//

Machine Learning VII

maXbox Starter 69 - Data Science with Max

From Document to Sentiment ?
Sentimental !

This tutor puts a trip to the kingdom of prime classes with dataframe knowledge.

First we generate a list of all prime numbers less than 10000:

```
   myprimes:= TStringlist.Create;
   pcount:= 0;
   sumcnt:= 0;
   myprimes.add('N,'+'P')
   for i:= 1 to 10000 do begin    //1229 primes
     if isprimeRM(i) then begin
        myprimes.add(itoa(i)+','+'1')
        inc(pcount)
        sumcnt:= sumcnt+i;
     end else
        myprimes.add(itoa(i)+','+'0');
   end;
   myprimes.savetofile(exepath+'primes10000.csv')
   writeln('found primes: '+itoa(pcount))
   writeln('delta count primes: '+itoa(sumcnt))
   myprimes.free;
```

So we get 1229 primes in the following file format (P=1 is a prime):

```
N,P
1,0
2,1
3,1
4,0
5,1
6,0
7,1
```

Next we load this csv file into a dataframe structure. This assumes that the data is comma-separated. Hope you read the preceding tutorial 65, 66 and 67 with clustering and 3D plot of dataframes.
By default, *pd.read_csv* uses header=0 (when names parameter is also not speci-

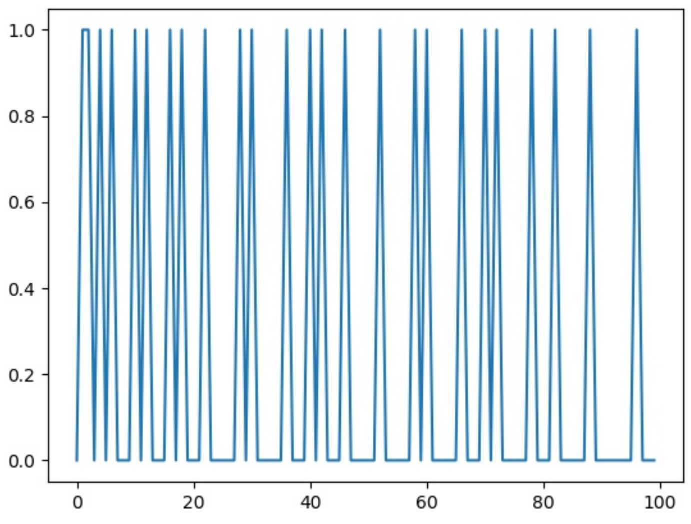
fied) which means the first (i.e. 0th-indexed) line is interpreted as column names N and P we have (primes.columns).

```
>>> primes = pd.read_csv(BASEPATH2+'primes10000.csv',sep=',',encoding = "ISO-885
9-1",header=0)
So next is the default same:

>>> primes = pd.read_csv(BASEPATH2+'primes10000.csv')

>>> primes.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 2 columns):
N    10000 non-null int64
P    10000 non-null int64
dtypes: int64(2)
memory usage: 156.3 KB
>>> primes.head(8)
     N  P
0    1  0
1    2  1
2    3  1
3    4  0
4    5  1
5    6  0
6    7  1
7    8  0
```

Now we want to visualize those primes, but before starting it is important to note what a prime number is.

1. A prime number has to be a positive integer
2. Divisible by exactly 2 integers (1 and itself)
3. 1 is not a prime number

```
>>> primes['P'][0:100].plot()
```



Now we see the first 25 primes as a binary distribution, means each prime is 1 and between is 0 and the more numbers we have the less primes we get (distance get larger). Lets dive into feature extraction and produce a third column with only prime numbers from P and N:

```
primes['pnumber'] = np.where(primes['P']==1, primes.N, 0)
```

This is read by get all numbers primes.N where a flag P as prime must 1 else 0. And a last column which extracts difference between those primes, called delta3:

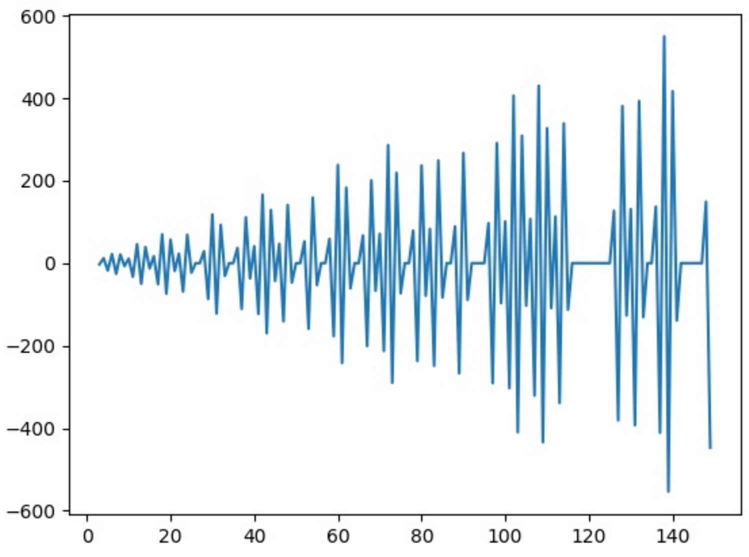
```
primes['delta3'] = primes['pnumber'].diff().diff().diff()
```

Our enhanced dataset has now this structure:

```
>>> primes.head(15)
     N  P  pnumber  delta3
0    1  0        0     0.0
1    2  1        2     0.0
2    3  1        3     0.0
3    4  0        0    -3.0
4    5  1        5    12.0
5    6  0        0   -18.0
6    7  1        7    22.0
7    8  0        0   -26.0
8    9  0        0    21.0
9   10  0        0    -7.0
10  11  1       11    11.0
11  12  0        0   -33.0
12  13  1       13    46.0
13  14  0        0   -50.0
14  15  0        0    39.0
```

And you know what, we plot our new feature:

```
primes.delta3[1:150].plot()
<matplotlib.axes._subplots.AxesSubplot object at 0x000000224C227710>
```



While there are many magic ways to solve the mystery of prime numbers, here is a new different approach with a classifier. We opt for a LogisticRegression as classifier. In statistics, the logistic model is a statistical model that is usually taken to apply to a binary dependent variable.
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

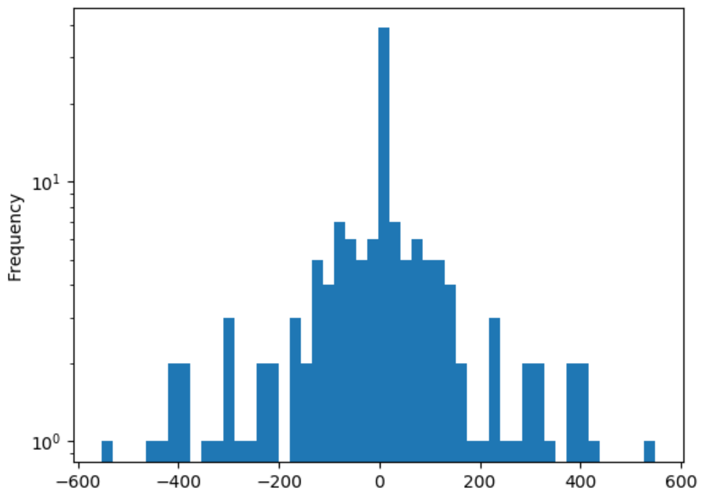
```
from sklearn.linear_model import LogisticRegression
```

for this we need a feature input and a target output as supervised learning:

```
>>> Xp=primes.delta3
>>> yp=primes.P
>>> clfp = LogisticRegression(solver = 'liblinear',C=1.0).fit(Xp, yp)

then we get:
learn\utils\validation.py", line 552, in check_array
    "if it contains a single sample.".format(array))
ValueError: Expected 2D array, got 1D array instead:
array=[0. 0. 0. ... 0. 0. 0.].
Reshape your data either using array.reshape(-1, 1) if your data has a single fe
ature or array.reshape(1, -1) if it contains a single sample.

OK, we reshape it to a single feature:
>>> Xp=Xp.reshape(-1,1)

and we get another error (terror):
ValueError:
Input contains NaN, infinity or a value too large for dtype('float64').

OK, we fill the NaNs with a defined value:
>>> primes.delta4.fillna(0, inplace=True)
>>> clfp = LogisticRegression(solver = 'liblinear',C=1.0).fit(Xp, yp)
>>> print(clfp.score(Xp,yp))
0.8623
```

Wow we have a first score (0.8623)! What does it mean? Lets have a look at the histogram of triple delta (3diff):

```
>>> primes.delta3[1:150].plot(kind='hist', bins=50, logy=True)

<matplotlib.axes._subplots.AxesSubplot object at 0x000000224C6B2D30>
```



The most values a 0 means no prime. The others a kind of distribution or density of primes to the distance from one prime to the next prime. So our classifier thinks he can predict the next prime which would be a sensation but it is NOT. We test that with a confusion matrix to get the real targets.
This example demonstrates how a confusion matrix can be used to assess the performance of a classifier. All off-diagonal elements on the confusion matrix represent misclassified data.

```
>>> print(metrics.confusion_matrix(yp, clfp.predict(Xp)))
```

```
[[8558  213]
 [1164   65]]
```

We have a lot of false negatives, the true value is one and a prime but gets classified (predicted) as zero as non prime numbers!

```
>>> print(metrics.classification_report(yp, clfp.predict(Xp)))
              precision    recall  f1-score   support

           0       0.88      0.98      0.93      8771
           1       0.23      0.05      0.09      1229

   micro avg       0.86      0.86      0.86     10000
   macro avg       0.56      0.51      0.51     10000
weighted avg       0.80      0.86      0.82     10000
```

But its more than we expect because the samples are unbalanced means we have a lot more non primes (8771) than primes (1229). On the other side the score results with none of train and test split or the crossvalidation.
The research is open:

https://www.quora.com/Could-you-train-a-machine-learner-to-predict-the-next-prime-number-I-know-there-is-no-pattern-to-PNs-I-am-wondering-if-the-ML-would-figure-it-out

https://stackoverflow.com/questions/14266409/why-can-machine-learning-not-recognise-prime-numbers

The basic difficulty here is that the sequence of primes

2, 3, 5, 7, 11, 13, 17, 19, 23, . . .

behaves "unpredictably" or "randomly", we don't have a (useful) exact formula for the nth prime number!

In the end some descriptive summary of our research (correlations) for your own experiments:

```
>>> primes.corr()
                  N       P  pnumber      delta3
N        1.0000e+00 -0.0432   0.1659 -1.9158e-08
P       -4.3176e-02  1.0000   0.8280  2.8073e-01
pnumber  1.6586e-01  0.8280   1.0000  3.2176e-01
delta3  -1.9158e-08  0.2807   0.3218  1.0000e+00

>>> primes.describe()
                N           P     pnumber       delta3
count  10000.0000  10000.0000  10000.0000   10000.0000
mean    5000.5000      0.1229    573.6396       0.0001
std     2886.8957      0.3283   1850.9238    9033.2410
min        1.0000      0.0000      0.0000  -39722.0000
25%     2500.7500      0.0000      0.0000       0.0000
50%     5000.5000      0.0000      0.0000       0.0000
75%     7500.2500      0.0000      0.0000       0.0000
max    10000.0000      1.0000   9973.0000   39718.0000
```

Appendix: See also two other classifiers

SGDClassifier
LogisticRegressionCV

```
SGDClassifier
    incrementally trained logistic regression (when given parameter loss="log").
LogisticRegressionCV
    Logistic regression with built-in cross validation
```

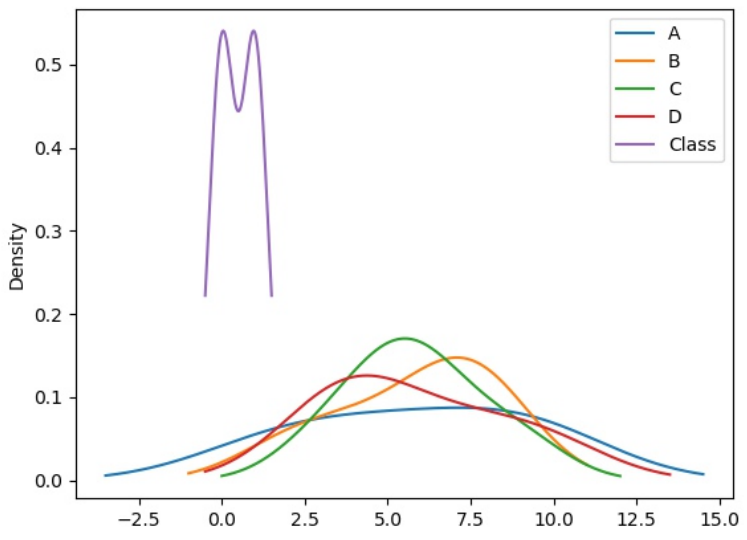
Notes:

The underlying C implementation uses a random number generator to select features when fitting the model. It is thus not uncommon, to have slightly different results for the same input data. If that happens, try with a smaller tol parameter or set random state to 0.

Mathematically, a histogram is a mapping of bins (intervals or numbers) to frequencies. More technically, it can be used to approximate a probability density function (PDF) of the underlying variable that we see later on.
Moving on from a frequency table above (density=**False** counts at y-axis), a true histogram first <bins> the range of values and then counts the number of values that fall into each bin or interval. A plot of a histogram uses its bin edges on the x-axis and the corresponding frequencies on the y-axis.

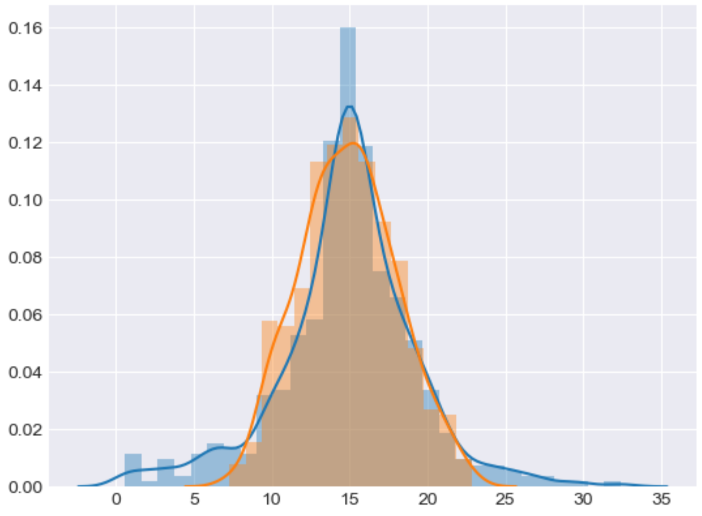
Sticking with the Pandas library, you can create and overlay density plots using plot.kde(), which is available for both [Series] and [DataFrame] objects.

```
df.iloc[0:,0:4].plot.kde()
```



This is also possible for our binary targets to see a probabilistic distribution of the target class values (labels in supervised learning): **[0. 0. 1. 1. 0. 1.]**
Consider at last a sample of floats drawn from the Laplace and Normal distribution together. This distribution graph has fatter tails than a normal distribution and has two descriptive parameters (location and scale):

```
>>> d = np.random.laplace(loc=15, scale=3, size=500)
>>> d = np.random.normal(loc=15, scale=3, size=500)
```



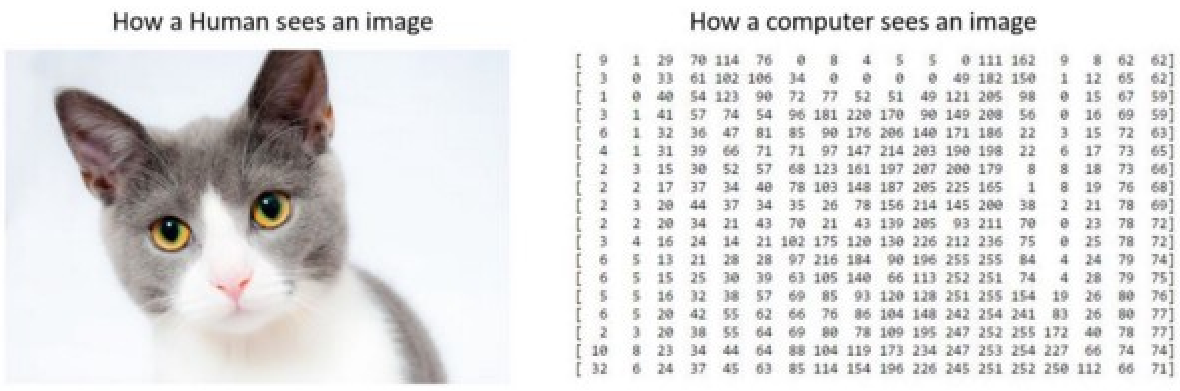
The script can be found:

http://www.softwareschule.ch/examples/classifier_compare2confusion2.py.txt

Author: Max Kleiner

Ref:
    http://www.softwareschule.ch/box.htm
    https://scikit-learn.org/stable/modules/
    https://realpython.com/python-histograms/

Doc:
    https://maxbox4.wordpress.com



```
[   9    1   29   70  114   76    0    8    4    5    5    0  111  162    9    8   62   62]
[   3    0   33   61  102  106   34    0    0    0    0   49  182  150    1   12   65   62]
[   1    0   40   54  123   90   72   77   52   51   49  121  205   98    0   15   67   59]
[   3    1   41   57   74   54   96  181  220  170   90  149  208   56    0   16   69   59]
[   6    1   32   36   47   81   85   90  176  206  140  171  186   22    3   15   72   63]
[   4    1   31   39   66   71   71   97  147  214  203  190  198   22    6   17   73   65]
[   2    3   15   30   52   57   68  123  161  197  207  200  179    8    8   18   73   66]
[   2    2   17   37   34   40   78  103  148  187  205  225  165    1    8   19   76   68]
[   2    3   20   44   37   34   35   26   78  156  214  145  200   38    2   21   78   69]
[   2    2   20   34   21   43   70   21   43  139  205   93  211   70    0   23   78   72]
[   3    4   16   24   14   21  102  175  120  130  226  212  236   75    0   25   78   72]
[   6    5   13   21   28   28   97  216  184   90  196  255  255   84    4   24   79   74]
[   6    5   15   25   30   39   63  105  140   66  113  252  251   74    4   28   79   75]
[   5    5   16   32   38   57   69   85   93  120  128  251  255  154   19   26   80   76]
[   6    5   20   42   55   62   66   76   86  104  148  242  254  241   83   26   80   77]
[   2    3   20   38   55   64   69   80   78  109  195  247  252  255  172   40   78   77]
[  10    8   23   34   44   64   88  104  119  173  234  247  253  254  227   66   74   74]
[  32    6   24   37   45   63   85  114  154  196  226  245  251  252  250  112   66   71]
```