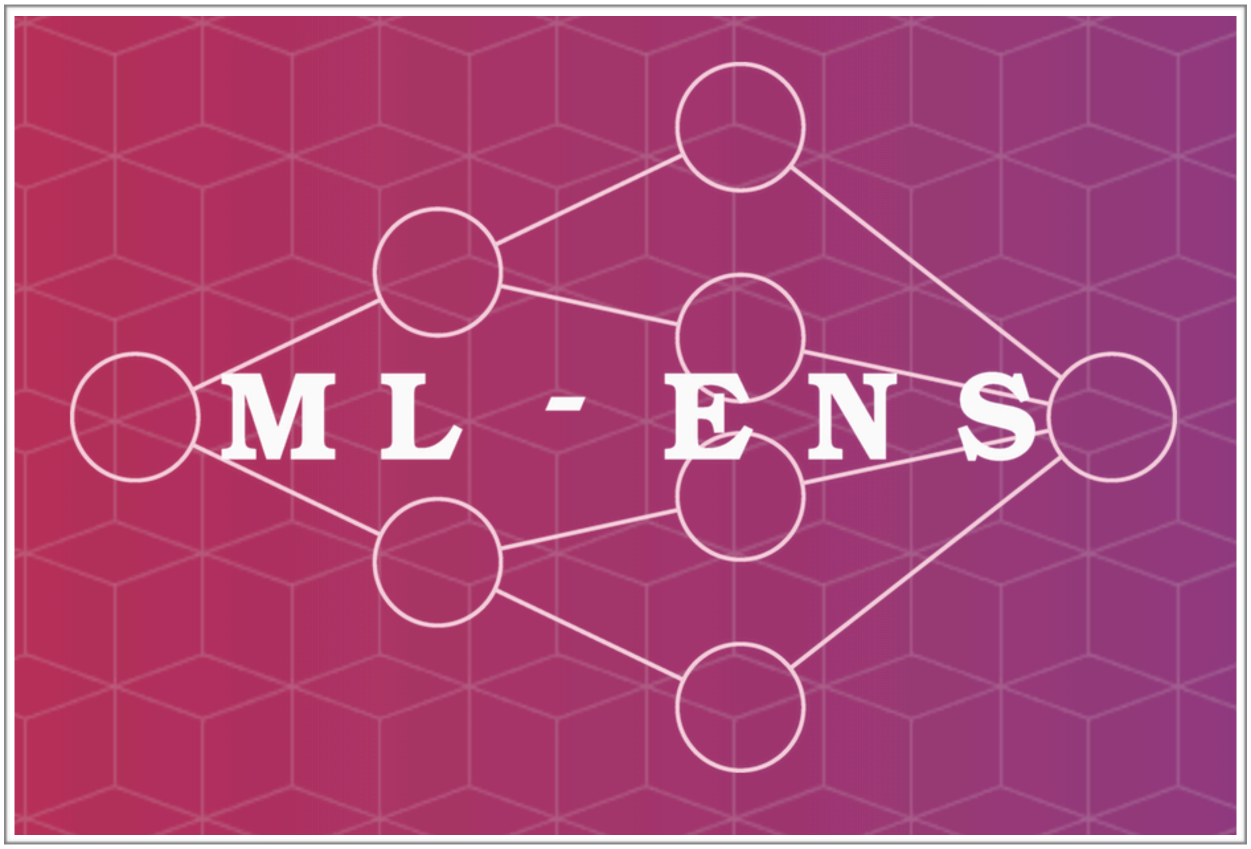
Ensembling Using MLEns

Exploring ensemble efficiency and optimization using the MLEnsemble library.



Bryce Badura

Spring 2018

Ensembling Using MLEns

Exploring ensemble efficiency and optimization using the MLEnsemble library.

The Goal

The purpose of this exploration is to understand how MLEns and its classes might be used to add additional types of ensembles to the BASS algorithm. The goal is to find combinations that consistently perform with high accuracy and under reasonable time constraints. Various datasets and ensemble elements were tested to determine the optimal outcomes, which will be highlighted below.

Ready Made Ensemble Classes

MLEns comes with four, ready made ensemble classes available. These classes include:

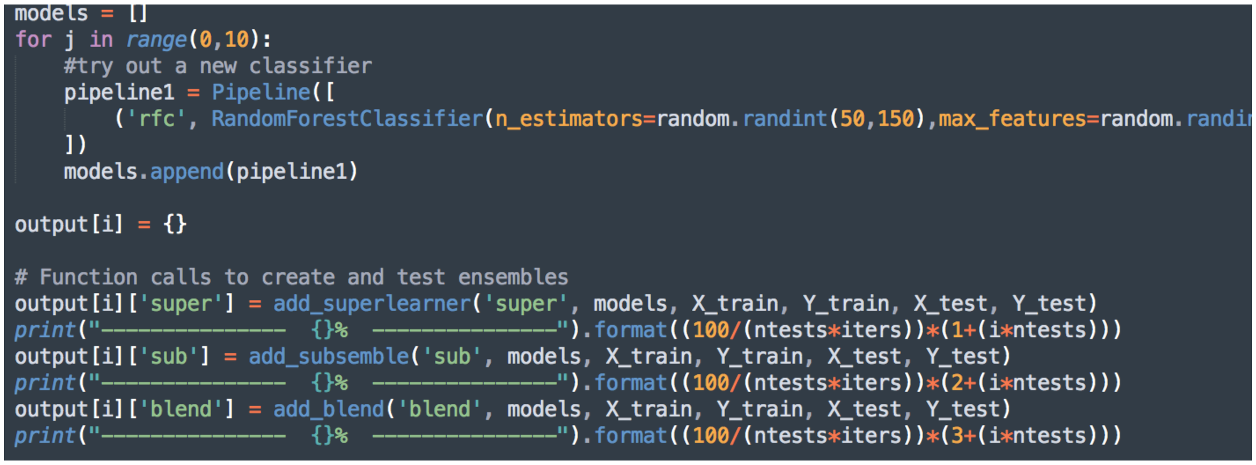
- **Super Learner** (the stacking ensemble) - a supervised ensemble algorithm. This class is used in conjunction with the `propagate_features` flag to ensure that models are being trained on both input models and original features.
- **Subsemble** - a supervised ensemble algorithm that uses subsets of the full data to fit a layer. Additionally, "By partitioning the data into subset and fitting on those, a Subsemble can reduce training time considerably if estimators does not scale linearly. Moreover, Subsemble allows estimators to learn different patterns from each subset, and so can improve the overall performance by achieving a tighter fit on each subset."
- **Blend Ensemble** - a supervised ensemble closely related to the Super Learner. It differs in that, "to estimate the prediction matrix Z used by the meta learner, it uses a subset of the data to predict its complement, and the meta learner is fitted on those predictions."
- **Sequential Ensemble*** - allows users to build ensembles with different classes of layers. Instead of setting parameters upfront during instantiation, "the user specifies parameters for each layer when calling add. The user

must thus specify what type of layer is being added (blend, super learner, subsemble), estimators, preprocessing if applicable, and any layer-specific parameters."

*Due to the nature of this project, Sequential Ensemble was not extensively explored, as it is a wrapper classes for layering the first 3 unique ensemble classes

Testing of Ensemble Classes

A program called ensemble_1.2 was built to run the various iterations required to test the ensemble classes and inputs. The program takes in two CSV files (a train and test file) and then chooses the 20 best attributes to use for model generation using SelectKBest. Input models are generated (however many and of whatever type we find necessary) and functions are called to run different different combinations of input models on MLEns classes. An example of this process looks like the following:



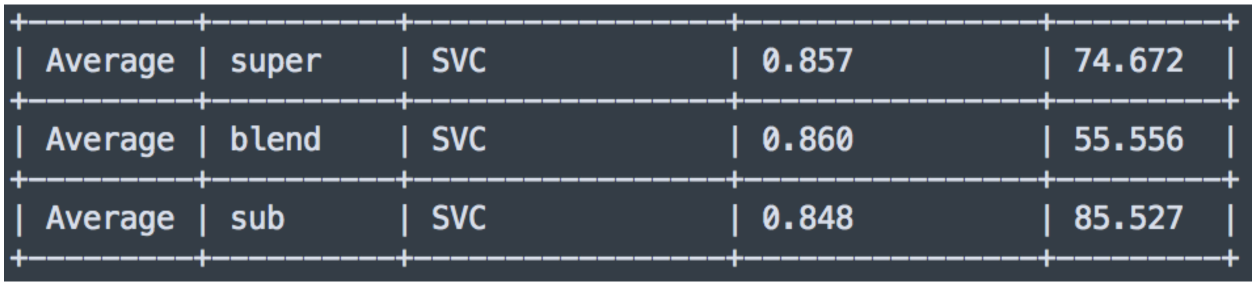
```
models = []
for j in range(0,10):
    #try out a new classifier
    pipeline1 = Pipeline([
        ('rfc', RandomForestClassifier(n_estimators=random.randint(50,150),max_features=random.randi
    ])
    models.append(pipeline1)

output[i] = {}

# Function calls to create and test ensembles
output[i]['super'] = add_superlearner('super', models, X_train, Y_train, X_test, Y_test)
print("---------------  {}%  ---------------").format((100/(ntests*iters))*(1+(i*ntests)))
output[i]['sub'] = add_subsemble('sub', models, X_train, Y_train, X_test, Y_test)
print("---------------  {}%  ---------------").format((100/(ntests*iters))*(2+(i*ntests)))
output[i]['blend'] = add_blend('blend', models, X_train, Y_train, X_test, Y_test)
print("---------------  {}%  ---------------").format((100/(ntests*iters))*(3+(i*ntests)))
```

In this example, we generate 10 RandomForestClassifier models, and then run the Super Learner, Subsemble, and Blend Ensemble algorithms using the 10 random forest models as input. The output will be saved in python dictionary format. You may notice the "i" index in the output dictionary, which is used simply because we are running these tests 5 times and taking their average for more reliable results. This index helps to separate the data and make for easier aggregation after all testing is complete.

Once all the data is generated, we can take the averages and produce an output table that aggregates the averages and reports the average accuracy and time of each model combination. The data, when put into excel and plotted, is quite useful in finding models with promising utility. Additionally, accuracy scores are compared to the same model run on the current version of BASS to see if there is any potential benefit using MLEns. An example of the output is show below:



```
+---------+----------+----------------+---------------+---------+
| Average | super    | SVC            | 0.857         | 74.672  |
+---------+----------+----------------+---------------+---------+
| Average | blend    | SVC            | 0.860         | 55.556  |
+---------+----------+----------------+---------------+---------+
| Average | sub      | SVC            | 0.848         | 85.527  |
+---------+----------+----------------+---------------+---------+
```

Results

After running many tests, the **Blend Ensemble class using multiple XGBoost models as input** performed the best over an array of datasets. It consistently ranked in the highest accuracies and performed in the shortest amount of time when tested against other model combinations. An excel file*, which can be found in the "ople-project" GitHub associated with this project, has all the test data and plots to depict the performance of these tests. As is evidenced by the data, there are plenty of datasets that have scores from this ensemble pair that are higher than that achieved by BASS (including obtrain, ionosphere, ecoli). And while MLEns didn't beat BASS every time, I feel there is enough evidence to support investigating the utility of MLEns usage in some of the BASS calculations.

It is also worth noting that certain datasets that do not traditionally perform well on "forest" algorithms (like mtrain/mtest) do perform well using an ensemble of K-Neighbors classifier. Additionally, K-Neighbors and Decision Tree ensembles perform much faster and sometimes don't have a significantly worse result, so if time/performance constraints are an issue, these could be potentially viable options.

*Please see attached file named averages.xlsx

Conclusions

MLEns is a powerful library that simplifies some of the more complex components of ensembling techniques. It doesn't require much overhead or additional code to implement, but as shown in the results from testing, has the potential to bring results that are more accurate than the current implementation of BASS. While I am not sure of the inner-workings of the BASS algorithm, if there is a way to simply add an additional test that incorporates MLEns classes, that might be a worthwhile investment!

References

For future reference to MLEns and its inner-workings, use the link below. There are limited resources, and this page doesn't appear in a Google search. However, I believe this is the official documentation page for this library (and is the source of all quoted text above):

- Home Page
    - http://ml-ensemble.com/
- Documentation
    - http://ml-ensemble.com/info/index.html
- GitHub for MLEns
    - https://github.com/flennerhag/mlens
- GitHub for this project
    - https://github.com/bbadura/ople-project