A Python package for performing penalized maximum likelihood estimation of conditional logit models using Kernel Logistic Regression

José Ángel Martín-Baos
Ricardo García-Ródenas
Department of Mathematics, Faculty of Computer Science.
University of Castilla-La Mancha. 13071, Ciudad Real, Spain.
Luis Rodriguez-Benitez
Department of Information Systems and Technologies, Faculty of Computer Science.
University of Castilla-La Mancha. 13071, Ciudad Real, Spain.

ABSTRACT

In the last few years, Machine Learning (ML) methods have acquired great popularity due to their success in numerous applications such as autonomous cars, image and voice recognition systems, automatic translation systems, etc. This success has led to an increase in the use of ML methods and the extension of their applications to areas such as transport planning.

One of the main tasks within transport planning is the analysis of transport demand. To do so, it is necessary to analyse the way in which users make their decisions about the trips they make and, therefore, be able to predict the number of passengers on the transport network in relation to respect to interventions made on the transport system. Consequently, transport policies and plans can be evaluated according to the behaviour of the passengers. Discrete choice models based on random utility maximization have been developed over the last four decades and currently they have acquired a high degree of sophistication, becoming the canonical tool for transport demand analysis. Nowadays, the use of ML methods could provide an alternative to discrete choice models, as they offer a high level of accuracy in their predictions. In addition, the analyst is relieved from the need of specifying the functional expressions for the utility functions beforehand.

A Python software package called PyKernelLogit was developed to apply a ML method called Kernel Logistic Regression (KLR) to the problem of predicting the transport demand. This package allows the user to specify a set of models using KLR and the estimation of those using a Penalized Maximum Likelihood Estimation procedure. Moreover, this tool also provides a set of indicators for goodness of fit and the application of model validation techniques. Finally, it allows to obtain the willingness to pay or value of time indicators commonly used in transport planning.

1. INTRODUCTION

The efficiency and reliability of any public transport system depends on the design, management and operation of its infrastructure. These factors have a direct impact on the economic activity of the population, on their fidelity, as well as on the environment (such as pollution). There are two main strategies for improving a public transport system. On the one hand, the reliability of the infrastructure and its operations has to be improved. On the other hand, it is necessary to define an offer of transport services which is oriented to its potential demand. To achieve the second objective, it is crucial to analyse the transport demand, i.e. it is necessary to analyse the way in which passengers choose their mode of transport and thus predict the number of passengers in the transport network. Therefore, policies and plans can be established and evaluated according to the needs of the passengers.

Traditionally, process-driven models are used to predict passenger behaviour. Discrete choice models based on random utility theory describe how a rational-decision maker chooses an alternative within a set of options depending on the characteristics of each of them and the specific peculiarities of each decision-maker. The choice process involves some latent (unobservable) functions known as *utility functions*, which measure the interest of each alternative for a decision-maker. Random utility theory supposes that any decision-maker would choose the alternative that maximizes his or her utility. The utility associated with each alternative is composed of two parts: A deterministic part (also called the *systematic part*) which depends on the attribute vector of the alternative; and a *stochastic part*, whose probability distribution determines the resulting model. The Multinomial Logit (MNL) model is the most widespread example and assumes an extreme value Gumbel distribution. These models are generally estimated using a Maximum Likelihood Estimation (MLE) methodology that allows an asymptotic distribution of the estimators to be determined and to test assumptions about the value of the parameters.

The continuous technological evolution has allowed the development of new intelligent systems and Internet of Things (IoT) devices which automatically gather information about passengers on the transport system. Consequently, the amount of information used by these models has significantly increased, and the models are becoming more complex. For this reason, there is a need for new alternatives that complement the classical demand modelling methods, which are based on Random Utility Models (RUM). In this context of massive data acquisition, new models are needed, such as data-driven models. These models build relationships between input and output data, without worrying too much about the underlying processes, using statistical and Machine Learning (ML) techniques.

In the last years, the transport community has been evaluating new proposals to predict the passenger behaviour which are based on ML. Preliminary studies have revealed a significantly higher predictive performance of ML methods, which encourages further research. In ML, the *radial basis function kernel*, or RBF kernel, is a popular kernel function used in various kernel-based machine learning algorithms, such as support vector machine (SVM) classification. The combination of MNL models with radial basis functions is known in the ML community as Kernel Logistic Regression (KLR) (Zhu & Hastie, 2005). In KLR the parameter estimation problem is based on a Penalized Maximum Likelihood Estimation (PMLE) in which the goodness of fit criterion balances empirical risk and complexity. The main advantage of the KLR method over the RUM is

that it does not require the modeler to specify a functional expression of the utilities in advance, being enough to choose certain hyperparameters such as the kernel function.

At the present moment, there are several packages for different programming languages which allows to estimate a wide variety of RUM models, such as logit models. Nonetheless, these packages do not allow to estimate discrete choice models based on KLR. In this work a Python package for estimating discrete choice models based on RUM and KLR is described. In order to avoid re-codifying functions that already exist in other software packages, it has been decided to take as a baseline a current open source package for estimating a wide variety of RUM models and to use it as a basis for aggregating the KLR models. Furthermore, it is sought to implement in this library some functions to estimate some indicators such as the Willingness to Pay or the Value of Time. The developed software package has been called PyKernelLogit.

2. RELATED WORKS

Discrete choice models based on Random Utility Theory have dominated travel behaviour research during the last decades (M. Ben-Akiva & Bierlaire, 1999; M. E. Ben-Akiva et al., 1985; McFadden, 1978; Train, 2003). For this reason, during the last years, a large number of software packages have emerged to estimate this type of discrete choice of models. These tools are usually based on the estimation of logit models and their variants, such as Nested Logit models and more recently Mixed Logit Models, among others. The following lines describe the most popular software packages which are currently available for practitioners.

2.1. Biogeme

Biogeme (Bierlaire, 2018) is an open source multi-platform package which is specially designed to estimate discrete choice models. Nonetheless, it can also be used for the estimation of general parametric models using the maximum likelihood procedure. Biogeme has been developed by Michel Bierlaire, which is a professor in the Transportation and Mobility Laboratory at the Ecole Polytechnique Fédérale de Lausanne, Switzerland.

The last version of the Biogeme package, which is called PandasBiogeme, was released in the year 2018. This version was coded mainly in Python 3 and relies on the Pandas Data Analysis Library for data management. The most compute-intensive software parts were coded in C++, for the sake of efficiency. This package can be easily installed using the Python Package Index (PIP) package manager.

Biogeme allows to define the utility functions using traditional mathematical operations which makes the utility specification very straight forward. It also uses a dictionary to describe the availability conditions of each alternative, i.e. supports datasets where the choice set may vary across observations. Biogeme supports a wide range of choice models,

which are listed in Table 1. Biogeme also allows to calculate some economic indicators which are relevant in the context of discrete choice models, such as willingness to pay, value of time, market shares, revenues, and elasticities using a simulation technique which is called *stratified random sampling*. Finally, this package also allows to estimate models with latent variables, i.e. variables that are not directly observed but can be inferred from other observed variables.

2.2. PyLogit

PyLogit (Brathwaite & Walker, 2018) is a Python package intended to be used for performing maximum likelihood estimation of conditional logit models and other similar discrete choice models. This package can be installed using the PIP or Anaconda python package managers and the source codes are available in a GitHub repository.

PyLogit takes as input a long format dataset, which is composed of one row per individual per available alternative. The main advantage of the long format dataset is that it allows to calculate the utility functions $V_{ij} = x_{ij}\beta_i$ directly using a matrix dot product. PyLogit also supports different choice sets per individuals and model specifications admit alternative specific attributes, subset specific attributes or generic attributes.

An important difference between PyLogit and other packages, such as PythonBiogeme or mlogit, is the need to pre-calculate all variables to be used in the model. In other words, PyLogit does not perform variable creation, it only focuses on the model estimation using previously pre-computed variables. Hence, the user must define all the variables to be used in the model before executing the estimation procedure. Once the variables have been defined, the model can be specified and estimated. The specification of the utility functions is less intuitive than in PandasBiogeme because they are defined using a list for each variable indicating the alternatives which are affected by that variable. PyLogit also supports a wide range of choice models which are listed in Table 1.

2.3. ChoiceModels

ChoiceModels (Urban Data Science Toolkit, n.d.) is a Python package used for discrete choice modeling, as part of the Urban Data Science Toolkit, an open source portfolio developed and maintained by Berkeley Urban Analytics Lab. This library integrates several open source packages such as PyLogit. ChoiceModels is a kind of wrapper that allows to manage in an easy way the data before applying the choice model. In addition, this package also provides a tool for Monte Carlo simulation of choices given the probability distributions from fitted models. However, its main limitation is that it only integrates Multinomial Logit Models, losing the other powerful models which are provided by PyLogit.

2.4. mlogit

mlogit (Croissant, 2019) is a R package which allows to estimate random utility discrete choice models using maximum likelihood procedure. The package can be installed directly in R from the CRAN package repository.

In mlogit the utility functions are written using an extended model of the R *Formula* package, which allows to write these expressions using a notation which is similar to the one used in ChoiceModels with Patsy, making it very intuitive. It allows to define alternative specific variables with specific and generic coefficients and individual specific variables on the utility function. Nevertheless, one of the major limitations of this package is that it does not integrate support for varying choice sets over individuals.

2.5. Other alternatives

There are also other alternatives for discrete choice modeling. However, they have not been analysed in detail because they are not open source, free to use or widely popular between practitioners. Kenneth E. Train personal webpage contains several Matlab, R and Gauss codes for estimating Mixed Logit models. It should be highlighted the software piece called *Mixed Logit Estimation by Hierarchical Bayes* which was coded in Matlab to elaborate Chapter 12 of (Train, 2003). Finally, there are also some proprietary and commercial packages which can be used for discrete choice model estimation problems, such as nlogit, Stata, SPSS, etc.

2.6. Comparative of the different packages

To summarise the different software packages, Table 1 compares the discrete choice models which can be estimated by each of the different software packages and Table 2 summarises the key features of each alternative.

Supported models \ Package	Biogeme	PyLogit	ChoiceModels	mlogit
Multinomial Logit	✓	✓	✓	✓
Probit	✓ (only binary)	✗	✗	✓
Nested Logit	✓	✓	✗	✓
Cross-Nested Logit	✓	✗	✗	✗
Multivariate Extreme Value	✓	✗	✗	✗
Models with nonlinear utility functions	✓	✗	✗	✗
Models designed for panel data	✓	✗	✗	✗
Heteroscedastic models	✓	✗	✗	✗
Multinomial Clog-log	✗	✓	✗	✗
Multinomial Scobit	✗	✓	✗	✗
Multinomial Uneven Logit	✗	✓	✗	✗
Multinomial Asymmetric Logit	✗	✓	✗	✗
Mixed Logit	✓	✓	✗	✗

Table 1 – Discrete choice models that can be estimated by each package

Package	Programming Language	Active?	Open Source?	Data preprocessing	Different choice sets per individual?	Allows to calculate Indicators?
Biogeme	Python 3 (implemented in C++)	Yes	Yes, but unknown license	Yes, using Pandas package and internal methods	Yes	Yes, willingness to pay, value of time, market shares, revenues, and elasticities
PyLogit	Python 2 and Python 3	Yes	Yes, BSD-3-Clause license	Yes, using Pandas package. Datasets must be converted to long format.	Yes	No
ChoiceModels	Python 2 and Python 3	Yes	Yes, BSD-3-Clause license	Yes, using Pandas package	Yes (only with PyLogit equation format)	No
mlogit	R	Yes	Yes, GPLv2	Yes, datasets must be converted to long format.	No	No

Table 2 – Comparative of the discrete choice model packages

3. PyKernelLogit PACKAGE

The PyLogit package provides a good set of characteristics, integrates a variety of discrete models and its source code are available with an open-source license, therefore, it has been decided to extend this package to incorporate KLR models, which has led to the creation of the PyKernelLogit Python package. Figure 1 summarizes the main functionalities and dataflow necessary to estimate MNL and KLR models on PyKernelLogit. All the necessary functionalities for the estimation of KLR models have been integrated over the previously existing functionalities of PyLogit for the estimation of MNL models. In the following lines, the functionalities involved in the specification and estimation of a KLR model are described.

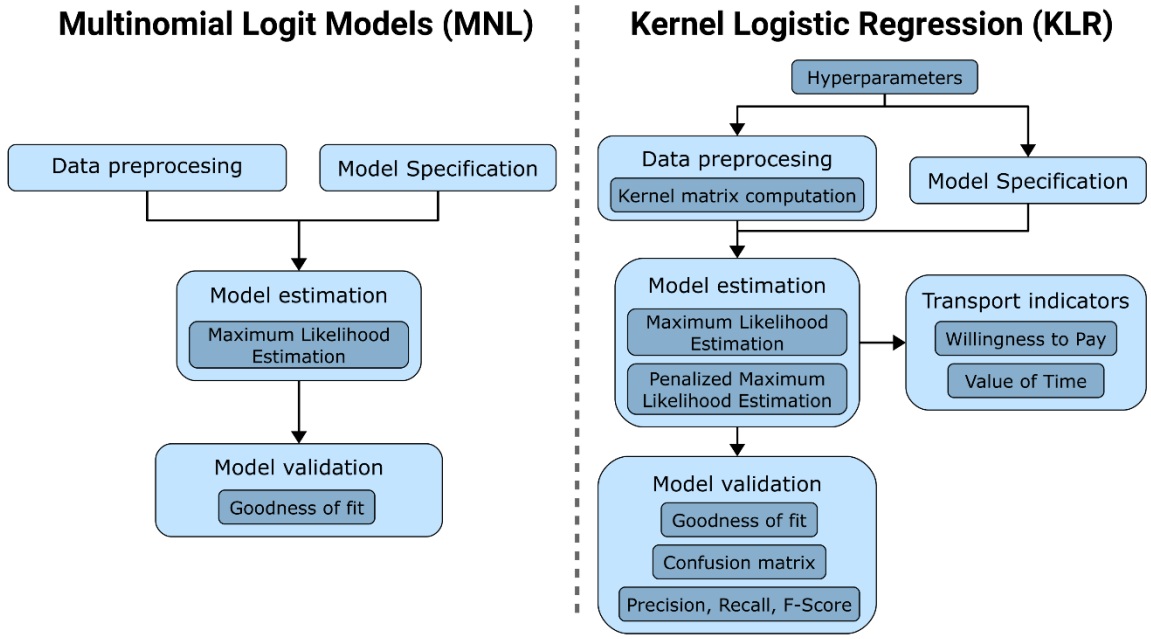

Fig. 1 – Comparison of MNL and KLR models on PyKernelLogit

3.1. Specification of the Kernel Logistic Regression Model

The KLR model builds some *latent functions*, $\mathrm{V_i}(\mathbf{x})$ for all the alternatives i in the choice set C, which are equivalent to the systematic part of the utility functions in RUM, but in KLR they are considered as black box functions. The main problem is to find for each alternative i, its latent function $\mathrm{V_i} : \boldsymbol{X} \mapsto \mathbf{R}$. KLR search for functions $\mathrm{V_i}(\mathbf{x})$ within function spaces which are named *Reproducing Kernel Hilbert Spaces* (RKHS). The RKHS space is a vector space which is univocally generated by the so-called *kernel function* $\mathrm{k}(\mathbf{x}, \mathbf{x}')$, and its associated RKHS space is denoted by $\mathcal{H}_\mathrm{k}$. The family of functions $\{\mathrm{k}(\mathbf{x}, \mathbf{x}')\}_{\mathrm{x}' \in \mathcal{X}}$ constitutes a basis of the vector space. Any element from $\mathcal{H}_k$ can be represented as a linear combination of the basis elements, in particular, from $\mathrm{V_i}(\mathbf{x}) \in \mathcal{H}_\mathrm{k}$, the expression of the utilities is given by

$$\mathrm{V_i}(\mathbf{x}, \alpha_\mathrm{i}) = \sum_{\mathrm{n}=1}^{\mathrm{N}} \alpha_{\mathrm{in}} \mathrm{k_i}(\mathbf{x}_\mathrm{n}, \mathbf{x}). \tag{1}$$

KLR provides estimates of the class probabilities based on these utility functions $\mathrm{V_i}(\mathbf{x})$

$$P(i|V, C) = \frac{\exp(V_i(x))}{1+\sum_{i=1}^{I-1} \exp(V_i(x))}. \tag{2}$$

It should be noted in Eq. (1) that the non-parametric utilities defined by the KLR model are linear in their parameters. In the MNL models the deterministic part of the utility $\mathrm{V_{in}}$ can be defined using the linear function $\mathrm{V_i}(\mathrm{x_{in}}, \beta_\mathrm{i}) = \beta_\mathrm{i}^\mathrm{T} \mathrm{x_{in}}$. Therefore, the methods defined in PyLogit to estimate a MNL model can be also used to estimate the KLR model, but the vector $\mathbf{x}$ should be substituted by the kernel function $\mathrm{k}(\mathbf{x}, \mathbf{x}')$. Hence, instead of estimating the $\boldsymbol{\beta}$ vector of parameters, the program is going to estimate $\boldsymbol{\alpha}$. But first, it is necessary to pre-compute the Gram matrix (or kernel matrix) $\mathbf{K}^{(\mathrm{i})}$, which is defined as

$$\mathbf{K}^{(\mathrm{i})}_{\mathrm{n}',\mathrm{n}} = \mathrm{k_i}(\mathbf{x}_{\mathrm{in}'}, \mathbf{x}_{\mathrm{in}}). \tag{3}$$

Intuitively, the Kernel matrix represents how similar is the decision-maker n' with respect to the decision-maker n for the alternative i. PyKernelLogit allows to compute the kernel matrix for any input dataset. The software package allows to specify many different types of kernel functions, such as the Radial-basis function, the Rational Quadratic kernel, the Matern Kernel, etc. It also allows to specify compound kernels, which are kernel functions composed of a set of other kernels. Figure 2 shows a sample kernel matrix computed using a Radial-basis function for a dataset with 453 observations. PyKernelLogit provides the possibility to specify a reference dataset $\mathbf{Z}$ to be used when computing the kernel matrix. If no reference matrix is provided, then the kernel matrix would be a square matrix which contains the similarity between any two observations from the dataset. Nevertheless, as this approach is computationally heavy, sometimes it is required to use a resampling method to select a subset of the observations as the reference matrix $\mathbf{Z}$ or to generate this reference matrix artificially, for example using a grid over the domain of the data. If $\mathbf{Z}$ is specified, then Eq. (3) becomes

$$\mathbf{K}^{(\mathrm{i})}_{\mathrm{n}',\mathrm{n}} = \mathrm{k_i}(\mathbf{z}_{\mathrm{in}'}, \mathbf{x}_{\mathrm{in}}). \tag{4}$$

	K_i_1	K_i_2	K_i_3	K_i_4	K_i_5	...	K_i_449	K_i_450	K_i_451	K_i_452	K_i_453
0	1.000000	0.859819	0.884954	0.993861	0.976290	...	0.757557	0.929142	0.980138	0.673098	0.725472
1	0.859819	1.000000	0.992509	0.888991	0.862176	...	0.969062	0.985683	0.929744	0.893956	0.927796
2	0.884954	0.992509	1.000000	0.902722	0.904262	...	0.933217	0.986216	0.954549	0.837235	0.878386
3	0.993861	0.888991	0.902722	1.000000	0.956273	...	0.805321	0.951511	0.979215	0.732930	0.782034
4	0.976290	0.862176	0.904262	0.956273	1.000000	...	0.736968	0.918418	0.984070	0.626018	0.681269
...	...	...	...	...	...	...	...	...	...	...	...
448	0.757557	0.969062	0.933217	0.805321	0.736968	...	1.000000	0.932803	0.831198	0.972509	0.986445
449	0.929142	0.985683	0.986216	0.951511	0.918418	...	0.932803	1.000000	0.972541	0.856337	0.897445
450	0.980138	0.929744	0.954549	0.979215	0.984070	...	0.831198	0.972541	1.000000	0.732485	0.784137
451	0.673098	0.893956	0.837235	0.732930	0.626018	...	0.972509	0.856337	0.732485	1.000000	0.995798
452	0.725472	0.927796	0.878386	0.782034	0.681269	...	0.986445	0.897445	0.784137	0.995798	1.000000

Fig. 2 – Kernel matrix $\mathbf{K}^{(i)}$

Once the kernel matrix has been created, PyKernelLogit allows to automatically define the kernel model specification, which was presented in Eq. (1), using the PyLogit syntax. The user can select in the model specification if the same vector of parameters to estimate $\boldsymbol{\alpha}$ is going to be used for all the alternatives or, otherwise, if one vector of parameters $\boldsymbol{\alpha}_i$ is going to be used for each alternative i. Once the model has been specified, the choice model can be created an estimated.

3.2. Estimation of the Kernel Logistic Regression Model

PyKernelLogit extends the estimation capabilities of PyLogit by implementing a Penalised Maximum Likelihood Estimation (PMLE) procedure. In order to estimate the KLR model the following optimization problem must be solved

$$\underset{\boldsymbol{\alpha}}{\text{Minimize}} \sum_{n=1}^{N} \sum_{i=1}^{I} L\left(y_{in}, V_i(\mathbf{x}_{in}, \boldsymbol{\alpha}_i)\right) + \lambda \sum_{i=1}^{I} R_i(\boldsymbol{\alpha}_i), \tag{5}$$

where $\mathrm{L}(\cdot)$ is a loss function that measures the discrepancies between the observed and the predicted classifications and λ is a regularization parameter in charge of controlling the trade-off between the goodness of fit and the complexity of the model. Finally, $\mathrm{R_i}(\cdot)$ is the penalty function of the α_i parameters. PyKernelLogit integrates two of the most relevant penalty functions $\mathrm{R_i}(\cdot)$ proposed in the literature, which are the LASSO and the Ridge regression (Castro, 2013).

The estimation of the KLR model is computationally heavy if the number of observations is high enough (Ouyed & Allili, 2018; Zhu & Hastie, 2005). The implementation of the PMLE procedure reduces significantly the estimation time. Nevertheless, PyKernelLogit allows to use different optimisation algorithms, so it has been tested the use of tree canonical methods for solving the PMLE problem defined in Eq. (5). The selected methods are:

- Newton-CG. This method, also known as the truncated Newton method, uses the nonlinear conjugate gradient algorithm by Polak and Ribiere to compute the search direction (Nocedal & Wright, 2006). This method is suitable for large-scale problems.

- BFGS. This method uses the quasi-Newton method of Broyden, Fletcher, Goldfarb, and Shanno (Nocedal & Wright, 2006). It uses the first derivatives only. BFGS has proven good performance even for non-smooth optimizations.
- L-BFGS-B. It is an optimization method based on quasi-Newton methods that approximates the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) algorithms using a limited amount of computer memory (Byrd et al., 1995). It is a popular algorithm for parameter estimation in ML, because it is optimized for functions with a large number of parameters or with a high degree of complexity.

A KLR model has been estimated for all the combinations of penalty functions and optimisation algorithms previously described. The estimation was executed using a 5-Fold cross-validation procedure on a choice model problem. Table 3 shows the mean value for the CPU time, the objective function (the log-likelihood) and the precision value obtained using the different combinations of penalty functions and optimization algorithms. The best recommendation base on the results obtained is to use the L-BFGS-B algorithm and the Ridge regression model. Nevertheless, if the number of variables to estimate is not considerably large, the BFGS algorithm can also be applied.

Estimation model	Algorithm	CPU time (seconds)	Objective function	Precision (%)
Maximum likelihood	Newton-CG	8.63	-265.5857	66.65
	BFGS	5.65	-265.9253	65.54
	L-BFGS-B	0.80	-273.1642	66.20
LASSO	Newton-CG	0.59	-342.8578	67.09
	BFGS	3.55	-307.6523	67.53
	L-BFGS-B	0.46	-312.3938	67.31
Ridge	Newton-CG	6.32	-291.4132	67.31
	BFGS	0.33	-291.4132	67.31
	L-BFGS-B	0.07	-291.4136	67.31

Table 3 – Comparison of different penalty functions and optimization algorithms

3.3. Model validation and assessment

PyKernelLogit allow to obtain several metrics for assessing whether a model is good enough for a specific application. One of them is the goodness of fit of the model (Train, 2003), which is a statistic that describes how well a model, with its estimated parameters, performs compared with a model in which all its parameters are zero (which is usually equivalent to having no model at all). Once the model is estimated, PyKernelLogit provides the likelihood ratio index (also called McFadden R square), which is defined as

$$\rho^2 = 1 - \frac{LL(\hat{\beta})}{LL(0)}. \tag{6}$$

However, goodness of fit metrics cannot asses how the model predicts on new data points, for which it has not yet seen the true value of the dependent variable (the alternative chosen). For this reason, PyKernelLogit integrates some other model validation techniques. One of these techniques is the confusion matrix (Géron, 2019). Once the model has been

estimated using training data, the confusion matrix can be computed over the test data for which the true values are known. Each column of the confusion matrix represents the instances in a predicted class while each row represents the instances in an actual class. Using this matrix, it easy to identify if the model is confusing two classes.

The confusion matrix gives useful information, although sometimes it is preferred a more concise metric. Another interesting way of assessing the performance of a model is to look the accuracy of the positive predictions, which is called *precision*. This metric has an important drawback because if the model makes a single positive prediction and it is correct, then the precision will be 100%. For that reason, the precision metric is commonly used along with another metric called *recall*, or *sensitivity*. The recall is the ratio of positive instances that are correctly detected by the classifier. Finally, there is a third metric called *F-score*, which combines the precision and the recall results to compute the score. The *F-beta score* is the weighted harmonic mean of precision and recall, reaching its optimal value at 1 and its worst value at 0. The β parameter determines the weight of recall in the combined score. When $\beta < 1$, the F-score lends more weight to precision, while $\beta > 1$ favors recall. The evaluation of the precision, recall and F-Score metrics requires to compute for each class i the number of True Positives (TP), False Negatives (FN), True Negatives (TN) and False Positives (FP) instances.

The previous metrics are defined por binary classification problems, i.e. only two alternatives are considered. Nevertheless, when there are more than two alternatives in the model, the quality of the overall classification is usually assessed using two techniques (Sokolova & Lapalme, 2009):

- Marco-averaging. Each measure is the average of the individual measures calculated for all the alternatives.
- Micro-averaging. Each measure is obtained computing firstly the cumulative TP, FN, TN and FP for all the alternatives.

PyKernelLogit allows to obtain the precision, recall and F-score metrics for micro- and macro-average methods. Table 4 shows how the previous metrics have been implemented in the software package.

	Precision	**Recall**	**F-Score**
Micro-average (m)	$\frac{\sum_{i=1}^{I} TP_i}{\sum_{i=1}^{I}(TP_i+FP_i)}$	$\frac{\sum_{i=1}^{I} TP_i}{\sum_{i=1}^{I}(TP_i+FN_i)}$	$\frac{(\beta^2+1)\,\text{Precision}_m\,\text{Recall}_m}{\beta^2\,\text{Precision}_m+\text{Recall}_m}$
Macro-average (M)	$\frac{\sum_{i=1}^{I} \frac{TP_i}{TP_i+FP_i}}{I}$	$\frac{\sum_{i=1}^{I} \frac{TP_i}{TP_i+FN_i}}{I}$	$\frac{(\beta^2+1)\,\text{Precision}_M\,\text{Recall}_M}{\beta^2\,\text{Precision}_M+\text{Recall}_M}$

Table 4 – Precision, recall and F-score metrics for multi-class classification

3.4. Computation of economic indicators

Once a discrete choice model has been estimated using the PyKernelLogit package, it can be useful for the analyst to extract some indicators from this model. There are many indicators that are particularly relevant in the context of discrete choice models, such as, Willingness to Pay (WTP), Value of Time (VOT), elasticities, consumer surplus, market shares and revenues. Currently, PyKernelLogit allows the estimation of the WTP and VOT indicators.

The WTP allows to analyse the trade-off between any variable from the model and money. Let consider that c_{in} is the cost of the alternative i for a decision-maker n, and x_{in} is the value of another model variable. Now, let $V_{in}(c_{in}, x_{in})$ be the value of the utility function that is associated to an alternative i for a decision-maker n. Then, the WTP can be computed as

$$\text{WTP} = -\frac{\partial V_{in}/\partial x_{in}}{\partial V_{in}/\partial c_{in}}. \tag{7}$$

The VOT indicator is closely related with the previous one, as it reflects the price that a traveller is willing to pay to decrease the travel time in one unit. Therefore, VOT can be computed as

$$WTP = \frac{\partial V_{in}/\partial t_{in}}{\partial V_{in}/\partial c_{in}}. \tag{8}$$

The WTP and VOT indicators can be computed using a closed form expression for linear MNL models. Nonetheless, in KLR Eq. (7) and (8) does not have a closed form expression for some kernel functions and the partial derivatives must be approximated numerically. PyKernelLogit allows to calculate the WTP and VOT indicators for KLR models easily. It implements the Newton's difference quotient method to compute the partial derivatives of Eq. (7) and (8). One advantage of PyKernelLogit is the fact that it allows to compute the WTP an VOT indicators for certain values of the attributes.

3.5. PyKernelLogit package installation

PyKernelLogit source codes are available under an open-source license in a Github repository (https://github.com/JoseAngelMartinB/PyKernelLogit). In addition, a compiled version of PyKernelLogit is available for its installation on the Python Package Index (PIP) at https://pypi.org/project/pykernellogit/. The package can be installed on a computer with any operating system, provided that a Python distribution is available (preferably Python 3.7 or later), using the command:

```
$  pip install PyKernelLogit
```

Interested authors can find more detailed information on (Martín Baos, 2019).

4. CONCLUSIONS

In this work, an open source software package called PyKernelLogit has been developed for Python programming language. This software package allows to implement several discrete choice models, to apply several model validation techniques and to obtain a set of economic indicators. The package extends the capabilities of the original PyLogit package, which was able of estimating MNL models and its derivatives, and integrates the KLR models. The interest on aggregating the KLR model to the software package arises from two reasons. Firstly, KLR models are machine learning models which relieves the analyst from specifying a functional expression of the utilities beforehand through the use of the kernel functions. Moreover, KLR models have better predictive capabilities than the MNL models for non-lineal effects.

Our future work will be focus on extending the KLR model utility specifications to other discrete choice models such as Nested Logit models. In addition, we will integrate into PyKernelLogit estimation procedure the Tikhonov regularization procedure. Finally, new features will be implemented to retrieve other economic indicators such consumer surplus, market shares, elasticities, revenues, etc.

ACKNOWLEDGEMENTS

The research of Martín-Baos has been supported by the FPU Predoctoral Program of the Spanish Ministry of Science, Innovation and Universities (Grant Ref. FPU18/00802); and the research of García-Ródenas was supported by Project TRA2016-76914-C3-2-P of the Spanish Ministry of Science and Innovation, co-funded by the European Regional Development Fund.

REFERENCES

Ben-Akiva, M., & Bierlaire, M. (1999). Discrete Choice Methods and their Applications to Short Term Travel Decisions. In R. W. Hall (Ed.), Handbook of Transportation Science (pp. 5–33). Springer US.

Ben-Akiva, M. E., Lerman, S. R., & Lerman, S. R. (1985). Discrete choice analysis: theory and application to travel demand (Vol. 9). MIT press.

Bierlaire, M. (2018). PandasBiogeme: a short introduction. http://transp-or.epfl.ch/documents/technicalReports/Bier18.pdf

Brathwaite, T., & Walker, J. L. (2018). Asymmetric, closed-form, finite-parameter models of multinomial choice. Journal of Choice Modelling, 29, 78–112.

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A Limited Memory Algorithm for Bound Constrained Optimization. SIAM Journal on Scientific Computing, 16(5), 1190–1208.

Castro, S. (2013). Estimación y selección de variables en grandes dimensiones. Regresión Ridge, GNN, Lasso, Elastic Net, SCAD. Facultad de Ingeniería. Universidad de La República.

Croissant, Y. (2019). Package “mlogit.” https://doi.org/10.1017/CBO9780511805271

Géron, A. (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.).

Martín Baos, J. Á. (2019). Design and implementation of a software library for the estimation and analysis of non-parametric discrete choice models. Application to transport planning. Universidad de Castilla-La Mancha. http://hdl.handle.net/10578/23090

McFadden, D. L. (1978). Modelling the Choice of Residential Location. In A. K. et al. (Ed.), Spatial Interaction Theory and Residential Location (pp. 75–96). North Holland.

Nocedal, J., & Wright, S. J. (2006). Numerical Optimization. Springer New York.

Ouyed, O., & Allili, M. S. (2018). Feature weighting for multinomial kernel logistic regression and application to action recognition. Neurocomputing, 275, 1752–1768.

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing and Management, 45(4), 427–437.

Train, K. E. (2003). Discrete choice methods with simulation. In Discrete Choice Methods with Simulation (Vol. 9780521816).

Urban Data Science Toolkit. (n.d.). ChoiceModels documentation. Retrieved July 1, 2019, from https://choicemodels.readthedocs.io

Zhu, J., & Hastie, T. (2005). Kernel logistic regression and the import vector machine. Journal of Computational and Graphical Statistics, 14(1), 185–205.