PyPLT: Python Preference Learning Toolbox

Elizabeth Camilleri
Institute of Digital Games
University of Malta
Msida, Malta
elizabeth.camilleri.12@um.edu.mt

Georgios N. Yannakakis
Institute of Digital Games
University of Malta
Msida, Malta
georgios.yannakakis@um.edu.mt

David Melhart
Institute of Digital Games
University of Malta
Msida, Malta
david.melhart@um.edu.mt

Antonios Liapis
Institute of Digital Games
University of Malta
Msida, Malta
antonios.liapis@um.edu.mt

***Abstract*—There is growing evidence suggesting that subjective values such as emotions are intrinsically relative and that an ordinal approach is beneficial to their annotation and analysis. Ordinal data processing yields more reliable, valid and general predictive models, and preference learning algorithms have shown a strong advantage in deriving computational models from such data. To enable the extensive use of ordinal data processing and preference learning, this paper introduces the Python Preference Learning Toolbox. The toolbox is open source, features popular preference learning algorithms and methods, and is designed to be accessible to a wide audience of researchers and practitioners. The toolbox is evaluated with regards to both the accuracy of its predictive models across two affective datasets and its usability via a user study. Our key findings suggest that the implemented algorithms yield accurate models of affect while its graphical user interface is suitable for both novice and experienced users.**

***Index Terms*—ordinal annotation, affect models, preference learning, open source, software, tools, Python**

I. Introduction

The question of how to best label subjective constructs such as emotions and process the obtained labels is central to disciplines involving demonstration data from humans, including affective computing and human computer interaction. A recent trend in the annotation and processing of such data builds on extensive empirical evidence across several domains and reveals the benefits of labeling and treating data of a subjective nature as ordinal [1], [2]. Such a potential paradigm shift towards ordinal data processing calls for methods and tools which are accessible to researchers of these disciplines, enabling the widespread use of ordinal (preference) machine learning algorithms and further advancing the state of the art in human-based annotation and modeling. Meanwhile, existing preference learning tools [3] are limited by their narrow target audience, and are outdated given the rapid expansion of modern deep learning frameworks and algorithms.

To address the current lack of tools for processing ordinal labels, this paper introduces the Python Preference Learning Toolbox (PyPLT). The tool is based on the existing Preference Learning Toolbox [3], introduced in 2015, but is written in a more suitable and relevant language and contains additional and improved features. PyPLT is designed to serve as a Python-based, unified and accessible application for preference learning, offering a user-friendly graphical interface (for beginner and advanced users) in addition to an application programming interface (for machine learning developers). The toolbox is available at http://plt.institutedigitalgames.com. Currently the toolbox gives users access to a number of ordinal data processing methods and popular preference learning algorithms based on support vector machines, artificial neural networks, and deep learning. Importantly, the toolbox is open source so that algorithms and other features may continue to be added. The tool is relevant for research in ordinal label processing and preference learning and is targeted to researchers and practitioners in affective computing, user modeling, preference handling, and human computer interaction at large.

The structure of the paper is as follows. Section II surveys related tools for preference learning, while Section III describes the various modes and components of PyPLT in detail. Section IV tests the capacity of PyPLT in deriving accurate predictive models across two affective datasets (Section IV-A) and demonstrates its usability via a user study (Section IV-B). The paper concludes in Section V with a discussion on the potential of PyPLT for further development through the addition of algorithms and other data processing methods.

II. Related Tools

Preference learning (PL) is the area of supervised learning dedicated to the processing and analysis of ordinal labels [4], [5]. The approach has been increasingly popular within affective computing given the availability of ordinal-based affective datasets; see [2] for a detailed review. However, current software tools for preference learning are limited to individual algorithms, such as SVM^{rank}[1], a software implementation of an algorithm for training Ranking SVMs [6], [7]. While the LPCforSOS framework[2] and extensions such as WEKA-LR[3], MEKA [8] and MULAN [9] allow for a wide variety of classification methods offered by the WEKA toolkit [10] to be trained on ordinal labels, they do not cater for partially ordered non-absolute data. The Lemur Project's RankLib library[4] also offers quite an extensive range of learning-to-rank algorithms; however, it suffers from the same limitation as it only allows for models to be trained on data in the form of ratings. On the other hand, the jPL framework [11] caters for various kinds of PL problems. As with most of the aforementioned tools, jPL lacks a graphical interface which makes it quite

[1] http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html
[2] https://sourceforge.net/projects/lpcforsos/
[3] https://cs.uni-paderborn.de/?id=63906
[4] https://sourceforge.net/p/lemur/wiki/RankLib/

978-1-7281-3888-6/19/$31.00

inaccessible for many potential users who lack knowledge of and experience with machine learning and the particular tool. Furthermore, the tool focuses only on the modelling and evaluation steps of the PL process, overlooking important steps such as feature extraction, normalization, and feature subset selection. The WEKA extensions, RankLib library, and jPL, as well as the earlier Java implementation of the Preference Learning Toolbox [3], are also limited by the underlying programming language (Java) which is vastly less suitable for data science compared to languages such as Python and R. The choice of language makes the tools less likely to be integrated with existing projects or further extended, as current advances in machine learning more broadly use the Python language.

Beyond all aforementioned tools, PyPLT introduces a Python-based tool which is open source and accessible, integrates widely used machine/deep learning frameworks such as TensorFlow and Keras, and features popular PL algorithms and ordinal data processing methods under a unified framework.

III. The PyPLT Interface and Algorithms

The PyPLT toolbox may be used either as a software application via its Graphical User Interface (GUI) or as a library via its Application Programming Interface (API). Regardless of the interface used, PyPLT offers various methods for each step in the preference data modelling process. How the tool can be used in each step is described below.

A. Modes

The GUI of PyPLT allows the user to select between two modes of operation: *beginner* mode and *advanced* mode. The beginner mode, depicted in Fig. 1, simplifies the experiment setup process into 5 quick and easy steps: loading the data set, choosing whether or not to apply automatic feature extraction, choosing whether or not to apply feature selection, choosing a PL algorithm, and finally running the experiment. In the beginner mode, the algorithms' parameters are pre-set to default values of the external libraries used (*TensorFlow*, *scikit-learn*, *Keras*), or the earlier version of PLT [3]. The advanced mode involves the same 5 steps as the beginner mode, but each of the first four steps is encapsulated in its own detailed tab (as shown in part in Figs. 2-4). In the advanced mode, each tab contains a set of options or parameters through which the experiment setup may be fine-tuned by the user. Both modes provide a help dialog containing useful information to guide the user for each of the steps.

B. Dataset Loading and Parsing

As Fig. 1 shows, the first step in the preference data modelling process involves loading the dataset intended for preference learning. Data from any domain may be loaded into the tool in the form of a Comma-Separated Values (CSV) file structured for PyPLT[5]. A dataset needs to contain two elements: a set of objects (input) and the relation or order among them (output). In PyPLT, the dataset may be loaded in one of two formats: a *single file format* for problems where a *global order* of objects exists and a *dual file format* for problems where a *partial order* of objects exists. A global order of objects is a *rating* value given for each object in a dataset, whereas a partial order is a set of *pairwise preferences* given for a number of objects in the dataset. The GUI guides the user through the loading process, ensuring that the dataset is loaded correctly through various prompts. The interface also offers a range of parameters for parsing the file.

[5]Details on preparing a CSV file for PyPLT can be found at http://plt.institutedigitalgames.com/howto.php

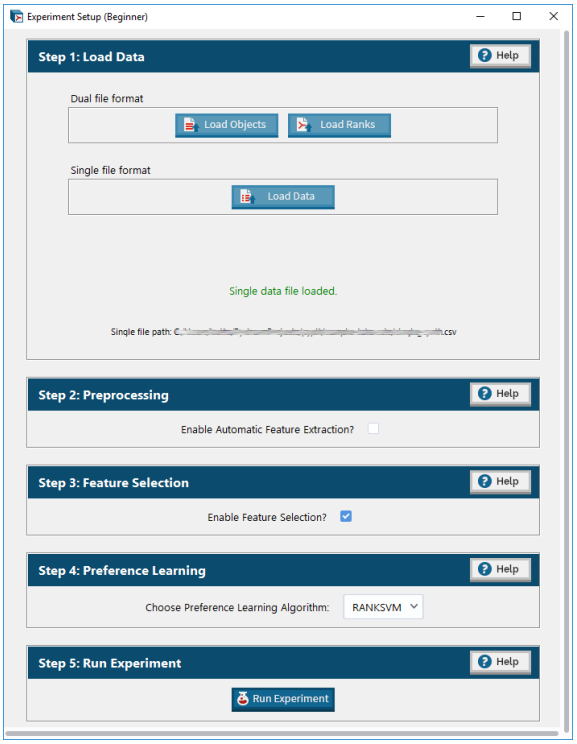

Fig. 1. Screenshot of the beginner mode which hides the more detailed options available in the advanced mode, offering a more simplified interface for setting up an experiment.

C. Data Pre-processing

In PyPLT, the loaded dataset can be pre-processed in several ways: via automatic feature extraction, manual feature selection, feature normalization, and dataset shuffling; see Fig. 2.

1) Automatic Feature Extraction: If the given dataset does not include predetermined features, PyPLT allows the user to apply automatic feature extraction. There are several feature extraction or compression algorithms that we have considered for inclusion in PyPLT, such as PCA variants and clustering methods. Given the widespread use of autoencoders for single or two dimensional signals in machine learning and affective computing, however, the initial version of the PyPLT toolbox features vanilla autoencoders to extract a user-specified number of features from the provided data. The autoencoder is a neural network consisting of two parts—the *encoder* and the *decoder*—and is considered a form of non-linear dimensionality reduction or data compression [12]. The encoder compresses the input data whereas the decoder decompresses

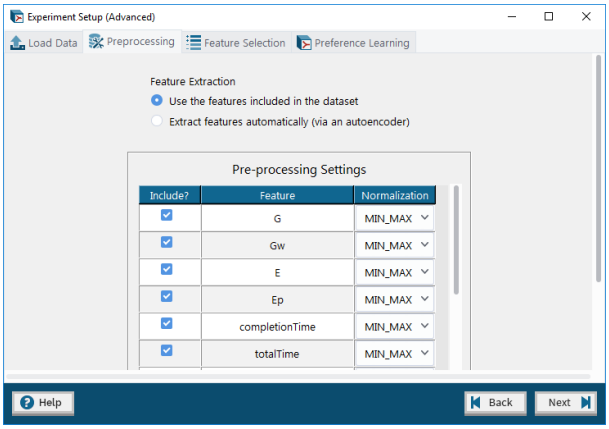

Fig. 2. Screenshot of the Preprocessing tab (advanced mode) with various options for preprocessing the data.

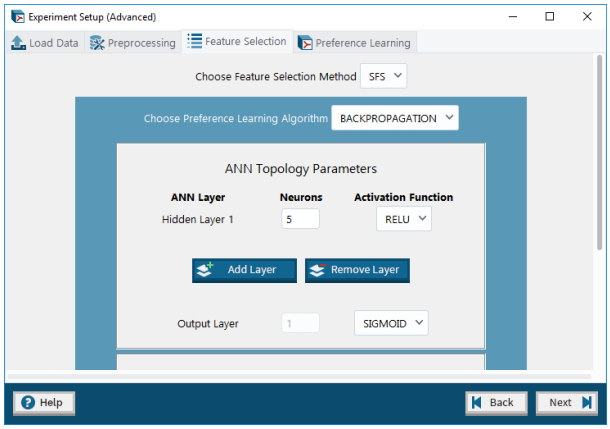

Fig. 3. Screenshot of the Feature Selection tab (advanced mode) where users may opt to have the toolbox automatically select a subset of the most predictive attributes in their data.

the compressed version of the data. The autoencoder is trained via backpropagation to create as accurate a reconstruction of the input at the output layer as possible. The layer in-between the encoder and the decoder (i.e., the code layer) stores the compressed (encoded) version of the input (i.e., the extracted features). The autoencoder is optimized using the Adam Optimizer [13] in the *TensorFlow*[6] library and its performance is determined via the mean squared error function. In the advanced mode and via the API, the topology of both the encoder and the decoder as well as the code size (the number of neurons in the code layer) and other parameters of the learning algorithm may be specified by the user.

2) Manual Feature Selection: Whether the dataset contains predetermined features or the features are extracted automatically, the advanced mode of PyPLT offers users the option to manually include only a particular subset of these features in the experiment while excluding others. This is facilitated through user-friendly checkboxes in the GUI.

3) Feature Normalization: The initial version of PyPLT offers two different methods for normalizing the features' values: *min-max* normalization transposes the values of the given feature to fit between a given range of values (from 0 to 1 by default), and *z-score* normalization which transforms the values of the given feature such that the average value of the feature is zero and the standard deviation is one. Both methods are used widely in data normalization, hence their inclusion to the initial version of PyPLT. In beginner mode, min-max normalization is applied to all features by default.

4) Data Shuffling: In advanced mode, one may also choose whether the dataset is shuffled prior to running any experiment via a simple checkbox. Data shuffling is particularly important for eliminating any order biasing in the data. If the dual file format is being used, it is the order of the ranks (pairwise preferences) that is randomized whereas if the single file format is being used, it is the order of the samples that is randomized. Moreover, randomization may be controlled via a random seed specified by the user, allowing for replicability.

[6]https://www.tensorflow.org/

D. Automatic Feature Selection

While users may opt to filter which features are used in the experiment manually (as described in Section III-C2), PyPLT is able to automatically find the most relevant subset of input features for preference models derived from the given data. Users may opt for this via the Feature Selection tab depicted in Fig. 3. The Sequential Forward Selection (SFS) [14] method is currently available to this end. In this hill-climbing algorithm, the selection procedure begins with an empty feature set and a new feature is added with each iteration in bottom-up fashion. The feature to be added is selected from the subset of remaining features such that the new feature set generates the maximum value of the performance function over all candidate features. The method terminates when an added feature yields equal or lower performance to the performance obtained without it. Performance is computed as the prediction accuracy of a model trained using that feature set as input. Any of the PL algorithms implemented in the toolbox (see Section III-E) may be used to train this model. If feature selection is enabled, the performance of each feature is displayed in the progress menu while the experiment is running, and the final experiment report (see Section III-G) lists which features were selected and the order in which they were selected.

E. Preference Learning

Once the data has been loaded and pre-processed, and a subset of particularly relevant features has been automatically selected (if applicable), a model may be inferred from the data via preference learning. The current version of PyPLT offers a number of PL algorithms for the user to choose from; in the advanced mode and via the API, various parameters for each algorithm may also be tuned (see Fig. 4). The three available algorithms are representative PL methods used widely in the literature, based on support vector machines and artificial neural networks. The implementation of these algorithms in PyPLT is based on existing code imported from popular machine learning libraries that are efficient and powerful; these

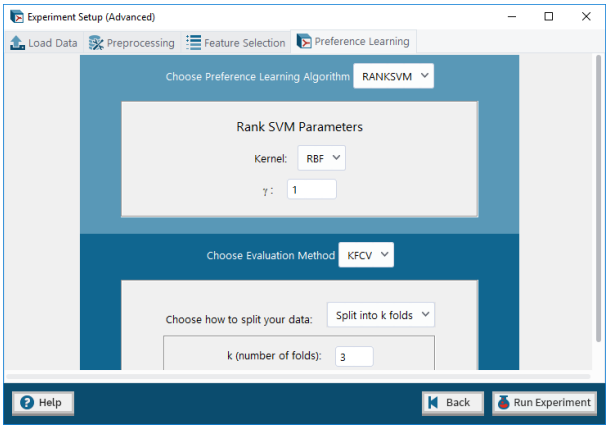

Fig. 4. Screenshot of the Preference Learning tab (advanced mode) which offers three different algorithms and evaluation methods.

supervised learning algorithms have been repurposed to handle ranks and preferences.

1) RankSVM: This algorithm is a rank-based version of the traditional Support Vector Machine (SVM) algorithm. Provided with data examples of annotated classes or categories as a form of output, an SVM uses a predefined kernel function to map the data instances onto geometric points in a high-dimensional space according to the input features that define them [15]. The RankSVM algorithm [6], [16] attempts to separate the data instances in the space via a hyperplane, in order to match the given preference information. Unseen instances may then be mapped to the space according to their features and an output is produced based on which sub-space they correspond to, according to the hyperplane. In PyPLT, the algorithm was implemented using the *scikit-learn*[7] library. The kernel function may be specified by the user in the advanced mode or via the API.

2) Feedforward ANN Backpropagation: This is a gradient-descent algorithm that iteratively adjusts the weights of an artificial neural network (ANN) model in order to minimize the error between the predicted network output and the desired pairwise preferences over the given data instances. The error is calculated using the Rank Margin function. Given a pair of data samples A and B, with $A \succ B$ (A preferred over B) the function yields 0 if the network output for A (i.e., f_A) is more than one unit larger than the network output for B (i.e., f_B) and $1 - (f_A - f_B)$ otherwise. The total error is averaged over the complete set of pairwise preferences in the training set. The ANN training continues until the error is below a certain threshold or the specified number of epochs (iterations) is reached [17]. The topology of the ANN, the number of epochs, the termination threshold, and the learning rate may be specified by the user in advanced mode or via the API. In PyPLT, the algorithm was implemented using the *TensorFlow* library.

3) RankNet: RankNet [18], as implemented in PyPLT, is an extension of the ANN backpropagation algorithm that uses a probabilistic cost function to handle ordered pairs of data. As with backpropagation, the algorithm optimizes the error function by adjusting the weights of an ANN model at each iteration until the specified number of epochs is reached. However, RankNet uses the binary cross-entropy function [18] as the error function. As with the other algorithms, the network topology and algorithm parameters may be specified by the user in the advanced mode or via the API. In PyPLT, the algorithm was implemented using the *Keras*[8] library.

F. Model Evaluation

Models may be trained using the complete dataset (no validation); in this case, performance is assessed as the percentage of correctly classified training pairs. It is a common machine learning practice, however, to test the generality of the results using an evaluation method. In the advanced mode and via the API, PyPLT allows the user to train without validation or choose between the two popular methods of evaluation described below. In the beginner mode, *Holdout* is used by default.

1) Holdout: This method trains the model on a given proportion of the dataset (e.g., 70%) and then tests the model on the remaining proportion of the dataset (e.g., 30%). The proportions can be specified in the advanced mode or via the API; in the beginner mode, the training proportion is fixed to 70%.

2) K-Fold Cross Validation: This method trains a number of models using different folds (train-test partitions) of the data and considers the average accuracy of the models across these folds. The dataset may be split into folds either *automatically* by specifying the k number of folds or *manually* by uploading a file that maps each sample in the dataset to a fold.

G. Visualisation of Results and Trained Models

When running an experiment via the GUI, a progress bar and training report are displayed during execution. Upon completion, a report with the experiment's details and its results is shown to the user. Users can also save the experiment report and the model (for a given data fold, if applicable) to a human-readable CSV file, either via the GUI or via the API.

IV. Benchmark Tests

This section, evaluates the predictive capacity of PyPLT's algorithms on two affective datasets (Section IV-A) and describes a user study on PyPLT's usability (see Section IV-B).

A. Efficiency and Modelling Accuracy

In order to showcase the efficiency and accuracy of PyPLT in deriving computational models from data, we test its algorithms on two affective datasets: the *Sonancia* audio clip dataset [19] and the DEAP dataset [20]. The Sonancia dataset contains crowdsourced pairwise preference annotations of arousal, tension and valence with respect to audio clips which are described by 387 features extracted from the openSMILE tool [21]. The DEAP dataset contains rating-based annotations

[7]http://scikit-learn.org/

[8]https://keras.io/

TABLE I
HIGHEST CROSS-VALIDATION ACCURACY AND CORRESPONDING COMPUTATION TIME (IN BRACKETS) ACHIEVED USING EACH AVAILABLE ALGORITHM ON THE SONANCIA AND DEAP DATASETS.

	Sonancia	DEAP
RankSVM	70.5% (1.1 sec)	94.5% (123.7 sec)
ANN backpropagation	69.6% (2.7 sec)	55.1% (30.7 sec)
RankNet	72.7% (2.1 sec)	67.4% (75.1 sec)

of music videos in terms of arousal, valence, like/dislike, dominance and familiarity. For this paper, we use a *second-order* data processing method [1], [2] to derive ordinal labels from ratings and a set of 7 physiological features (4 features of heart rate and 3 features of skin conductance) describing the annotators' reaction to the videos—as derived in a recent study [22]. All experiments in this paper consider the arousal dimension of the annotations for both datasets: 671 datapoints for Sonancia and 17,653 datapoints for DEAP. The baseline accuracy for Sonancia (assuming the most common rank is always chosen) is 63.7%, while for DEAP it is 50%.

All of the experiments ran on a 64-bit Windows computer with 16GB of memory, a Core i7-8700 CPU at 3.20GHz, and an NVIDIA GeForce GTX 1060 (6GB) graphics card. We compare computation times as well as model performance when applying the three different algorithms in PyPLT over these two datasets; 3-fold cross validation was used to evaluate the models. Reported results are based on the best hyper-parameter set per algorithm and task, after extensive exploration. Furthermore, for each setup, the ANN-based algorithms ran 10 times due to their non-deterministic nature and the average performance was considered. Table I reports the best performances achieved together with the computation time it took to execute the corresponding experiment.

Table I shows that PyPLT was able to construct fairly accurate models on both datasets, with one exception. While all three algorithms performed similarly on the Sonancia dataset (with cross-validation accuracies hovering around the 70% mark), the algorithms showed more variation in modelling capacity on the DEAP dataset. The most accurate models for DEAP were achieved using RankSVM (with a cross-validation accuracy of almost 95%) whereas ANN backpropagation yielded the least accurate models. This result highlights the importance of exploring various setups, algorithms and parameters when deriving models from real data—a process which is facilitated by the various options in the tool's GUI.

Table I also shows that the computation speed of PyPLT experiments is generally quite fast, although it is clearly affected by the dataset size: the execution of every Sonancia-based experiment lasted under 3 seconds, while for the DEAP-based experiments execution time reached 2 minutes. Interestingly, RankSVM was much faster than other algorithms in Sonancia, while the opposite is true in the larger DEAP dataset.

B. Usability

To test the usability of the interface, users of varying levels of experience with the tool were timed as they carried out a predefined set of tasks using the toolbox in both its modes—beginner and advanced—on a given dataset (i.e., the Sonancia dataset used in IV-A). The set of tasks assigned for each mode were designed to be incremental in terms of difficulty, and built on previous tasks. Six tasks were given to testers in total. The first three tasks (1-3) used the beginner interface, and included running a different PL algorithm and adding more complexity in each task (in task 2 turning on automatic feature extraction; in task 3 also using SFS). Tasks 4-6 used the advanced interface, and required more customization but followed the same iterative complexity as tasks 1-3 (i.e., adding automatic feature extraction in task 5 and SFS in task 6).

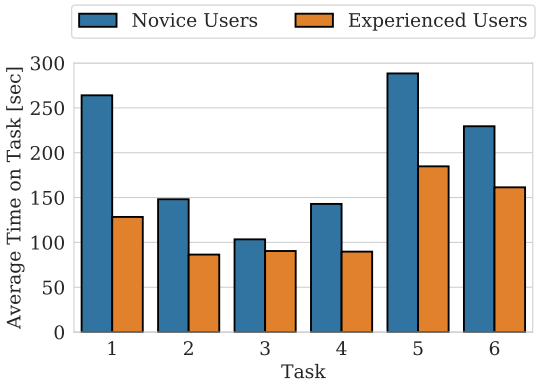

Fig. 5. Average time on task for novice and experienced participants.

After completing the assigned tasks, the participant answered a brief usability questionnaire: a binary adaptation of the Post-Study System Usability Questionnaire (PSSUQ) [23] in which each of its 19 Likert items were instead presented as a question with three possible answers: "Yes", "No", and "I don't know/Not relevant". For every case, a "Yes" answer indicates good usability practice, whereas "No" indicates a negative user experience.

The data was collected from eight participants in total (1 female; 7 males) aged between 21 and 61 (mean age 31.5). We compare the difference in task completion performance between two types of participants: *novice* users and *experienced* users. Novice users are considered to be those who answered "No, never" to the question "Have you ever used the Preference Learning Toolbox (prior to this experiment)?" while the experienced users are considered to be those who answered "Yes, a few times", "Yes, many times", or "Yes, I consider myself an expert". In general, the novice users were also vastly less familiar with machine learning and preference learning than the experienced users.

Fig. 5 shows the average time taken by each type of user (novice and experienced) to complete each task in both PyPLT modes. Note that the computation time taken to execute the experiments was excluded from these results. From these results, it is evident that none of the tool's tested features require an enormous effort to set up. On average, an experiment could be set up reasonably quick (in under 5 minutes) by both novice

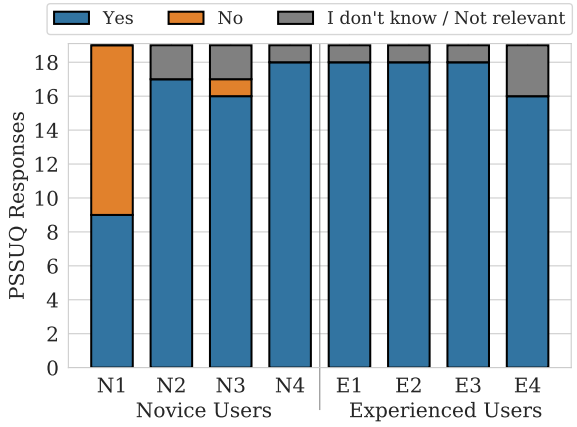

Fig. 6. Responses (y-axis) to the 19 questions of the Post-Study System Usability Questionnaire (PSSUQ) [23] across all participants (x-axis).

and experienced users. With the exception of the very first task, the tasks in the beginner mode took users less time to complete than those in advanced mode. The novice users were always slower to complete the tasks than the experienced users, on average. While task 3 was more complex than task 2, which was more complex than task 1, we observe that the time it took both novice and experienced users dropped in subsequent tasks. This suggests that the experiments became easier to set up as the users became more acquainted with the interface. This is corroborated by the responses to the questionnaire, in which all experienced users and 3 of 4 novice users responded that it was "easy to learn to use" the software. Only task 5 took longer to complete than the previous for either user type as it involved setting up several pre-processing options (automatic feature extraction, normalization, and data shuffling), each with its own parameters. In fact, this task took the longest to complete on average across modes and user types.

Figure 6 shows the number of positive, negative and neutral responses from each participant across all 19 questions of the post-study questionnaire. The vast majority of responses are positive, as 7 of 8 participants answered "Yes" to more than 75% of the questions. Specifically, all users of both types responded that they were satisfied with how easy it is to use PyPLT, and were able to complete the tasks quickly and effectively using the tool. Moreover, they all agreed that it was easy to find the information they needed and that the organization of information on the software screens was clear. They also unanimously liked using the GUI and were overall satisfied with the toolbox.

On the negative aspects of usability, only one novice user (N1) was underwhelmed by the use of PyPLT, and responded "No" to 10 of the 19 questions including "Was it simple to use this system?", "Did you feel comfortable using this system?", and "Was the information provided for the system easy to understand?". N1 informally expressed his frustration with the formatting parameter menu for loading the datasets, which he felt was slightly unclear. Furthermore, some users of both types were confused by a clash between scrollable areas in the pre-processing tab in advanced mode, as some of the options in the tab were not easily accessible when using the mouse wheel to scroll down the tab. This detail likely contributed to the prolonged amount of time taken by both user types to complete the second task in the advanced mode (see Fig. 5).

V. Conclusions and Future Work

This paper presented a Python-based Preference Learning Toolbox, an all-in-one software application and package for the modelling of ordinal data. The toolbox contains a variety of popular algorithms and methods for each stage of the modelling process, including pre-processing, feature selection and preference learning. The toolbox also allows for deep learning and is equipped with an autoencoder for automatic feature extraction. The toolbox may be used via its GUI or API, the former of which offers a simplified beginner mode for novice users as well as an advanced mode with detailed setup options for more experienced users. The setup details, results summary, as well as the SVM or ANN models derived using the tool may easily be stored for future use. Experiments reported in this paper show that PyPLT is generally easy to use and able to yield accurate models from real affect data.

The tool is *open source* and has been designed to facilitate further development and improvement. Along with the addition of new algorithms, methods and other options, we envisage that future work on PyPLT will include extending the tool to allow saved (pre-trained) models to be loaded into the toolbox to predict new instances and continue training. Another important extension would be to handle 2D data such as images and videos as input, which would require enhancements to the deep learning capabilities of PyPLT by integrating convolutional layers with ANN models. As revealed in the usability study, the GUI could be further refined, by making the formatting parameter menu for loading datasets and the scrolling functionality in the pre-processing tab (advanced mode) more intuitive.

The PyPLT website gives access to the latest distributions of the software and further documentation on how to use the tool through the beginner or advanced interface. Further guidance is also provided for developers seeking to modify or improve the software, which is version-controlled using Git.

Given the importance of ordinal labeling and annotation for subjective constructs such as emotion [1], [2] we aspire that the Python Preference Learning Toolbox will become a widely used and accessible software to researchers of affective computing and human computer interaction at large.

Acknowledgments

We thank Maltco Lotteries and the H2020 project Com N Play Science (project no: 787476) which co-funded the development of the PyPLT. We also thank Héctor P. Martínez, Vincent Farrugia and Phil Lopes for their help with migrating from the legacy version; Konstantinos Makantasis for his contributions to PyPLT; and all of the participants who tested and provided feedback on the tool at various stages.

REFERENCES

[1] G. N. Yannakakis, R. Cowie, and C. Busso, "The ordinal nature of emotions," in *Proceedings of the Intl. Conference on Affective Computing and Intelligent Interaction*. IEEE, 2017, pp. 248–255.

[2] ——, "The ordinal nature of emotions: An emerging approach," *IEEE Transactions on Affective Computing*, 2018.

[3] V. E. Farrugia, H. P. Martínez, and G. N. Yannakakis, "The preference learning toolbox," *arXiv preprint arXiv:1506.01709*, 2015.

[4] J. Fürnkranz and E. Hüllermeier, *Preference learning*. Springer, 2010.

[5] S. Kaci, *Working with preferences: Less is more*. Springer Science & Business Media, 2011.

[6] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*. ACM, 2002, pp. 133–142.

[7] ——, "Training linear SVMs in linear time," in *Proceedings of the SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*. ACM, 2006, pp. 217–226.

[8] J. Read, P. Reutemann, B. Pfahringer, and G. Holmes, "MEKA: a multi-label/multi-target extension to WEKA," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 667–671, 2016.

[9] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning," *Journal of Machine Learning Research*, vol. 12 (Jul), pp. 2411–2414, 2011.

[10] F. Eibe, M. A. Hall, and I. H. Witten, *The WEKA workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*, 4th ed. Morgan Kaufmann, 2016.

[11] P. Gupta, A. Hetzer, T. Tornede, S. Gottschalk, A. Kornelsen, S. Osterbrink, K. Pfannschmidt, and E. Hüllermeier, "jPL: A java-based software framework for preference learning," in *Proceedings of the LWDA 2017 Workshops: KDML, FGWM, IR, and FGDB*, September 2017.

[12] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and helmholtz free energy," in *Advances in Neural Information Processing Systems*, 1994, pp. 3–10.

[13] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv e-prints*, p. arXiv:1412.6980, Dec 2014.

[14] P. Pudil, J. Novovičová, and J. Kittler, "Floating search methods in feature selection," *Pattern recognition letters*, vol. 15, no. 11, pp. 1119–1125, 1994.

[15] C. Cortes and V. Vapnik, "Support-vector network," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[16] R. Herbrich, T. Graepel, and K. Obermayer, "Support vector learning for ordinal regression," in *Proceedings of the Intl. Conference on Artificial Neural Networks*, 1999.

[17] H. P. Martínez, Y. Bengio, and G. N. Yannakakis, "Learning deep physiological models of affect," *IEEE Computational Intelligence Magazine*, vol. 8, no. 2, pp. 20–33, 2013.

[18] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. N. Hullender, "Learning to rank using gradient descent," in *Proceedings of the Intl. Conference on Machine learning*, 2005, pp. 89–96.

[19] P. Lopes, A. Liapis, and G. N. Yannakakis, "Modelling affect for horror soundscapes," *IEEE Transactions on Affective Computing*, 2017.

[20] S. Koelstra, C. Muhl, M. Soleymani, J.-S. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, and I. Patras, "DEAP: A database for emotion analysis; using physiological signals," *IEEE Transactions on Affective Computing*, vol. 3, no. 1, pp. 18–31, 2012.

[21] F. Eyben, F. Weninger, F. Gross, and B. Schuller, "Recent developments in openSMILE, the Munich open-source multimedia feature extractor," in *Proceedings of the Intl. Conference on Multimedia*. ACM, 2013, pp. 835–838.

[22] D. Melhart, K. Sfikas, G. Giannakakis, G. N. Yannakakis, and A. Liapis, "A study on affect model validity: Nominal vs ordinal labels," in *Proceedings of the IJCAI workshop on AI and Affective Computing*, 2018.

[23] J. R. Lewis, S. C. Henry, and R. L. Mack, "Integrated office software benchmarks: A case study." in *Interact*, 1990, pp. 337–343.