OCAPIS: R package for Ordinal Classification And Preprocessing In Scala

M. Cristina Heredia-Gómez[a,*], Salvador García[a,**], Pedro Antonio Gutiérrez[b], Francisco Herrera[a]

[a] *DaSCI Andalusian Institute of Data Science and Computational Intelligence, University of Granada, Spain*

[b] *Department of Computer Science and Numerical Analysis, University of Córdoba, Campus de Rabanales, edificio Albert Einstein 14017, Córdoba, Spain*

Abstract

Ordinal Data are those where a natural order exist between the labels. The classification and pre-processing of this type of data is attracting more and more interest in the area of machine learning, due to its presence in many common problems. Traditionally, ordinal classification problems have been approached as nominal problems. However, that implies not taking into account their natural order constraints. In this paper, an innovative R package named **ocapis** (Ordinal Classification and Preprocessing In Scala) is introduced. Implemented mainly in Scala and available through Github, this library includes four learners and two pre-processing algorithms for ordinal and monotonic data. Main features of the package and examples of installation and use are explained throughout this manuscript.

Keywords: Ordinal classification, Ordinal regression, Data preprocessing, Machine learning, R, Scala

1. Introduction

The development of supervised classification and pre-processing techniques for data with numerical targets is a central topic in machine learning

*Corresponding author

**Correspondence

Email addresses: mrcrstnherediagmez@gmail.com (M. Cristina Heredia-Gómez), salvagl@decsai.ugr.es (Salvador García)

and data science [1, 2]. Nonetheless, it is now that more attention is being paid to classification and pre-processing of ordinal and monotonic data, given their big presence in everyday problems. For example, there is an increasing amount of data from service rating surveys whose target is based on an ordinal scale $[bad, regular, good, very\ good, excellent]$ and therefore class labels incorporate order information, consequently an instance with class *excellent* has a higher rating than another from *regular* class [3]. Monotonic data is a special case of ordinal data where monotonicity constraints exist between instances and class labels in such a way that given two instances x, x' where $x \leq x' \Rightarrow f(x) \leq f(x')$. That is, if an instance is smaller or equal than other instance, then its associated class cannot be greater. Monotonic constraints are present in many real data, such as house prices, since they increase directly with the size of the house and the year of construction and inversely with the distance to the city. They are also very present in finance, where there are some companies that dominate others for all financial indicators[4]. The main challenges when dealing with this kind of data are, on the one hand, considering the ordering information to build more realistic models, and on the other hand, using different misclassification cost depending on error, as labeling an instance as *good* when its real label is *very good* is not the same error as labeling it as *bad*.

Although there are numerous scientific publications on ordinal classification (also ordinal regression), there are very few open source libraries for performing ordinal and monotonic classification and pre-processing tasks.

For R we find the very recent **ordinalNet** package [5], that fits ordinal regression models with elastic penalty and supports model families from element-wise link multinomial-ordinal class. Another very recent R package is **ordinal** [6] which also implements Ordered Regression models, commonly named proportional odds models. Like **ordinalNet**, it allows specifying a link function from $[logit, probit, loglog, cloglog, cauchit]$. Since last year, **monMLP** [7] is also available which offers a multi-layer perceptron neural network where monotonicity constraints can be optionally applied. Although there are others packages related to ordinal data, they offer an isolated task or algorithm, like ordinal data conversion [8, 9], mixture models [10], penalized ratio models [11, 12], multiple ordinal tobit models [13], clustering [14] or rule models [15].

For Matlab and Octave we find **orca** [3] a more complete library than those mentioned above, which offers many algorithms for ordinal data classification.

However, there are three main issues with the software mentioned above. First, both **ordinalNet** and **ordinal** essentially just offer highly customizable proportional odds models, without considering other techniques. Something similar happens with **monMLP** which offers multilayer perceptron models. Second, none of them offer pre-processing techniques for ordinal data. Third, although **orca** offers many classification techniques, it is less accessible, more complicated to install and less efficient, specially when dealing with high dimensional data.
In this paper an innovative and efficient R package named **ocapis** [1] is presented. It is built mainly in Scala [16], a pretty young JVM language well known for its scalability, mixed paradigm (object-oriented and functional programming), mixin-composition constructs for composing classes and traits, decomposition of objects by pattern matching and its powerful abstraction for types and values, which has made Scala one of the most used languages in Big Data [17, 18].

Developing **ocapis** primarily in Scala along with R has been possible by using the very recent **rscala** [19] package. The proposed package is, to our knowledge, the third R package built in Scala after **shallot** [20] and **bamboo** [21], both from **rscala** creator. **Ocapis** aims to provide an open source library of classification and pre-processing methods for ordinal data that currently lack an implementation in R, including non-linear ordinal classification techniques and one of the most recent instance selector proposed in the literature.
The rest of the manuscript is arranged as follows. Section 2 presents the software and implemented algorithms. Section 3 shows some illustrative examples of use. Section 4 exposes the experimental framework and results. Finally, Section 5 sets out conclusions.

2. Software

The importance of creating specific techniques for data of an ordinal nature is beyond all doubt. Since the problem was first studied in statistics by using a link function to model underlying probabilities [22], the field of ordinal classification has evolved a great deal in recent years [23, 24]. In this new

[1] https://github.com/CristinaHG/OCAPIS

package, four of the best-known techniques for ordinal data classification are implemented, along with two pre-processing algorithms, an ordinal feature selector adapted to deal also with monotonic data and a newly proposed instance selector:

- svmop. The Support Vector Machine with Ordered Partitions (SVMOP classifier) is an ensemble of weighted support vector machines for ordinal regression proposed in [25], based on Frank & Hall binary decomposition method [26].

- pom. The Proportional Odd Model for Ordinal Regression (POM) is a member of a family of linear models known as cumulative link models or ordered regression models, proposed by [27]. It is based on a link function to model class probabilities. Accepted link functions are ($logit, probit, cloglog, loglog, cauchit$), where *logit* is usually the standard choice.

- kdlor. Kernel Discriminant Learning for Ordinal Regression (KDLOR) is a Kernel version of LDA applicable to non-linear data of an ordinal nature. Proposed by [28], it minimizes the distance within classes and maximizes the distance between classes, while considering the order information of the different classes.

- wknnor. Weighted k-Nearest-Neighbor for ordinal classification (WKNNOR) proposed by [29] maps neighbors distances to weights according to a kernel function. Accepted kernels are: *rectangular, triangular, epanechnikov, biweight, triweight, cosine, gaussian, inversion*. The algorithm has been adapted to cope with monotonic data, incorporating the monotonicity constraints suggested in [30].

- fselector. This Feature selector for monotonic classification was originally proposed in [31]. The pre-processing algorithm is based on Fuzzy Rank Mutual Information (FRMI) [32] and the search strategy of min-redundancy and max-relevance (mRMR) is used to select best features.

- iselector. Training Set Selection for Monotonic Ordinal Classification. This new proposal [4] introduces a triphasic instance selector where first, feature selection is performed, then a collision removal is carried out, and finally an evaluation metrics process is applied.

3. Examples of use

All classification algorithms are designed to have a fit and an analogous predict method. In the following example an ordinal dataset named **balace-scale** is loaded. Then an example about how to apply the two pre-processing techniques over the training set is given. Finally, we illustrate how to perform classification and prediction using the SVMOP, POM, KDLOR and WKNNOR classifiers.

```r
# Load train and test data
dattrain<-read.table("train_balance-scale.0", sep=" ")
trainlabels<-dattrain[,ncol(dattrain)] # train labels
traindata=dattrain[,-ncol(dattrain)] # train data
dattest<-read.table("test_balance-scale.0", sep=" ")
testdata<-dattest[,-ncol(dattest)] # test labels
testlabels<-dattest[,ncol(dattest)] # test data

# Select the three most important features using k and beta=2
selected<-fselector(traindata,trainlabels,2,2,3)
trainselected<-traindata[,selected]

# Select the most relevant instances with a candidate rate=0.02, collision rate=0.1 and considering maximum 5 neighbors
selected<-iselector(traindata,trainlabels,0.02,0.1,5)
trainselected<-selected[,-ncol(selected)]
trainlabels<-selected[,ncol(selected)]

# Classifying using SVMOP using weights per instance, cost=0.1 and gamma=0.1
modelstrain<-svmofit(traindata,trainlabels,TRUE,0.1,0.1)
predictions<-svmopredict(modelstrain,testdata)
sum(predictions[[2]]==testlabels)/nrow(dattest)
[1] 0.9235669

# Classifying using POM with logistic link function
fit<-pomfit(traindata,trainlabels,"logistic")
predictions<-pompredict(fit,testdata)
projections<-predictions[[1]]
predictedLabels<-predictions[[2]]
sum(predictedLabels==testlabels)/nrow(dattest)
[1] 0.910828

# Classifying using KDLOR with RBF kernel, optimization parameter=10, parameter for H matrix=0.001 and kernel param =1
```

```
33 myfit<-kdlortrain(traindata,trainlabels,"rbf",10,0.001,1)
34 pred<-kdlorpredict(myfit,traindata,testdata)
35 sum(pred[[1]]==testlabels)/nrow(dattest)
36 [1] 0.8343949
37
38 # Classifying using WKNNOR considering 5 nearest neighbors, euclidean
   ↪  distance, rectangular kernel to compute weights and without
   ↪  monotonicity constraints
39 predictions<-wknnor(traindata,trainlabels,testdata,5,2,
40 "rectangular",FALSE)
41 sum(predictions==testlabels)/nrow(dattest)
42 [1] 0.7515924
```

In the previous example the first 7 lines read the train and test datasets, separating the class labels from the data. In lines 9-11 a feature selection is performed over the training data, choosing the three most relevant features. Similarly, in lines 13-16 an instance selection is performed over the training set. As it returns a complete dataset with the selected instances, we make it our new training set. Then an example of the use of the four implemented classifiers is given. For each of them we start by fitting the model using the training data. After that, predictions are made using the test data. Finally, model accuracy is computed and shown for each model.

4. Experimental framework and results

Experiments have been carried out thought a comparison of performance and CPU time consumption between the only software solution mentioned above that implements three of this four classification techniques, **orca** [3], and **ocapis**. For performance evaluations, two widely used metrics in the field of ordinal classification have been used, named **MZE** (Mean Zero-one Error) and **MAE** (Mean Absolute Error).
The Mean Zero-one Error is the error rate of the classifier:

$$MZE = \frac{1}{N}\sum_{i=1}^{N}[[y_i^* \neq y_i]] = 1 - Accuracy,$$

where y_i, y_i^* are the real and predicted values respectively. This metric ranges from 0 to 1 and relates to global performance, without considering the order. The MAE is the average deviation in absolute value of the predicted rank from the true one[3]:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - y_i^*|,$$

where (y_i, y_i^*) represents each real-prediction pair. MAE values range from 0 to $Q-1$, where Q denotes the number of categories, and it uses an absolute cost.

The datasets used for the experiments are described in Table 1. The parameter configuration used is shown in Table 2. The parameters has been left by default to illustrate performance. Table 3 shows the performance comparison between **orca** and **ocapis**, where check-marks represent cases where **ocapis** performs better than **orca**. In order to illustrate the pre-processing techniques behavior, Table 4 and Table 5 show their performance over the mentioned datasets, where check-marks are used to show cases where the pre-processing has shown to improve the base classification results from Table 3. Finally, Table 7 shows CPU times for **orca** and **ocapis** classifiers, while Table 6 shows CPU time for the two pre-processing algorithms implemented in **ocapis**.

Table 1: Ordinal datasets used in experiments

Dataset	Instances	Features	Classes
balance-scale	625	4	3
winequality-red	1599	11	6
SWD	1000	10	4
contact-lenses	24	6	3
toy	300	2	5
ESL	488	4	9
LEV	1000	4	5
Automobile	205	71	6
Pasture	36	25	3
Squash-stored	52	51	3

Table 2: Parameters configuration used for experiments

Algorithm	Configuration
SVM [33]	C =0.1,γ=0.1
POM [27]	logistic linkfunction
KDLOR [28]	RBF kernel, d=10, u=0.001, k=1
WKNNOR [29]	Rectangular kernel, k=5, distance=1
FSelector [31]	k, β =2, selects half of the characteristics
ISelector [4]	candidates=0.01,collisions=0.02, kEdition=5

Table 3: Performance comparison between orca and ocapis (orca/ocapis)

	SVM		POM		KDLOR		WKNNOR	
Dataset	MAE	MZE	MAE	MZE	MAE	MZE	MAE	MZE
balance-scale	0.0890/0.0890	0.0764/0.0760	0.1019/0.1019	0.0891/0.0891	0.1656/0.1656	0.1656/0.1656	0.4076	0.2484
winequality-red	0.5120/0.5050✓	0.4325/0.4300✓	0.4475/0.4425✓	0.4100/0.4020✓	0.5000/0.5100	0.4470/0.4600✓	2.6350	0.9950
SWD	0.4400/0.4400	0.4280/0.4280	0.4800/0.4800	0.4640/0.4640	0.5560/0.5080✓	0.4840/0.4560✓	1.3240	0.8400
contact-lenses	0.3330/0.3330	0.3330/0.3330	0.5000/ -	0.3330/ -	0.5000/0.5000	0.5000/0.5000	0.5000	0.3333
toy	0.4930/0.5860	0.4270/0.4800	0.8800/0.8800	0.6670/0.6670	0.1460/0.1460	0.1460/0.1460	1.9333	0.8933
ESL	0.3850/0.3770✓	0.3690/0.3600✓	0.3610/0.3610	0.3270/0.3270	0.4180/0.3930✓	0.4016/0.3524✓	1.8033	0.8852
LEV	0.4640/0.4360✓	0.4240/0.400✓	0.4120/0.4120	0.3760/0.3760	0.4840/0.4840	0.4040/0.4200	1.4400	0.7840
Automobile	2.8269/1.1540✓	0.9810/0.6920✓	- / -	- / -	1.0192/1.0192	0.7307/0.7307	2.8269	0.9808
Pasture	1/0.6670✓	0.6670/0.6670	0.7780/ -	0.6670/ -	0.6670/0.6670	0.6670/0.6670	1	0.6667
Squash-stored	0.7690/0.7690	0.6150/0.6150	0.7692/ -	0.6923/ -	0.5385/0.5385	0.5385/0.5385	0.7692	0.6150

Table 4: Performance of ocapis Feature selector

	SVM		POM		KDLOR		WKNNOR	
Dataset	MAE	MZE	MAE	MZE	MAE	MZE	MAE	MZE
balance-scale	0.5159	0.2994	0.5159	0.2994	0.4777	0.3376	0.8089	0.4458
winequality-red	0.6125	0.5200	0.5525	0.4800	0.9500	0.6275	2.6350	0.9950
SWD	0.5240	0.4960	0.5400	0.4880	0.5640	0.5040	1.2040✓	0.8200✓
contact-lenses	0.5000	0.3333	0.5000✓	0.3333✓	0.8333	0.8333	0.5000	0.3333
ESL	0.4918	0.4344	0.5000	0.4426	0.5164	0.4590	1.8032	0.7623✓
LEV	0.5720	0.5120	0.5840	0.5120	0.700	0.5720	1.4120✓	0.8040
Automobile	0.9423✓	0.5961✓	1.1346✓	0.7692✓	0.9808✓	0.7115✓	2.8269	0.9808
Pasture	1	0.6667	0.2222✓	0.2222✓	0.6667	0.6667	1	0.6667
Squash-stored	0.7692	0.6154	0.3846✓	0.3077✓	0.5385	0.5385	0.7692	0.6154

Table 5: Performance of ocapis Instance selector

	SVM		POM		KDLOR		WKNNOR	
Dataset	MAE	MZE	MAE	MZE	MAE	MZE	MAE	MZE
balance-scale	0.1338	0.1274	0.1911	0.1401	0.4522	0.4331	0.4458	0.2675
winequality-red	0.5150	0.4325	0.4400	0.4025	0.5075	0.4575✓	2.6350	0.9950
SWD	0.4280✓	0.4120✓	0.4760✓	0.4640	0.4920✓	0.4480✓	1.2800✓	0.8120✓
contact-lenses	1	1	1✓	0.5000✓	0.5000	0.8333	0.5000	0.3333
toy	1.1467	0.6533	1.1200	0.6400✓	0.5467	0.4267	1.8800✓	0.8800✓
ESL	0.5328	0.5164	0.4262	0.3934	0.6475	0.5819	2.5246	0.9508
LEV	0.4520	0.3960✓	0.4400	0.3880	0.4800✓	0.4120✓	1.3960✓	0.7440✓

Table 6: Times of preprocessing algorithms

Dataset	Feature Selector	Instance selector
balance-scale	1.0900	3.4782
winequality-red	38.9118	8.5956
SWD	7.2385	4.5675
contact-lenses	0.0089	1.665
toy	0.3214	2.6208
ESL	0.9519	2.2521
LEV	3.8742	4.367
Automobile	17.1677	1.8520
Pasture	0.1679	1.7769
Squash-stored	1.295	1.9367

Table 7: Time comparison between orca and ocapis (seconds) (orca/ocapis)

Dataset	SVM	POM	KDLOR	WKNNOR
balance-scale	3.1671/0.2194✓	0.0733/0.0034✓	0.5686/0.0022✓	0.00142
winequality-red	2.3509/2.0927✓	0.0482/0.1222	6.0023/6.9004	0.1977
SWD	1.2154/0.5304✓	0.0371/0.07723	1.2800/1.4996	1.4529
contact-lenses	1.8219/1.1578✓	0.0400/0.0500	0.1993/0.0164✓	0.0313
toy	1.3535/0.1433✓	0.0256/0.0220✓	0.2432/0.1194✓	0.0288
ESL	0.9492/0.4393✓	0.0287/0.0570	0.3676/0.2532✓	0.0368
LEV	1.2184/0.5660✓	0.0275/0.0609	1.1943/1.4839	0.0905
Automobile	1.0813/0.1780 ✓	0.1183/0.1062✓	0.2706/0.0728 ✓	0.0223
Pasture	1.0201/0.0208✓	0.0421/0.0417 ✓	0.2151/0.0117✓	0.0053
Squash-stored	1.1047/0.0245✓	0.0866/0.0616✓	0.2152/0.0153✓	0.0117

From Table 3 we may conclude that our implementation performs equal and sometimes better than **orca** algorithms. Main performance differences can be seen in SVMOP, where a lot of check-marks denotes that the SVMOP implemented in **ocapis** gets better results than the SVMOP implemented in **orca**. In spite of both uses libsvm-weights [33] implementation underneath, as it is originally implemented in C, one uses the Matlab wrapper while the other uses the Python wrapper. In KDLOR, we can see that **ocapis** performs exactly equal and in three cases better than **orca**. The cause is that while **orca** uses the QP solver from Matlab, **ocapis** uses the QP solver from the very new Scala library **Breeze** [34] still under development.

Besides that we can see that for large datasets as *Automobile* with 71 features, a preprocessing step is mandatory to reduce the problem dimensionality, as some algorithms like POM may present problems to converge with such amount of features. In this case **ocapis** is a more complete software option as it offers two preprocessing algorithms while **orca** does not include

any. From Table 4 and Table 5 we can see that classification techniques can greatly benefit from a previous preprocessing step, especially when dealing with datasets where number of features or instances is large. Lastly, from Table 7 we can point a clear advantage in performing times for **ocapis** over **orca** in the algorithms implemented in Scala, which are SVMOP, KDLOR and WKNNOR, this difference is not so large for POM, which is implemented in R. In addition, for all of them, **ocapis** gets much shorter CPU times than **orca** when dealing with high-dimensional datasets as *Automobile* (71 features), *Pasture* (25 features) and *Squash-stored* (51 features), due to its Scala implementations applying functional programming and immutability principles.
Whereas WKNNOR is not included in ORCA and not tested with these datasets in its original proposal [29], this Scala implementation has shown a very good time performance even with high-dimensional datasets. From Table 6 we conclude that even though pre-processing is usually the most expensive task in terms of computing time, **ocapis** performs well even when the number of features to select and the number of instances is high.

5. Conclusions

Considering embedded order and monotonic restrictions present in ordinal and monotonic data is crucial when developing classification and preprocessing algorithms for data of that nature.
In this paper we have presented the **ocapis** package for R. It was intended to provide efficient and scalable algorithms implemented in Scala for ordinal and monotonic data that are not yet available for researchers and practitioners of the R community. Firsts, it includes two pre-processing techniques, an instance selector and a feature selector. Second, it includes four ordinal classification algorithms, one linear (Proportional Odds Models for Ordinal Regression) and three non-linear (Kernel Discriminant Learning for Ordinal Regression, Support Vector Machines with Ordered Partitions and Weighted k-Nearest-Neighbor for Ordinal Regression).
As future work, we propose to keep maintaining and adding algorithms for ordinal and monotonic data to our package, building a package to offer the vast majority of the major techniques proposed in the literature for ordinal regression and monotonic classification. Therefore, there are good perspectives to continue improving the software in the near future.

Acknowledgements

This work is supported by the Project BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

Appendix A. Installation Guide

To install *ocapis*, R language is needed (see the R official site for further instructions on how to install it). Also, the required software includes a version of Python $\geq$2.7 (see Python installation guide), Scala $\geq$2.11 (see Scala installation guide) and libsvm-weights (see libsvm-weights README). Once the requirements are satisfied, the latest developed version of *ocapis* can be easily installed directly from Github through R with the *devtools* package[35]:

```
devtools::install_github("cristinahg/OCAPIS/OCAPIS")
```

For further installation information, check the ocapis website.

References

[1] I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, Data Mining: Practical machine learning tools and techniques, Morgan Kaufmann, 2016.

[2] V. Cherkassky, F. Mulier, Learning from data: concepts, theory, and methods. 2007.

[3] P. A. Gutierrez, M. Perez-Ortiz, J. Sanchez-Monedero, F. Fernandez-Navarro, C. Hervas-Martinez, Ordinal regression methods: survey and experimental study, IEEE Transactions on Knowledge and Data Engineering 28 (1) (2016) 127–146.
URL https://github.com/ayrna/orca

[4] J.-R. Cano, S. García, Training set selection for monotonic ordinal classification, Data & Knowledge Engineering 112 (2017) 94–105.

[5] M. Wurm, P. Rathouz, B. Hanlon, ordinalnet: Penalized ordinal regression, R package.
URL https://CRAN.R-project.org/package=ordinalNet

[6] R. H. B. Christensen, ordinal-regression models for ordinal data. r package version 2015.6-28, See http://www.cran.r-project.org/package= ordinal.

[7] A. Cannon, monmlp: Monotone multi-layer perceptron neural network, R package version 1 (3).
URL https://CRAN.R-project.org/package=monmlp

[8] H. Demirtas, Y. Wang, R. Allozi, Concurrent generation of binary, ordinal and continuous data, r package binordnonnor (2016).
URL https://CRAN.R-project.org/package=BinOrdNonNor

[9] D. W. Betebenner, toOrdinal: Function for Converting Cardinal to Ordinal Numbers by Addi
r package version 1.0-0.0 (2017).
URL https://centerforassessment.github.io/toOrdinal

[10] M. Iannario, D. Piccolo, R. Simone, Cub: A class of mixture models for ordinal data, R package version 0.1, available at: cran.r-project.org/web/packages/CUB/CUB.pdf, accessed October 29 (2016) 2016.

[11] K. Archer, Glmnetcr: fit a penalized constrained continuation ratio model for predicting an o
(2018).
URL https://CRAN.R-project.org/package=glmnetcr

[12] K. J. Archer, A. A. Williams, L1 penalized continuation ratio models for ordinal response pred
Statistics in Medicine 31 (2012) 1464–1474.
URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3718008/

[13] M. N. Wright, A. Ziegler, Multiple censored data in dentistry: A new statistical model for ana
Biometrical Journal 57 (3) (2015) 384–394.
URL https://CRAN.R-project.org/package=lmmot

[14] M. Selosse, J. Jacques, C. Biernacki, ordinalclust: a package for analyzing ordinal data.
URL https://CRAN.R-project.org/package=ordinalClust

[15] R. Hornung, Ordinal forests.
URL https://CRAN.R-project.org/package=ordinalForest

[16] M. Odersky, L. Spoon, B. Venners, Programming in scala, Artima Inc, 2008.

[17] J. Maillo, S. Ramírez, I. Triguero, F. Herrera, knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data, Knowledge-Based Systems 117 (2017) 3–15.

[18] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, F. Martínez-Álvarez, Multi-step forecasting for big data time series based on ensemble learning, Knowledge-Based Systems.

[19] D. B. Dahl, Integration of r and scala using rscala.
URL https://github.com/cran/rscala

[20] D. B. Dahl, R. Day, J. W. Tsai, Random partition distribution indexed by pairwise information, Journal of the American Statistical Association 112 (518) (2017) 721–732.
URL https://github.com/dbdahl/shallot

[21] Q. Li, D. B. Dahl, M. Vannucci, H. Joo, J. W. Tsai, Bayesian model of protein primary sequence for secondary structure prediction, PloS one 9 (10) (2014) e109832.
URL https://github.com/dbdahl/bamboo

[22] J. A. Anderson, Regression and ordered categorical variables, Journal of the Royal Statistical Society. Series B (Methodological) (1984) 1–30.

[23] C.-W. Seah, I. W. Tsang, Y.-S. Ong, Transductive ordinal regression, IEEE transactions on neural networks and learning systems 23 (7) (2012) 1074–1086.

[24] Q. Tian, S. Chen, X. Tan, Comparative study among three strategies of incorporating spatial structures to ordinal image regression, Neurocomputing 136 (2014) 152–161.

[25] W. Waegeman, L. Boullart, An ensemble of weighted support vector machines for ordinal regression, International Journal of Computer Systems Science and Engineering 3 (1) (2009) 47–51.

[26] E. Frank, M. Hall, A simple approach to ordinal classification, in: European Conference on Machine Learning, Springer, 2001, pp. 145–156.

[27] P. McCullagh, Regression models for ordinal data, Journal of the royal statistical society. Series B (Methodological) (1980) 109–142.

[28] B.-Y. Sun, J. Li, D. D. Wu, X.-M. Zhang, W.-B. Li, Kernel discriminant learning for ordinal regression, IEEE Transactions on Knowledge and Data Engineering 22 (6) (2010) 906–910.

[29] K. Hechenbichler, K. Schliep, Weighted k-nearest-neighbor techniques and ordinal classification.

[30] W. Duivesteijn, A. Feelders, Nearest neighbour classification with monotonicity constraints, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2008, pp. 301–316.

[31] Q. Hu, W. Pan, L. Zhang, D. Zhang, Y. Song, M. Guo, D. Yu, Feature selection for monotonic classification, IEEE Transactions on Fuzzy Systems 20 (1) (2012) 69–81.

[32] Q. Hu, M. Guo, D. Yu, J. Liu, Information entropy for ordinal classification, Science China Information Sciences 53 (6) (2010) 1188–1200.

[33] C.-C. Chang, C.-J. Lin, Libsvm: a library for support vector machines, ACM transactions on intelligent systems and technology (TIST) 2 (3) (2011) 27.

[34] D. Hall, D. Ramage, et al., Breeze: numerical processing library for scala (2009).
URL https://github.com/scalanlp/breeze

[35] H. Wickham, W. Chang, Devtools: tools to make developing r packages easier. r package version 1.12. 0. 2016, URL http://CRAN. R-project. org/package= devtools.

Required Metadata

Current executable software version

Nr.	**(executable) Software metadata description**	**Please fill in this column**
S1	Current software version	1.0.0
S2	Permanent link to executables of this version	*github.com/CristinaHG/OCAPIS*
S3	Legal Software License	GPL-3.0
S4	Computing platform/Operating System	Linux, OS X, Microsoft Windows
S5	Installation requirements & dependencies	Rscala, Reticulate, libsvm-weights-3.17
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	*cristinahg.github.io/OCAPIS/*
S7	Support email for questions	mrcrstnherediagmez@gmail.com

Table A.8: Software metadata (optional)

Current code version

Nr.	**Code metadata description**	**Please fill in this column**
C1	Current code version	1.0.0
C2	Permanent link to code/repository used of this code version	*github.com/CristinaHG/OCAPIS*
C3	Legal Code License	GPL-3.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	R (3.4.1), Scala (2.12), Python ($\geq$ 2.7)
C6	Compilation requirements, operating environments & dependencies	Rscala, Reticulate, libsvm-weights-3.17
C7	If available Link to developer documentation/manual	*cristinahg.github.io/OCAPIS/*
C8	Support email for questions	*mrcrstnherediagmez@gmail.com*

Table A.9: Code metadata (mandatory)