SEAS 6414 Sprint 2024 Mid Term cheat sheet.

Lecture 1

Arguments are passed by reference – changes will impact them.

Using isinstance Function: Allows checking if an object is an instance of a specific type.

Strings and tuples are immutable, others mutable

The strftime method is used to format a datetime object as a string, while strptime converts strings into datetime objects.

Lecture 2

Break – exit, continue – next loop, pass – noop

Range – up to but not including the end range(start, end, step)

Tupple – (), immutable, tup3 = tup1 + tup2 \* 🡪 multiple copies, count(n) counts number of n in tupple

Lists – [], mutable, append, insert, pop, remove, extend is faster than +

Slicing [start:stop:step], stop is not included, negative starts from the right. The use of [::-1] reverses the list.

Dictionary {“key” : “value”, “key” : “value”,..}, mutable, del, pop, update to merge, valyes can be anything, keys need to be immutable, can use hash function to check

Zip(a,b,..) merges them item by item.

Sets – set function or Curly braces, support union / |, intersection / &, set elements must be hashable, issubset, issuperset

Enumerate returns index, value

List, set, and dictionary comprehensions [expr for value in collection if condition], for a dictionary it is {key-expr: value-expr for value in collection if condition}, set uses curly braces

Lecture 3 - NumPy

Import numpy as np, data = np.array(…), all elements must be of the same type – data.shape, data.dtype, data.dim, np.zeros, np.ones, arrange is like range but returns a numpy array, eye / identity both return an identity array. Binary operators between two arrays of the same shape are done element by element, An array and scalar applies the scalar to every element. Comparing two same shape arrays creates a Boolean array. The .copy() makes a copy of the array. Array indexes are views. Arr[x , y] – x and y are applying to different dimensions. Arr[3, :2] says to take the third element in the first dimension and then take 0-2 values in the second dimension. If Names has a list of names that is the same number as the first dimension of data, can say data[name == “Bob”] to get all of Bob. Name == “Bob” creates a binary mask. This always creates a new array.

Transpose function, arr.T is the transpose of arr. Np.where(cond, Xarr, Yarr) – if true, from Xarr, else Yarr. Sum, mean, std, etc.

Lecture 4 – Pandas

Pd.series ([1,2,3,4]) – series with numeric 0 – n-1 as index. Pd.series([1,2,3,4], index=[“a”,”b”,”c”,”d”]) series with character index. Similar to a dictionary, a = pd.Series(aDict) will convert the dictionary with the key as index.

Dataframe has both a row and column index. A dictionary of equal length lists can be used to create a dataframe. Can use Columns parameter in pd.dataframe. Assigning or mentioning a column that does not exist will create the column with NaN. New columns cannot be create with the dot notation. A.newcol = 7 will not work but a[“newcol] = 7 will. Del df[“acol” will delete acol. Indexing is a view on the underlying dataframe. Use copy to create a new one. Can set name of index and columns using df.index.name = “row name” and df.columns.name = “col name”. Indexes are immutable. Can use reindex to reorder the index values and add new ones. Method can be used with reindex to fill in values – forward, backward, etc. Can use reindex on columns with the columns keyword. Drop can remove rows or columns. Use index for rows and Columns for columns. Loc and iloc are preferred. Loc is label based and iLoc is interger based. Pandas slices include the last element. Df[“a”] referes to a column named a in the dataframe df. Oh, and if you want two columns, you need two braces. So, df[[“a”,”b”]].

Df[df[“b”] > 5] gets all rows in df where column b is greater than 5. Df.loc[“Colorado”] gets the row with index label of Colorado. To get multiples, we need the double brackets. Df.loc[[“Colorado”, “Ohio”]]. Df.loc[ row, col] so, we can have df.loc[“Ohio”, “a”] to get column a for row Ohio. If sets, use the doulbel brackets. Df.loc[[“Ohio”,”New York”],[“a”,”b”]] gets columns a and b for rows Ohio and New York. Same for iLoc but you need to use integer references. Df.sort\_index() sorts rows by index. Df.sort\_index(axis=”columns”) sorts columns. Add , ascending= False to sort Descending. Use sort\_values to sort by values. .describe() provides summary statistics.

Lecture 5

Pd.concat is similar to np.concatenate. Pandas can have duplicate indexes. Verify\_integrit=True does not allow this – error. Ignore\_index=true creates new indexes. Specifying keys creates a hierarchy of indexes. Join=”inner” on concat. Merge will look for common keys and join on them. Use the on keyword to specify the merge key. Use left\_on and right\_on if they have different names. Keeps both keys so use drop to get rid of one. Use left\_index and right\_index to merge on indexes. Can use left\_on or left\_index and same for right. Join does an index based join by default. Use how to specify inner, outer, etc. Use suffixes to specify a suffix to make the names unique.

Df.groupby(“a”) – groups by a column. Filtering I done by a filter function that returns a Boolean.

Seaborn, MATPLOTLIB has outdated color and style – addressed in 20,, low level APIS requiring a lot of code, predates Pandas so does not work well with Pandas

Seaborn is built on top of MATPLOTLIB and adds integration to Pandas, plot and style defaults. It also performs statistical functions.

Lecture 6

Classification, Regression (continuous), clustering – infers that a label exists but does not know what it is, Dimensional Reduction

Supervised Learning: Classification, Regression

Unsupervised Learning: Clustering, Dimensional Reduction

Scikit-learn

Seaborn pairplot can help to visualize the data

The Scikit-Learn API is designed with the following guiding principles in mind, as outlined in the [Scikit-Learn API paper](http://arxiv.org/abs/1309.0238):

- \*Consistency\*: All objects share a common interface drawn from a limited set of methods, with consistent documentation.

- \*Inspection\*: All specified parameter values are exposed as public attributes.

- \*Limited object hierarchy\*: Only algorithms are represented by Python classes; datasets are represented

in standard formats (NumPy arrays, Pandas ``DataFrame``s, SciPy sparse matrices) and parameter

names use standard Python strings.

- \*Composition\*: Many machine learning tasks can be expressed as sequences of more fundamental algorithms,

and Scikit-Learn makes use of this wherever possible.

- \*Sensible defaults\*: When models require user-specified parameters, the library defines an appropriate default value.

In practice, these principles make Scikit-Learn very easy to use, once the basic principles are understood.

Every machine learning algorithm in Scikit-Learn is implemented via the Estimator API, which provides a consistent interface for a wide range of machine learning applications.

### Basics of the API

Most commonly, the steps in using the Scikit-Learn estimator API are as follows

(we will step through a handful of detailed examples in the sections that follow).

1. Choose a class of model by importing the appropriate estimator class from Scikit-Learn.

2. Choose model hyperparameters by instantiating this class with desired values.

3. Arrange data into a features matrix and target vector following the discussion above.

4. Fit the model to your data by calling the ``fit()`` method of the model instance.

5. Apply the Model to new data:

- For supervised learning, often we predict labels for unknown data using the ``predict()`` method.

- For unsupervised learning, we often transform or infer properties of the data using the ``transform()`` or ``predict()`` method.

We will now step through several simple examples of applying supervised and unsupervised learning methods.

#### 2. Choose model hyperparameters

An important point is that \*a class of model is not the same as an instance of a model\*.

Once we have decided on our model class, there are still some options open to us.

Depending on the model class we are working with, we might need to answer one or more questions like the following:

- Would we like to fit for the offset (i.e., \*y\*-intercept)?

- Would we like the model to be normalized?

- Would we like to preprocess our features to add model flexibility?

- What degree of regularization would we like to use in our model?

- How many model components would we like to use?

These are examples of the important choices that must be made \*once the model class is selected\*.

These choices are often represented as \*hyperparameters\*, or parameters that must be set before the model is fit to data.

In Scikit-Learn, hyperparameters are chosen by passing values at model instantiation.

We will explore how you can quantitatively motivate the choice of hyperparameters.

For our linear regression example, we can instantiate the ``LinearRegression`` class and specify that we would like to fit the intercept using the ``fit\_intercept`` hyperparameter:

Model.fit(X, y) performs the fitting / creation of the model in Scikit Learn.

Model.predict(NewX) will predict y values for the provided x values in NewX.

Scikit Learn train\_test\_split will split the set.

Principle Compone Analysis (PCA) – reduced the four dimensions of the Iris data to two.

A universal function (or [ufunc](https://numpy.org/doc/stable/glossary.html" \l "term-ufunc) for short) is a function that operates on **[ndarrays](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html" \l "numpy.ndarray" \o "numpy.ndarray)** in an element-by-element fashion, supporting [array broadcasting](https://numpy.org/doc/stable/user/basics.ufuncs.html#ufuncs-broadcasting), [type casting](https://numpy.org/doc/stable/user/basics.ufuncs.html#ufuncs-casting), and several other standard features. That is, a ufunc is a “[vectorized](https://numpy.org/doc/stable/glossary.html#term-vectorization)” wrapper for a function that takes a fixed number of specific inputs and produces a fixed number of specific outputs. For detailed information on universal functions, see [Universal functions (ufunc) basics](https://numpy.org/doc/stable/user/basics.ufuncs.html#ufuncs-basics).

There are currently more than 60 universal functions defined in **[numpy](https://numpy.org/doc/stable/reference/index.html" \l "module-numpy" \o "numpy)** on one or more types, covering a wide variety of operations. Some of these ufuncs are called automatically on arrays when the relevant infix notation is used (*e.g.*, [**add(a, b)**](https://numpy.org/doc/stable/reference/generated/numpy.add.html#numpy.add) is called internally when a + b is written and *a* or *b* is an **[ndarray](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html" \l "numpy.ndarray" \o "numpy.ndarray)**). Nevertheless, you may still want to use the ufunc call in order to use the optional output argument(s) to place the output(s) in an object (or objects) of your choice.

Recall that each ufunc operates element-by-element. Therefore, each scalar ufunc will be described as if acting on a set of scalar inputs to return a set of scalar outputs.

Functions that operate element by element on whole arrays.

To see the documentation for a specific ufunc, use [**info**](https://numpy.org/doc/stable/reference/generated/numpy.info.html#numpy.info). For example, np.info(np.sin). Because ufuncs are written in C (for speed) and linked into Python with NumPy’s ufunc facility, Python’s help() function finds this page whenever help() is called on a ufunc.

A detailed explanation of ufuncs can be found in the docs for [Universal functions (ufunc)](https://numpy.org/doc/stable/reference/ufuncs.html#ufuncs).

**Calling ufuncs:** op(\*x[, out], where=True, \*\*kwargs)

Apply *op* to the arguments *\*x* elementwise, broadcasting the arguments.

The broadcasting rules are:

* Dimensions of length 1 may be prepended to either array.
* Arrays may be repeated along dimensions of length 1.