Python Tips & Tricks *for Machine Learning*

from **ActiveState**

General Tips

- Stop wasting time explicitly importing all the data science libraries you need to use in your dev environment. Instead, use Pyforest to lazily import them for you only when you need them:

```
pip install pyforest
```

- Tired of writing for loops to join lists? Use the zip function instead:

Initial data	`a = ("John", "Charles", "Mike")` `b = ("For", "Against", "For")`
Zip function	`x = zip(a, b)`
Result	`(('John', For), ('Charles', 'Against'), ('Mike', 'For'))`

Pandas Tips

- Shortcut your initial data analysis with the pandas' profile function, which generates a detailed report of your data (missing values, variables, counts, etc) in just one line of code. For example:

```
df = pd.read_csv('somedata.csv')
df.profile_report()
```

- Speed up pandas operations with pandarallel. For example, instead of `df.progress_apply()` use `df.parallel_apply()` to run the process in parallel.

- Need to unstack a table? Use pandas, which can convert one level of an index into the columns of your data frame. For example:

Initial data (trimmed)

```
state           email_provider
AK              aol.com
                hotmail.cm
                cox.net
                kitty.com
AR              deleo.com
AZ              yahoo.com
                aol.com
                cox.net
                nikolozakes.org
                parvis.com
CA              gmail.com
                cox.net
                aol.com
```

Unstack code

```
----------------------------->
clients.groupby('state')['email_provider'].value_counts
().unstack().fillna(0)
----------------------------->
```

Result (trimmed)

```
email_provider      angalich.com     ankeny.org     aol.com
  state
    ak                  0.0             0.0            2.0
    ar                  0.0             0.0            0.0
    az                  0.0             0.0            2.0
    ca                  0.0             1.0            9.0
    co                  0.0             0.0            0.0
    ct                  0.0             0.0            1.0
    dc                  0.0             0.0            0.0
    fl                  0.0             0.0            4.0
    ga                  0.0             0.0            1.0
```

Visualization Tips

- Tired of rendering static plots in Jupyter with `%matplotlib` ? Try importing `%matplotlib notebook` which gives you interactive plots you can resize and zoom in on.

- Want to more easily spot patterns in tabulated data? Create a heatmap using Seaborn's gradient capabilities. For example:

```
hm = sns.light.palette('green', as_cmap = True)
style  = df.style.background_gradient(cmap = hm)
```

	A	B	C	D	E
0	-1.264051	1.52791	-0.970711	0.47056	-0.100697
1	0.303793	-1.72596	1.5851	0.13297	-1.10686
2	1.57823	0.10798	-0.764048	-0.775189	1.38385
3	0.760385	-0.285647	0.538367	-2.0839	0.937782

Jupyter Notebook Tips

- Need to debug your code in Jupyter notebooks? Just type `%debug` to launch an interactive debugger that takes you back to the point where the exception happened. Press `q` to exit.

- Computational costs matter. To check the running time of a block of code in a Jupyter notebook preface the code block with `%%time`

- Working with Python in a Jupyter Notebook, but wish you had access to R? Now you can run both of them together by typing `pip install rpy2.`

Scikit-Learn Tips

- Always use the `stratify` parameter to ensure test and train sets are split into equal proportions for better prediction and reproducibility of results. For example:

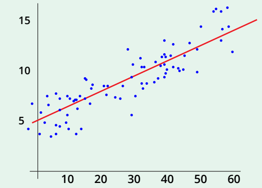
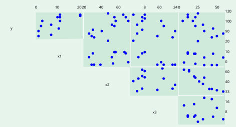
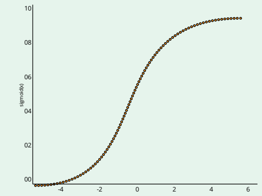
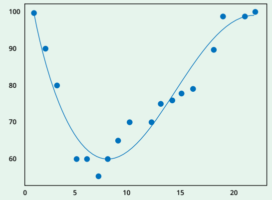
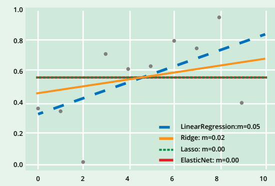
```
test_x, train_x, test_y, train_y = tain_test_split (x, y,
random_state = 59, stratify = y)
```

- Missing values in your dataset? Don't settle for univariate methods to create the missing values when scikit-learn's multivariate input based on k-Nearest Neighbor can offer better accuracy:

```
impute = KNNImputer(n_neighbors=2)
```

Predictive Analysis Guide

And finally, a simple rule guide for when to apply which regression technique when doing predictive analysis:

Regression Type	Usage	Result
Linear	Used to predict the value of a variable (called the dependent variable) based on the value of another variable `Y' = bX + A`	
Stepwise	Used when you have many variables and want to identify a subset of predictors $b_{j.std}=b_j(s_x * s_y^{-1})$	
Logistic	Used when the dependent variable is a binary value $\beta_0=\beta_1x_1+\beta_2x_2$	
Polynomial	Used for curvilinear data $\beta_0+\beta_0x_1+\epsilon$	
Lasso	Best used when you have a small number of significant parameters $N^{-1}\Sigma_{i=1}^{N} f(x_i, y_i, \alpha, \beta)$	
Ridge	Best used when you have a large number of significant parameters $\beta=(X^TX+\lambda*I)^{-1}X^Ty$	
ElasticNet	The happy medium between Lasso and Ridge $\|\|\beta\|\|_1=\Sigma_{j=1}^{p}\|\beta_j\|$	