//

Baseline Routines

____

maXbox Starter 84 - Baseline in Code - Max Kleiner

Machine Learning XII

"In 1950, Alan Turing published a seminal paper titled “Computing Machinery and Intelligence” in Mind magazine. In this detailed paper the question “Can Machines Think?” was proposed. The paper suggested abandoning the quest to define if a machine can think, to instead test the machine with the ‘imitation game’."

— https://www.unite.ai/what-is-the-turing-test-and-why-does-it-matter/

IEEE (IEEE Std. No. 610.12-1990) defines baseline as an agreed description and review of attributes of product, that afterward serve as basis for further development and defining change, and this changing can be done only through formal change control procedures”.

Software baseline instability has done more to undermine acquisition credibility and complicate effective management of the acquisition of software-intensive systems than the inability to establish realistic software development cost.

Our first baseline is synthetic data generated by a synthetic data random generator (sd-rage):


```
    {
      "date": "2021-3-4",
      "confirmed": 36223,
      "deaths": 1483,
      "recovered": 33632,
      "location": EU
    }
```


In order to be useful for a machine learning classifier, the synthetic data should have certain properties. While the data can be categorical, binary, or numerical, the length of the dataset should be arbitrary and the data should be randomly generated. The random processes used to generate the data should be controllable and based on various statistical distributions. Random noise may also be placed in the dataset.

• Random Number Generator •

A random number generator is a routine that produces a sequence of numbers that would pass statistical or probabilistic tests for randomness. To be really strict, routines that generate random numbers are said to be pseudorandom number generators (often abbreviated to PRNG) to differentiate them from true random number generators that rely on some kind of random events happening at a quantum level.

```
type
  TStRandomBase = class
    private
    protected
      function rbMarsagliaGamma(aShape   : double) : double;
      function rbMontyPythonNormal : double;
    public
      {uniform distributions}
      function AsFloat : double; virtual; abstract;
      function AsInt(aUpperLimit : integer) : integer;
      function AsIntInRange(aLowerLimit : integer;
                            aUpperLimit : integer) : integer;

```

```
      {continuous non-uniform distributions}
      function AsBeta(aShape1, aShape2 : double) : double;
      function AsCauchy : double;
      function AsChiSquared(aFreedom : integer) : double;
      function AsErlang(aMean  : double;
                        aOrder : integer) : double;
      function AsExponential(aMean : double) : double;
      function AsF(aFreedom1 : integer;
                   aFreedom2 : integer) : double;
      function AsGamma(aShape : double; aScale : double) : double;
      function AsLogNormal(aMean   : double;
                           aStdDev : double) : double;
      function AsNormal(aMean   : double;
                        aStdDev : double) : double;
      function AsT(aFreedom : integer) : double;
      function AsWeibull(aShape : double;
                         aScale : double) : double;
  end;

  TStRandomSystem = class(TStRandomBase)
    private
      FSeed : integer;
    protected
      procedure rsSetSeed(aValue : integer);
    public
      constructor Create(aSeed : integer);
      function AsFloat : double; override;
      property Seed : integer read FSeed write rsSetSeed;
  end;

https://github.com/TurboPack/SysTools/blob/master/source/StRandom.pas
```

Often, in conversation, people use the term random when they really mean arbitrary. When one asks for an arbitrary number, one is saying that one doesn’t really care what number one gets: almost any number will do. By contrast, a random number is a precisely defined mathematical concept: every number should be equally likely to occur. A random number will satisfy someone who needs an arbitrary number, but not the other way around.

The first test is the simplest: the uniformity test. This is the one we were discussing earlier. Basically, the random numbers we generate are going to be checked to see that they uniformly cover the range 0.0 to 1.0. We create 100 buckets, generate 1,000,000 random numbers, and slot them into each bucket. Bucket 0 gets all the random numbers from 0.0 to 0.01, bucket 1 gets them from 0.01 to 0.02, and so on. The probability of a random number falling into a particular bucket is obviously 0.01. We calculate the chi-square value for our test and check that against the standard table, using the 99 degrees of freedom line.

```
SelfTestCRandom;
writeln('GetRandomSeedX: '+inttostr64(GetRandomSeedX));

strand:= TStRandomSystem.create(42); //if 0 then randomseedx
 writeln(floattostr(strand.asFloat))
 writeln(floattostr(strand.asFloat))
 writeln(floattostr({System.}Random(10)));
 writeln(floattostr({System.}RandomE));
 writeln(itoa(strand.asInt(10000)))
 writeln(itoa(strand.asInt(10000)))
 writeln(itoa(strand.asInt($E15)))
```

```
 UniformityTest2(strand, ChiSquare, DegsFreedom)
 writeln('ChiSquare: '+floattostr(ChiSquare)+' , degsfreedom:'+itoa(degsfreedom));

 UniformityTest3(strand, ChiSquare, DegsFreedom)
 writeln('ChiSquare: '+floattostr(ChiSquare)+' , degsfreedom:'+itoa(degsfreedom));

strand.Free;
```

Current theories state that quantum events are truly random. The time of the decay of a radioactive atom into its byproducts cannot be predicted; all we can say is that there is a certain probability that it will decay within a certain period of time, and we can estimate that probability by observing the decay of many, many atoms.

• What is baseline model in machine learning? •

A baseline is a method that uses heuristics, simple summary statistics, randomness, or machine learning to create predictions for a dataset. You can use these predictions to measure the baseline's performance (e.g., accuracy)– this metric will then become what you compare any other machine learning algorithm against.

The three most commonly used baseline algorithms are:

- •Random Prediction Algorithm.
- •Zero Rule Algorithm.
- •Naive Bayes Algorithm

When starting on a new problem that is more sticky than a conventional classification or regression problem, it is a good idea to first devise a random prediction algorithm that is specific to your prediction problem. Later you can improve upon this and devise a zero rule algorithm. Come up with a baseline model. As you will see, Naive Bayes, acts as a tremendously good baseline model because of its probabilistic approach. It is comprehensible and simple.

A machine learning algorithm tries to learn a function that models the relationship between the input (feature) data and the target variable (or label). When you test it, you will typically measure performance in one way or another. For example, your algorithm maybe 85% accurate. But what does this mean? You can infer this meaning by comparing it with a baseline's performance.

Typical baselines include those supported by scikit-learn's “dummy” estimators:

Classification baselines:

- • stratified”: generates predictions by respecting the training set's class distribution.
- • most_frequent”: always predicts the most frequent label in the training set.
- • prior”: always predicts the class that maximizes the class prior.
- • uniform”: generates predictions uniformly at random.
- • constant”: always predicts a constant label that is provided by the user.

This is useful for metrics that evaluate a non-majority class.

• Conclusion:

Hence, more often than not I build a Naive Bayes model as my baseline and then go on towards building more complex ones such as Decision Tree, Support Vector Machine or Random Forest.

If you are dealing with a specific domain of machine learning (such as recommender systems), then you will typically pick baselines that are current state-of-the-art(SoTA) approaches - since you will usually want to demonstrate that your approach does better than these. For example, while you evaluate a new collaborative filtering algorithm, you may want to compare it to matrix factorization — which itself is a learning algorithm, but is now a popular baseline since it has been so successful in recommender system research.

Appendix for Python to get a Polynomial Regression Baseline:

Base Schema for Polynomial Regression

A straightforward way of doing multivariate polynomial regression for Python.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

rng=np.random.RandomState(1)
# get some sin data
X=10 * rng.rand(50)
y=np.sin(X)+0.1*rng.randn(50)

poly=make_pipeline(PolynomialFeatures(degree=7),LinearRegression())
Xfit=np.linspace(0,10,1000)
# train and predict
poly.fit(X[:,np.newaxis],y)
>>>Pipeline(steps=[('polynomialfeatures', PolynomialFeatures(degree=7)),
                ('linearregression', LinearRegression())])
yfit=poly.predict(Xfit[:,np.newaxis])
# plot it
plt.scatter(X,y)
>>><matplotlib.collections.PathCollection object at 0x000000523600E470>
plt.title("Polynomial mX7fit graph")
>>>Text(0.5, 1.0, 'Polynomial mX7fit graph')
plt.plot(Xfit, yfit)
>>>[<matplotlib.lines.Line2D object at 0x0000005236C1CF28>]
plt.show()
```

Ref:

https://simulatoran.com/what-is-baseline-model-in-machine-learning/
https://pomber.github.io/covid19/timeseries.json
script: 1026_json_automation_refactor2.txt

Doc:

https://maxbox4.wordpress.com
>>> https://entwickler-konferenz.de/speaker/max-kleiner/