QuickIntro

The qeML package: "Quick and easy" wrappers for machine learning

"Easy for learners, powerful for advanced users"

Norm Matloff, UC Davis,

I am a professor of computer science, and a former professor of statistics, highly active in the areas of machine learning and statistical computing, bio.

What this package is about

- "Quick and Easy" ML
  - "Works right out of the box!"
  - much simpler interface than **tidymodels**, **caret**, **mlr3**, **superlearner**, **SuperML** etc.
  - easy for learners, powerful/convenient for experts
- Special Feature for ML Learners
  - includes a **tutorial** on major ML methods
- Special Features for Those Experienced in ML
  - variety of functions for feeature selection and model development
  - large variety of ML algorithms, including some novel/unusual ones
  - advanced plotting utilities, e.g. Double Descent
  - includes **tutorials** on special topics

Overview

(Also see extensive Function List section below.)

The letters 'qe' in the package title stand for "quick and easy," alluding to the convenience goal of the package. We bring together a variety of machine learning

(ML) tools from standard R packages, providing wrappers with a simple, uniform interface. Hence the term "quick and easy."

For instance, consider the **mlb1** data included in the package, consisting of data on professional baseball players. Say we wish to predict weight of a player. For SVM, we would make the simple call

```
qeSVM(mlb1,'Weight')
```

For gradient boosting, the call would be similar,

```
qeGBoost(mlb1,'Weight')
```

and so on. It couldn't be easier!

Default values are used on the above calls, but nondefaults can be specified, e.g.

```
qeSVM(mlb1,'Weight',gamma=0.8)
```

Prediction

Each qe-series function is paired with a **predict** method, e.g. predict player weight:

```
> data(mlb1)
> z <- qeGBoost(mlb1,'Weight',holdout=NULL)
> predict(z,data.frame(Position='Catcher',Height=73,Age=28))
[1] 204.2406
```

A catcher of height 73 and age 28 would be predicted to have weight about 204.

Categorical variables can be predicted too. Where possible, class probabilities are computed in addition to class:

```
> w <- qeGBoost(mlb1,'Position',holdout=NULL)
> predict(w,data.frame(Height=73,Weight=185,Age=28))
$predClasses
[1] "Relief_Pitcher"

$probs
        Catcher First_Baseman Outfielder Relief_Pitcher Second_Baseman
[1,] 0.02396515    0.03167778  0.2369061      0.2830575      0.1421796
      Shortstop Starting_Pitcher Third_Baseman
[1,] 0.0592867         0.1824601    0.04046717
```

A player of height 73, weight 185 and age 28 would be predicted to be a relief pitcher, with probability 0.28.

Holdout sets

By default, the qe functions reserve a holdout set on which to assess accuracy.

```
> z <- qeRF(mlb1,'Weight')
holdout set has  101 rows
Loading required package: randomForest
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.
> z$testAcc
[1] 14.45285
> z$trainAcc
[1] 8.23018
> z$baseAcc
[1] 17.22356
```

The mean absolute prediction error on the holdout data was about 14.5 pounds. As is typical, it was much smaller on the training set, 8.2.

If one simply predicted every player using the overall mean weight, the MAPE would be about 17.2.

One can skip holdout by setting the **holdout** argument to NULL.

Of course, since the holdout set is random, the same is true for the accuracy numbers. To gauge the predictedive power of a model over many holdout sets, one can use **replicMeans()**, which is available in qeML by automatic loading of the **regtools** package. Say for 100 holdout sets:

```
> replicMeans(100,"qeRF(mlb1,'Weight')$testAcc")
[1] 13.6354
attr(,"stderr")
[1] 0.1147791
```

So the true MAPE for this model on new data is estimated to be 13.6. The standard error is also output, to gauge whether 100 replicates is enough.

Dimension reduction/Feature Selection

One can preprocess the data, both when fitting the training data and later when predicting new cases. For instance, consider the **pef** dataset included with the package. It consists of Census data on programmers and engineers in 2000.

```
> head(pef)
       age     educ occ sex wageinc wkswrkd
1 50.30082 zzzOther 102   2   75000      52
2 41.10139 zzzOther 101   1   12300      20
3 24.67374 zzzOther 102   2   15400      52
4 50.19951 zzzOther 100   1       0      52
5 51.18112 zzzOther 100   2     160       1
6 57.70413 zzzOther 100   1       0       0
```

First, let's try PCA. The **qePCA()** function calculates the principal components, retains the major ones, then applies a specified ML method on the reduced dataset. We'll specify that we want as many principal components as will comprise 60% of the total variance, and will use k-Nearest Neighbor analysis.

```
> data(pef)
>  w <- qePCA(pef,'wageinc','qeKNN',pcaProp=0.6)
holdout set has  1000 rows
> w$testAcc
[1] 24351.91
> w$baseAcc
[1] 31444.26
```

On average, our predictions were off about about $24K. If we were to just predict using the overall mean income, MAPE would be about $31K.

A much more powerful method of dimension reduction is FOCI (Feature Ordering by Conditional Independence). We have a wrapper.

Here we will use it on a 50K subset of the Million Songs dataset from the UCI Machine Language Data Repository. The goal is to predict the year of release of the song, based on 90 different audio measurements.

```
> system.time(z <- qeFOCI(s50,'V1'))
    user   system  elapsed
1464.245   22.246  208.174
```

It can be time-consuming. But it did reduce dimension:

```
> dim(s50)
[1] 50000    91
> dim(z$newData)
[1] 50000     9
```

FOCI settled on a set of 8 of the original 90 predictors.

Let's try predicting using random forests, say the **ranger** version:

```
> w <- qeRFranger(z$newData,'V1')
holdout set has  1000 rows
Loading required package: ranger
> w$testAcc
[1] 6.661694
> w$trainAcc
[1] 3.39568
> w$baseAcc
[1] 8.169616
```

So, we seem to be able to predict release year of a song by about 6.7 years on average. If we were to simply use the overall average year as our prediction, on average we'd be off by about 8.2 years, so yes, the features do help. Of course, we might try the same on the full 500K dataset, but used a subset here to save time.

Note again the tiny value of the training set accuracy, about 3.4 years! This is a great reminder of the fact that training set accuracy tends to be overly optimistic.

Function list

- ML algorithms
  - **qeAdaBoost()**: Ada Boosting, wraps **Jousboost** pkg
  - **qeDT()**: decision trees, wraps **party** pkg
  - **qeGBoost()**: gradient boosting, wraps **gbm** pkg
  - **qeISO()**: isotonice regression
  - **qeKNN()**: k-Nearest Neighbors, wraps **regtools** pkg; includes predictor importance settings; allows linear interpolation within a bin
  - **qeKNNna()**: k-Nearest Neighbors for NA-ridden data, special algorithm
  - **qeLASSO()**: LASSO and ridge regression, wraps **glmment** pkg
  - **qelightGBoost()**: gradient boosting, wraps **lightgbm** pkg
  - **qeLin()**: wraps R's **lm()**; can be used for multiclass classification, for speed
  - **qeLogit()**: wraps R's **glm()**
  - **qeNeural()**: wraps **keras** package, including CNN
  - **qePolyLASSO()**: LASSO/ridge applied to polynomial regression; wraps **glmnet**, **polyreg** pkgs
  - **qePolyLin()**: polynomial regression on linear models; uses Moore-Penrose inverse if overfitting; wraps **polyreg** pkg
  - **qePolyLog()**: polynomial regression on logistic models; wraps **polyreg** pkg
  - **qeRF()**: random forests, wraps **randomforest** pkg
  - **qeRFgrf**: random forests, wraps **grf** pkg; allows linear interpolation within a bin
  - **qeRFranger()**: random forests, wraps **ranger** pkg
  - **qeskRF()**: random forests, wraps Python **Scilearn** pkg
  - **qeskSVM()**: SVM, wraps Python **Scilearn** pkg
  - **qeSVM()**: SVM, wraps **e1071** pkg
  - **qeSVMliquid()**: SVM, wraps **liquid SVM** pkg

- k-NN, dec. trees, random forests, gradient boosting, SVM, linear/gen. linear models, ridge, LASSO, NNs, CNNs

- feature selection and model-fitting
  - **qeFOCI()**: fully nonparametric method for feature selection
  - **qeLASSO()**: for fit and/or feature selection
  - **qePCA()**: find principal components, number specified by user, then fit the resulting model, according to **qe*** function specified by user
  - **qeUMAP()**: same as **qePCA()** but using UMAP
  - **qeFT()**: automated grid hyperparameter search, *with Bonferroni-Dunn corrected standard errors*
  - **replicMeans()**: (from **regtools**, included in **qeML**) averages output, e.g. **testAcc**, over many holdout sets
  - **qeDoubleD()**: computation and plotting for exploring Double Descent
  - **qeROC()**: ROC computation and plotting, wraps **pROC** pkg

- application-specific functions (elementary)
  - **qeText()** text classification
  - **qeTS():** time series
  - *Image classification*: Our **imageClassR** package uses qe functions for this. (Under construction.)

- utilities
  - **qeCompare()**: compare the accuracy various ML methods on a given dataset
  - **qeParallel()**: apply "Software Alchemy" to parallize qe functions
  - **qePCA()**: apply PCA before running specified qe ML function
  - **qeUMAP()**: apply UMAP before running specified qe ML function