BAYCOMP Tutorial

What is Baycomp?

Baycomp is a Python library for comparing the accuracy of Machine Learning classifiers using Bayesian statistics, as opposed to frequentist statistics.

The name Baycomp is derived as a contraction of 'Bayesian Comparison'.

Why use Baycomp?

Baycomp analysis of superior classifiers differs from the orthodox approach of determining which classifiers outperform others on single, or multiple, datasets.

Typically, given two classifiers A and B, the probability that A outperforms B is determined through a null-hypothesis significance test.

By using this method, a significance level (α) is set (typically at 0.05). A p-value is then measured. This assumes a null hypothesis that the two classifiers are equal in their accuracy. Thus, a p-value less than 0.05 would reject the null hypothesis and indicate that A outperforms B.

However, it is incorrect to assume that, at a p-value of 0.05, there is a 95% chance that classifier A outperforms B. This only proves that the accuracy scores are indeed different, however this is working off the incorrect null hypothesis that the two are equivalent to begin with.

Using Bayesian statistics to compare the classifiers gives us the actual probability that A outperforms B.

For further evidence and mathematical verification of this please refer to *Alessio Benavoli et al.'s "Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis"*

What can Baycomp do?

Baycomp functions can be used to compare the accuracy of two classifiers on one or multiple datasets.

The output of the functions provides us with three probabilities:

- The probability that classifier A outperforms classifier B.
- The probability that the differences in accuracies are within the region of practical equivalence (ROPE).
- The probability that classifier B outperforms classifier A.

Furthermore, Baycomp has additional class methods that allow us to visualise the results, through plotting the posterior distributions.

Where is the Baycomp codebase?

Baycomp is divided into two distinct methods:

- The first for analysing two classifiers on a single dataset. The source code can be found at the following link:
    - https://baycomp.readthedocs.io/en/latest/_modules/baycomp/single.html#CorrelatedTTest.compute_statistics
- The second method analyses two classifiers on multiple datasets. The source code can be found at the following link:
    - https://baycomp.readthedocs.io/en/latest/_modules/baycomp/multiple.html#HierarchicalTest
- A GitHub repository with the entire Baycomp source code:
    - https://github.com/janezd/baycomp

GETTING STARTED WITH BAYCOMP:

Installation of Baycomp

With Python and PIP already installed on a system, Baycomp can be installed using the following code:

```
>>> pip install baycomp
```

Import Baycomp

Once Baycomp is installed, import it in your applications by adding the *import* keyword:

```
>>> import baycomp
```

Baycomp as bc

Baycomp is usually imported under the *bc* alias. Create an alias with the *as* keyword while importing:

```
>>> import baycomp as bc
```

Now the baycomp package can be referred to as bc instead of baycomp.

PREPARING TO USE BAYCOMP:

To use Baycomp we will need to initially load one dataset, and then carry out analysis using a machine learning classifier with 10*10-fold cross validation. For this tutorial I will be using Breast Cancer data, which can be found at:

- https://www.kaggle.com/datasets/nancyalaswad90/breast-cancer-dataset

This is a 569-row x 31 column dataset. A snippet of this data can be seen below:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	1	17.99	10.38	122.80	1001.0	0.11840
1	1	20.57	17.77	132.90	1326.0	0.08474
2	1	19.69	21.25	130.00	1203.0	0.10960
3	1	11.42	20.38	77.58	386.1	0.14250
4	1	20.29	14.34	135.10	1297.0	0.10030
...	...	...	...	...	...	...
564	1	21.56	22.39	142.00	1479.0	0.11100
565	1	20.13	28.25	131.20	1261.0	0.09780
566	1	16.60	28.08	108.30	858.1	0.08455
567	1	20.60	29.33	140.10	1265.0	0.11780
568	0	7.76	24.54	47.92	181.0	0.05263

569 rows × 31 columns

For this analysis, I will be comparing the performance of the Random Forest and Multi-Layer Perceptron classifiers. Import the classifiers using the following code:

```
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.ensemble import RandomForestClassifier
```

An important step to prepare for Baycomp analysis is to compute k-fold cross validation of the classifier on the data.

- K-fold Cross-Validation is when the dataset is split into a K number of folds and is used to evaluate the model's ability when given new data.
- In this case, I split the dataset into 90/10 training and test data. Thus, 10% of the data will be used as a test set.
- There are 10 distinct sets of 10% of the data.
- I will test each of these 10 times, hence the name 10*10-fold.
- This will produce 100 accuracy scores for each classifier on the data.

To ensure the data is split like-for-like when assessing each classifier, we must embed both analyses in the same loop.

```
from sklearn.model_selection import cross_val_score
```

Use the following code to assess the accuracy of the Random Forest and Multi-Layer Perceptron classifiers. Note that both functions set the same random state and both are executed within the same for-loop for train/test split consistency:

```
classifier1 = RandomForestClassifier(max_depth=2, random_state=12345)
classifier2 = MLPClassifier(random_state=12345)

kfold = KFold(n_splits=10, shuffle = False, random_state = 123)

acc1 = []
acc2 = []

for i in range(0, 10):
    acc_RF = cross_val_score(classifier1, X, y, cv=kfold, n_jobs=-1)
    acc_MLP = cross_val_score(classifier2, X, y, cv=kfold, n_jobs=-1)

    acc1.append(acc_RF)
    acc2.append(acc_MLP)
```

The following code will take the lists of accuracies calculated in the previous code and convert them into data frames of length 100, in preparation to be used for Baycomp:

```
acc1 = [arr.tolist() for arr in acc1]
acc_list = [item for sublist in acc1 for item in sublist]
df1 = pd.DataFrame(acc_list)
df1.columns = ['RF']
acc2 = [arr.tolist() for arr in acc2]
acc_list = [item for sublist in acc2 for item in sublist]
df2 = pd.DataFrame(acc_list)
df2.columns = ['MLP']
```

Merge the accuracy scores of each classifier into a single data frame:

```
frames = [df, df2]
results = pd.concat(frames, axis = 1)
results = results.T
for i in range(0, 100):
    results = results.rename(columns = {i : 'Run ' + str(i+1)})

results
```

The output of this will be a table with 3 rows and 100 columns. Each row represents the vector of accuracy scores for its corresponding classifier. Each column represents the accuracy score for its corresponding iteration. Below is a snippet of what it should look like:

	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	...	Run 91	Run 92	Run 93
RF	0.965517	0.862069	0.912281	0.964912	0.964912	0.964912	0.912281	0.982143	0.946429	0.982143	...	0.965517	0.862069	0.912281
MLP	0.877193	0.912281	0.947368	0.912281	1.000000	0.964912	0.947368	0.982456	0.912281	0.982143	...	0.877193	0.912281	0.947368

2 rows × 100 columns

<u>TWO ON SINGLE:</u>

- Rather intuitively, the two_on_single function uses Bayesian Correlated T-tests to evaluate the accuracy of two different classifiers on one dataset by computing the posterior.
- The posterior describes the distribution of the mean difference of accuracies between the two classifiers.
- In this analysis, our hypothesis is that one classifier outperforms another, and assess the probability of that happening -> e.g. P(RF > MLP).
- Querying the posterior distribution allows us to find this probability.
- In addition, we will use a ROPE value, the Region of Practical Equivalence. This is the difference in accuracies between the two classifiers that is statistically insignificant. Therefore, we can conclude the classifiers are equivalent in practice. This region is depicted by the two vertical lines.

<u>Usage Guide:</u>

- Call two_on_single function using Baycomp alias bc.
- The first two arguments are the accuracy values of:
    1. The first classifier

2. The second classifier

- Runs argument is required and should be expressed as an integer value of the number of runs of the classifier on the dataset being examined. In this case we did a 10*10-fold cross validation, so runs = 10.

- ROPE, or Region of Practical Equivalence as explained above, should be expressed as a percentage/ fraction and represents the max difference in accuracy for which the classifiers would be deemed equivalent.

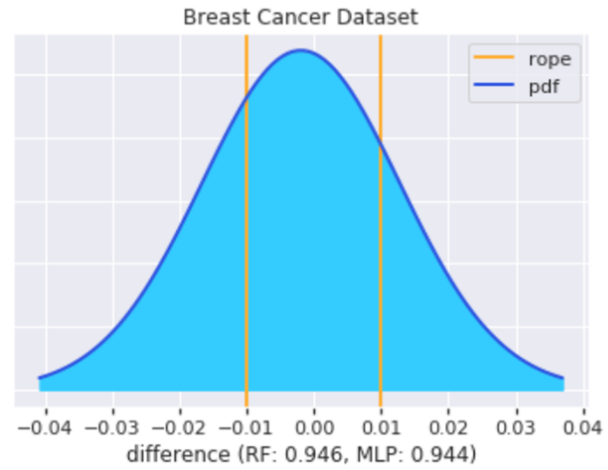
- Setting plot = True will produce the plot of the posterior distribution

- The names argument labels the accuracies of each classifier in the plot. They should be entered in the same chronological order as the first two arguments.

```
probs, fig = bc.two_on_single(results.loc['RF'].values, results.loc['MLP'].values,runs=10,
                              rope=1/100, plot=True, names=['RF','MLP'])
plt.title('Breast Cancer Dataset')
probs = (round(probs[0], 2), round(probs[1], 2), round(probs[2], 2))
print(probs)
```

Output:

```
(0.29, 0.5, 0.21)
```



Interpretation of TWO ON SINGLE:

- The dark blue line represents the posterior distribution in the difference in accuracies between the two classifiers.

- Rope value is set at 1/100 (1%), a widely acceptable upper-bound parameter for practical equivalence.

- We can find the probability that classifier 1 is superior to classifier 2 by finding the integral of the posterior in the range [-inf, -0.01].

- Likewise, the probability classifier 2 is superior to classifier 1 is the integral of the posterior in the range [0.01, inf].

- The probability that the two are equivalent is the integral of the posterior in the range [-0.01, 0.01].

- The probabilities are displayed by using function print(probs).

- We see from this that:

    - The probability Random Forest outperforms Multi-Layer Perceptron is 48%.
    - The probability that the differences in accuracies are within the region of practical equivalence is 45%.
    - The probability that Multi-Layer Perceptron outperforms Random Forest is 6.5%.