
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

Notebook Output

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

Embedding

Standalone HTML

```
>>> from bokeh.embed import file_html
>>> html = file_html(p, CDN, "my_plot")
```

Components

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

5 Show or Save Your Plots

```
>>> show(p1)
>>> show(layout)
```

```
>>> save(p1)
>>> save(layout)
```

Statistical Charts With Bokeh

Also see Data

Bokeh's high-level `bokeh.charts` interface is ideal for quickly creating statistical charts

Bar Chart

```
>>> from bokeh.charts import Bar
>>> p = Bar(df, stacked=True, palette=['red','blue'])
```

Box Plot

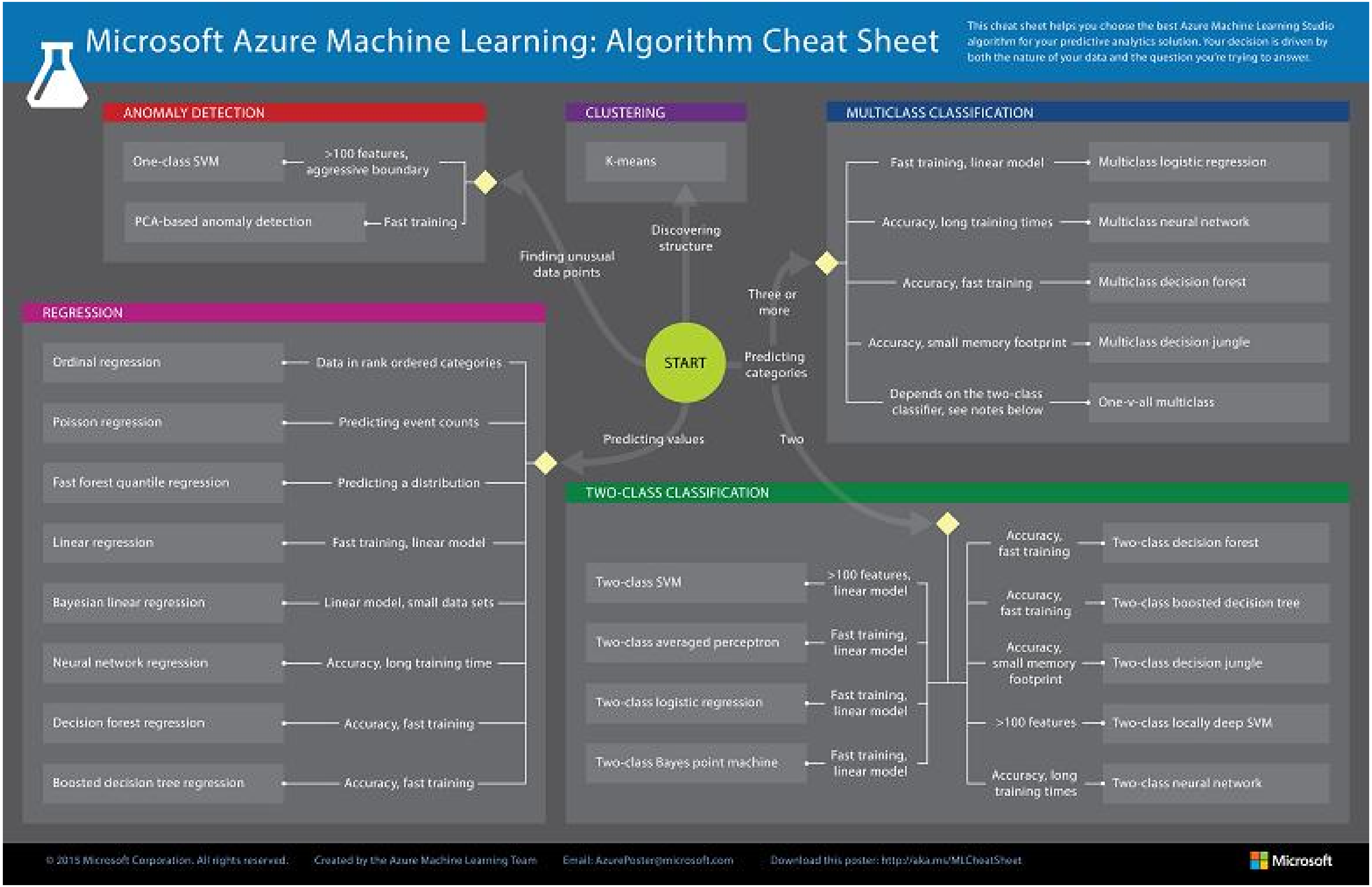
```
>>> from bokeh.charts import BoxPlot
>>> p = BoxPlot(df, values='vals', label='cyl',
                legend='bottom_right')
```

Histogram

```
>>> from bokeh.charts import Histogram
>>> p = Histogram(df, title='Histogram')
```

Scatter Plot

```
>>> from bokeh.charts import Scatter
>>> p = Scatter(df, x='mpg', y ='hp', marker='square',
                xlabel='Miles Per Gallon',
                ylabel='Horsepower')
```

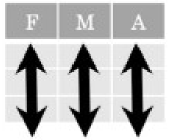


Microsoft Azure Machine Learning: Algorithm Cheat Sheet
This cheat sheet helps you choose the best Azure Machine Learning Studio algorithm for your predictive analytics solution. Your decision is driven by both the nature of your data and the question you're trying to answer.
ANOMALY DETECTION
One-class SVM
>100 features, aggressive boundary
PCA-based anomaly detection
Fast training
CLUSTERING
K-means
Discovering structure
Finding unusual data points
MULTICLASS CLASSIFICATION
Fast training, linear model
Multiclass logistic regression
Accuracy, long training times
Multiclass neural network
Accuracy, fast training
Multiclass decision forest
Accuracy, small memory footprint
Multiclass decision jungle
Depends on the two-class classifier, see notes below
One-v-all multiclass
Three or more
START
Predicting categories
Predicting values
Two
REGRESSION
Ordinal regression
Data in rank ordered categories
Poisson regression
Predicting event counts
Fast forest quantile regression
Predicting a distribution
Linear regression
Fast training, linear model
Bayesian linear regression
Linear model, small data sets
Neural network regression
Accuracy, long training time
Decision forest regression
Accuracy, fast training
Boosted decision tree regression
Accuracy, fast training
TWO-CLASS CLASSIFICATION
Two-class SVM
>100 features, linear model
Two-class averaged perceptron
Fast training, linear model
Two-class logistic regression
Fast training, linear model
Two-class Bayes point machine
Fast training, linear model
Accuracy, fast training
Two-class decision forest
Accuracy, fast training
Two-class boosted decision tree
Accuracy, small memory footprint
Two-class decision jungle
>100 features
Two-class locally deep SVM
Accuracy, long training times
Two-class neural network
© 2015 Microsoft Corporation. All rights reserved.
Created by the Azure Machine Learning Team
Email: AzurePoster@microsoft.com
Download this poster: http://aka.ms/MLCheatSheet
Microsoft

Data Wrangling
with pandas
Cheat Sheet
http://pandas.pydata.org

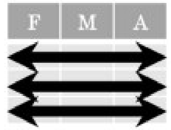
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



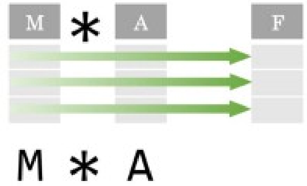
Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
      index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

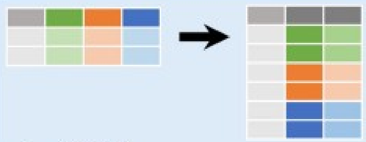
```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
           names=['n','v'])))
```
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
     )
```

Reshaping Data – Change the layout of a data set



`pd.melt(df)`
Gather columns into rows.

`df.pivot(columns='var', values='val')`
Spread rows into columns.

`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`
Order rows by values of a column (high to low).

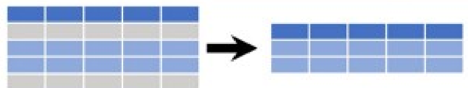
`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(['Length','Height'], axis=1)`
Drop columns from DataFrame

Subset Observations (Rows)



`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

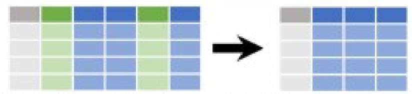
`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	`&,|,~,^,df.any(),df.all()`	Logical and, or, not, xor, any, all

Subset Variables (Columns)



`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='`*regex*`')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
''^(?!Species$).*'	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

Linear Vector Spaces:

Definition: A linear vector space, X is a set of elements (vectors) defined over a scalar field, F, that satisfies the following conditions:

1) if $x \in X$ and $y \in X$ then $x+y \in X$. 2) $x+y = y+x$ 3) $(x+y)+z = x+(y+z)$
4) There is a unique vector $0 \in X$, such that $x+0 = x$ for all $x \in X$.
5) For each vector $x \in X$ there is a unique vector in X, to be called $(-x)$, such that $x+(-x) = 0$. 6) multiplication, for all scalars $a \in F$, and all vectors $x \in X$,
7) For any $x \in X$, $1x = x$ (for scalar 1).
8) For any two scalars $a \in F$ and $b \in F$ and any $x \in X$, $a(bx) = (ab)x$.
9) $(a+b)x = ax + bx$. 10) $a(x+y) = ax + ay$.

Linear Independence: Consider n vectors $\{x_1, x_2, \ldots, x_n\}$. If there exists n scalars $a_1, a_2, \ldots, a_n$, at least one of which is nonzero, such that $a_1x_1 + a_2x_2 + \ldots + a_nx_n = 0$, then the $\{x_i\}$ are linearly dependent.

Spanning a Space:
Let X be a linear vector space and let $\{u_1, u_2, \ldots, u_n\}$ be a subset of vectors in X. This subset spans X if and only if for every vector $x \in X$ there exist scalars $x_1, x_2, \ldots, x_n$ such that $x = x_1u_1 + x_2u_2 + \ldots + x_nu_n$.

Inner Product: (x, y) for any scalar function of x and y.
1. $(x,y) = (y,x)$ 2. $(x, ay_1 + by_2) = a(x,y_1) + b(x,y_2)$
3. $(x,x) \geq 0$, where equality holds iff x is the zero vector.

Norm: Ascalar function $\|x\|$ is called a norm if it satisfies:
1. $\|x\| \geq 0$ 2. $\|x\| = 0$ if and only if $x = 0$.
3. $\|ax\| = |a|\|x\|$ 4. $\|x+y\| \leq \|x\| + \|y\|$

Angle: The angle θ bet. 2 vectors x and y is defined by $\cos\theta = \frac{(x,y)}{\|x\|\,\|y\|}$

Orthogonality: 2 vectors $x, y \in X$ are said to be orthogonal if $(x,y) = 0$.

Gram Schmidt Orthogonalization:
Assume that we have n independent vectors $y_1, y_2, \ldots, y_n$. From these vectors we will obtain n orthogonal vectors $v_1, v_2, \ldots, v_n$.

$$v_1 = y_1, \quad v_k = y_k - \sum_{i=1}^{k-1} \frac{(v_i, y_k)}{(v_i, v_i)} v_i,$$

$$where\ \frac{(v_i, y_k)}{(v_i, v_i)} v_i\ is\ the\ projection\ of\ y_k\ on\ v_i$$

Vector Expansions:

$$x = \sum_{i=1}^{n} x_i v_i = x_1v_1 + x_2v_2 + \cdots + x_nv_n,$$

$$for\ orthogonal\ vectors, x_j = \frac{(v_j, x)}{(v_j, v_j)}$$

Reciprocal Basis Vectors:

$$(r_i, v_j) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}, \quad x_j = (r_j, x)$$

To compute the reciprocal basis vectors: set $\mathbf{B} = [v_1\ v_2\ \ldots\ v_n]$,
$\mathbf{R} = [r_1\ r_2 \ldots\ r_n]$, $\mathbf{R}^T = \mathbf{B}^{-1}$ In matrix form: $\mathbf{x}^v = \mathbf{B}^{-1}\mathbf{x}^s$

Transformations:
A *transformation* consists of three parts:
domain: $X = \{x_i\}$, range: $Y = \{y_i\}$, and a rule relating each $x_i \in X$ to an element $y_i \in Y$.

Linear Transformations: transformation A is *linear* if:
1. for all $x_1, x_2 \in X$, $A(x_1+x_2) = A(x_1) + A(x_2)$
2. for all $x \in X$, $a \in R$, $A(ax) = aA(x)$

Matrix Representations:
Let $\{v_1, v_2, \ldots, v_n\}$ be a basis for vector space X, and let $\{u_1, u_2, \ldots, u_m\}$ be a basis for vector space Y. Let A be a linear transformation with domain X and range Y: $A(x) = y$
The coefficients of the matrix representation are obtained from

$$A(v_j) = \sum_{i=1}^{m} a_{ij} u_i$$

Change of Basis: $\mathbf{B}_t = [\mathbf{t}_1\ \mathbf{t}_2\ \ldots \mathbf{t}_n]$, $\mathbf{B}_w = [\mathbf{w}_1\ \mathbf{w}_2\ \ldots \mathbf{w}_n]$
$\mathbf{A}' = [\mathbf{B}_w^{-1}\mathbf{A}\mathbf{B}_t]$

Eigenvalues & Eigenvectors: $\mathbf{Az} = \lambda\mathbf{z}$, $|[\mathbf{A} - \lambda\mathbf{I}]| = 0$

Diagonalization: $\mathbf{B} = [\mathbf{z}_1\ \mathbf{z}_2\ \ldots \mathbf{z}_n]$,
where $\{\mathbf{z}, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$ are the eigenvectors of a square matrix A,
$[\mathbf{B}^{-1}\mathbf{AB}] = \mathrm{diag}([\lambda_1\ \lambda_2 \ldots \lambda_n])$

Perceptron Architecture:
$\mathbf{a} = hardlim(\mathbf{Wp} + \mathbf{b})$, $\mathbf{W} = [\,{}_1\mathbf{w}^T\ {}_2\mathbf{w}^T\ \ldots\ {}_S\mathbf{w}^T]^T$,

$$a_i = hardlim(n_i) = hardlim(\,{}_i\mathbf{w}^T\mathbf{p} + b_i)$$

Decision Boundary: ${}_i\mathbf{w}^T\mathbf{p} + b_i = 0$
The decision boundary is always orthogonal to the weight vector.
Single-layer perceptrons can only classify linearly separable vectors.

Perceptron Learning Rule

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{ep}^T, \mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e},$$
$$where\quad \mathbf{e} = \mathbf{t} - \mathbf{a}$$

Hebb's Postulate: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

Linear Associator: $\mathbf{a} = purelin(\mathbf{Wp})$

The Hebb Rule: *Supervised Form:* $w_{ij}^{new} = w_{ij}^{old} + t_{qi}p_{qj}$

$$\mathbf{W} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T + \cdots + \mathbf{t}_Q\mathbf{p}_Q^T$$

$$\mathbf{W} = [\mathbf{t}_1\ \mathbf{t}_2 \ldots \mathbf{t}_Q]\begin{bmatrix}\mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T\end{bmatrix} = \mathbf{TP}^T$$

Pseudoinverse Rule: $\mathbf{W} = \mathbf{TP}^+$
When the number, R, of rows of $\mathbf{P}$ is greater than the num ber of columns, Q, of $\mathbf{P}$ and the columns of $\mathbf{P}$ are independent, then the pseudoinverse can be computed by $\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T$

Variations of Hebbian Learning:

Filtered Learning (Ch.14): $\mathbf{W}^{new} = (1-\gamma)\mathbf{W}^{old} + \alpha\mathbf{t}_q\mathbf{p}_q^T$

Delta Rule (Ch.10): $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha(\mathbf{t}_q - \mathbf{a}_q)\mathbf{p}_q^T$

Unsupervised Hebb (Ch.13): $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha\mathbf{a}_q\mathbf{p}_q^T$

Taylor: $F(\mathbf{x}) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T|_{\mathbf{x}=\mathbf{x}^*}(\mathbf{x} - \mathbf{x}^*) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)\nabla^2 F(\mathbf{x})^T|_{\mathbf{x}=\mathbf{x}^*}(\mathbf{x} - \mathbf{x}^*) + \cdots$

Grad $\nabla F(\mathbf{x}) = \left[\frac{\partial}{\partial x_1}F(\mathbf{x})\ \ \frac{\partial}{\partial x_2}F(\mathbf{x})\ \ \ldots\ \ \frac{\partial}{\partial x_n}F(\mathbf{x})\right]^T$

Hessian: $\nabla^2 F(\mathbf{x}) =$

$$\begin{bmatrix}
\frac{\partial}{\partial x_1^2}F(\mathbf{x}) & \frac{\partial}{\partial x_1 \partial x_2}F(\mathbf{x}) \ldots & \frac{\partial}{\partial x_1 \partial x_n}F(\mathbf{x}) \\
\frac{\partial}{\partial x_2 \partial x_1}F(\mathbf{x}) & \frac{\partial}{\partial x_2^2}F(\mathbf{x}) \ldots & \frac{\partial}{\partial x_2 \partial x_n}F(\mathbf{x}) \\
\vdots & \vdots & \vdots \\
\frac{\partial}{\partial x_n \partial x_1}F(\mathbf{x}) & \frac{\partial}{\partial x_n \partial x_2}F(\mathbf{x}) \ldots & \frac{\partial}{\partial x_n^2}F(\mathbf{x})
\end{bmatrix}$$

Directional Derivatives:

1st Dir.Der.: $\frac{\mathbf{p}^T \nabla F(\mathbf{x})}{\|\mathbf{p}\|}$, **2nd Dir.Der.:** $\frac{\mathbf{p}^T \nabla^2 F(\mathbf{x})\mathbf{p}}{\|\mathbf{p}\|^2}$

Minima:
Strong Minimum: if a scalar $\delta > 0$exists, such that $F(x) < F(x + \Delta x)$ for all Δx such that $\delta > \|\Delta x\| > 0$.
Global Minimum: if $F(x) < F(x + \Delta x)$ for all $\Delta x \neq 0$
Weak Minimum: if it is not a strong minimum, and a scalar $\delta > 0$exists, such that $F(x) \leq F(x + \Delta x)$ for all Δx such that $\delta > \|\Delta x\| > 0$.

Necessary Conditions for Optimality:
1st-Order Condition: $\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^*} = 0$ (Stationary Points)
2nd-Order Condition: $\nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^*} \geq 0$ (Positive Semi-definite Hessian Matrix).

Quadratic fn.: $F(x) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\,\mathbf{x} + \mathbf{d}^T\mathbf{x} + c$

$\nabla F(x) = \mathbf{Ax} + \mathbf{d}$, $\nabla^2 F(x) = \mathbf{A}$, $\lambda_{min} \leq \frac{\mathbf{p}^T\mathbf{A}\,\mathbf{p}}{\|\mathbf{p}\|^2} \leq \lambda_{max}$