MLJ : Notes

Dainish Jabeen

June 21, 2023

1 General

MLJ is a machine learning toolbox for Julia, that wraps a large number of models and provides great tools such as resampling and evaluation [2] [3].

Models are structs storing hyperparameters.

Listing 1: Basics

```
#Split data randomly with seed (rng)
#       if y and X need to be separated
y, X = unpack(df,==(:col);rng=123);

df |> pretty #Output pretty version

#Load model
VAR = @load model pkg=""

var = VAR() #Default parameters

#evaluate model with cross validation
 via error measures

evaluate(var,X,y,resampling=CV(shuffle=true),measures=[rms])
```

1.1 Types

Each model has an expected variable type needed to train it:

target_scitype(model) provides type needed.

Listing 2: Change type

```
#Change type of var

y = coerce(df, :col=type,...)
```

2 Machines

Used to store training outcomes.

Listing 3: Machines

```
1
2       mach = machine(model,X,y)
3
4       #70:30 partition giving an index vector
5       train, test = partition(eachindex(y),0.7)
6
7       fit!(mach,train)
8       yhat = predict(mach,X[text,:]);
9
10      #error rate
11      misclassification_rate(yhat, test)
```

Can also evaulate! machines.

Machines will throw a warning if there is a mixmatch of types (continous and count).

2.1 Prediction

To predict with a given probability for a class use **broadcast**.

For a matrix of all classes: **pdf**.

For an output of a class: **predict_mode**.

2.2 Inspection

Two methods: fitted_params and report.

Fitted params: The learned parameters for a machine.

report: More detailed stats on machine.

3 Hyperparameter tuning

Check notebook Resampling(Julia-MLJ) for example.

- Create ensemble model
- Create ranges for parameters
- wrap in tuned model
- create mach, fit data
- report on best model
- mach will become the best model

learning_curve() gives a performance line for a tuning parameter.

4 Pipeline

Linear set of models chained together, that can be evaluated and used as a single model.

For example: Change var type |> tune parameter |> model |> train.

Similar to recipes, pipelines are a series of transforms on data, then fed into a model.

Listing 4: Pipeline example

```
1
2       pipe = X |> Standardizer |> Model
3
4       TunedModel(model=pipe)
5       Machine(pipe,X,y)
6       evaulate(pipe)
```

5 Resampling

Listing 5: Resampling

```
1
2       #Split data into sections of chosen percentage,
3       #       can also shuffle
4       holdout = Holdout(; fraction_train=0.7,
5                   shuffle=nothing,
6                   rng=nothing)
7
8       # Cross validate over number of folds
9       cv = CV(; nfolds=6,  shuffle=nothing, rng=nothing)
10
11      # For classification problems aims to retain the connection
12      #       between the predictors in train and test with the
13      #       response class level.
14
15      stratified_cv = StratifiedCV(; nfolds=6,
16                            shuffle=false,
17                            rng=Random.GLOBAL_RNG)
18
19
20      #CV for when observations are chronological and not expected
21      #       to be independent.
22
23      tscv = TimeSeriesCV(; nfolds=4)
```

6 Parallel

Using the parameter **acceleration** in train!, can be set to:

- CPU1 : Single threaded

- CPUProcesses: Multi process (core)
- CPUThreads: Mutli threaded
- CUDALibs: GPU Computation

Or can set **MLJ.default_resource()** to one of the above.

7 Transformers

Work similar to models, but only require the input data as they are unsupervised.

Listing 6: Standardizer

```
1
2       #features are the chosen predictors to standardise.
3       model = Standardizer(features=)
```

7.1 PCA

- PCA: Linear
- Kernal PCA: Non linear
- Probabilistic: Gaussian

Listing 7: Standardizer

```
1
2       #params include max output dims, kernel, solver, inverse transform
3       #, convergence total, max iterations.
4       model = KernelPCA()
```

8 Performance

$$ConfusionMatrix()(y_{pred}, y)$$

$$false\ pos\ rate,\ true\ postive\ rate,\ thresholds = roc(y_{probpredict}, y)$$

9 Save

Listing 8: Saving machine

```
1
2       # SAVE
3       MLJ.save("name.jls",mach)
4
5       # LOAD
6       mach = machine("name.jlso")
```

10 Scikit Learn: Notes

Many of the models used in MLJ are derived from Scikit learn [1].

10.1 Support vector machine

Advantages:

- Works well with high dimensional spaces
- Works well if p > n
- memory efficient
- Versatile (different kernels)

Three types:
SVC: Radial kernel, with cost as key balancing hyperparameter.
NuSVC: Radial kernel, with number of support vectors as key balancing hyperparameter.
LinearSVC: Linear kernel.

OneclassSVM is used for outlier detection.

$$O(n_{factor} * n^3_{samples}) \tag{10.1}$$

For regression SVR, NuSVR and LinearSVR exist.

If certain classes or samples have higher importance weights can be used.

10.1.1 MultiClass

If more than two classes are used SVC and NuSVC use one versus one model approach, linear uses one versus all.

$$O(1-1) = \frac{n(n-1)}{2} \tag{10.2}$$

$$O(1-n) = n \tag{10.3}$$

10.1.2 Probabilistic value

Use model ProbabilisticSVC, to provide probability values.

Use wrapper BinaryThresholdPredictor(model; threshold=0.5) to change to deterministic.

10.1.3 Parameters

Cost: Trade off between misclassification and decision surface simplicity, lower C gives a simpler decision surface.

Gamma: How much influence an observation has.

10.1.4 Tips

- Data should be scaled
- L1 penalization yields a sparse selection of predictors.
- C and gamma are key parameters
- Use Grid search CV

10.2 Outlier Detection

Outlier: Data far from others, in the training data.
Novelty: Outlier not in training data.

Unsupervised methods include using estimator.fit and estimator.predict functions. Outlier methods isolation forest and neighbor local outlier factor. OneClassSVM needs training for effective outlier detection.

Primary novelty detection is one OneClassSVM.

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[2] A. D. Blaom, F. Kiraly, T. Lienart, Y. Simillides, D. Arenas, and S. J. Vollmer, "MLJ: A julia package for composable machine learning," *Journal of Open Source Software*, vol. 5, no. 55, p. 2704, 2020. DOI: 10.21105/joss.02704. [Online]. Available: https://doi.org/10.21105/joss.02704.

[3] A. D. Blaom and S. J. Vollmer, *Flexible model composition in machine learning and its implementation in MLJ*, 2020. arXiv: 2012.15505 [cs.LG].