The AmTriangle Meta-Dataset for Playing with Machine Learning

Artur Marques, Rafael de Amorim Silva, and Filipe Madeira

Abstract AmTriangle is a meta-dataset for machine learning (ML) that supports convenient and configurable dataset production in addition to a classifier that performs supervised learning. The idea is to have a never-ending source of supervised datasets, minimizing learning barriers to ML, for training models, and facilitating educational experiments and comparisons between different workflows. It is a "meta-dataset" because it is a solution to generate sets of samples (datasets), as many as desired, correctly classified. Each sample is a triangle, classified as "acute", "obtuse" or "right", according to trigonometry. Each generated dataset can be used to "teach-by-example" how to classify new samples, by different techniques, namely K-Nearest-Neighbors and neural networks. The triangles could be other analogous objects, other tuples.

Keywords Dataset · Education 4.0 · KNN · Supervised learning

1 Introduction

Machine Learning (ML) is an Artificial Intelligence (AI) area in which computer programs, "machines", can autonomously learn and classify data, not by explicitly stating all the outputs for every anticipated input sample, but rather from examples, which can "train" a model [1–3], ideally well enough to adapt to new samples.

A. Marques (✉) · F. Madeira
Escola Superior de Gestão e Tecnologia, Instituto Politécnico de Santarém, Santarem, Portugal
e-mail: artur.marques@esg.ipsantarem.pt

F. Madeira
e-mail: filipe.madeira@esg.ipsantarem.pt

Centro de Investigação em Artes e Comunicação – Pólo de Literacia Digital e Inclusão Social, Instituto Politécnico de Santarém, Santarem, Portugal

R. de Amorim Silva
Federal University of Alagoas–Computing Institute, Maceio, Brazil
e-mail: rafael@ic.ufal.br

A. Mesquita et al. (eds.), *Perspectives and Trends in Education and Technology*,
Smart Innovation, Systems and Technologies 320,
https://doi.org/10.1007/978-981-19-6585-2_22

In this context, a model is like a function that, receiving arguments, computes them, or classifies them correctly, with a certain degree of accuracy. The accuracy or confidence in the computed classification alludes to the underlying statistical techniques. "Statistical Learning" is another name for ML. The AmTriangle construct is for discrete classification problems; however, in other ML applications, namely regression, the output is non-discrete.

The techniques applied to AmTriangle's output datasets, discussed next, take advantage of the availability of the correct classification for every sample. The output datasets are suited to train and test ML algorithms. It is the correct classification availability that justifies the expression "supervised learning" [4, 5]. In other problems, the correct rating might not be available, and the machine will have to discover it—that would be "unsupervised learning" [6, 7].

In each "AmTriangle" dataset, all and each of the samples are vectors of three floating-point numbers, corresponding to the three internal angles of a triangle. As an example, the vector (90.0, 2.06, 87.94) is a valid sample, corresponding to a triangle with a classification of "right" for having an internal angle of 90°—note that the sum of the three numbers is 180. Angles greater than 90° are labelled "obtuse", and angles less than 90° are labelled "acute".

A dataset with, for example, 999 samples, can be represented as a 2D matrix of 999 lines by 3 columns. Because there are three possible ratings ("right", "obtuse", "acute"), this is called a 3-class or 3-target classification problem.

From a ML software practitioner's perspective, AmTriangle is a set of free or "libre" tools, written in the Python programming language. In its current version, the end-user scripts are "`dsgen.py`" (a datasets generator) and "`mlclass.py`" (a ML classifier that can classify any 3-tuple, based on a model it builds while learning from a training dataset). These tools are also referred to as the "generator" and the "classifier".

The code in "`dsgen.py`" produces datasets.

The code in "`mlclass.py`" builds a ML model, from the examples in an input dataset, and then it can classify any new tuple, with measurable accuracy.

On average, larger datasets (datasets with more samples) should lead to more reliable models. However, if there is an imbalance in the number of examples per class/target in a dataset, the learning process may result biased, because of misrepresented possibilities, so size is not enough. The generator can assure balanced representations of all classes, being that optional though, since the purpose of having this instrument is to facilitate playing with ML and observing (and measuring) the consequences of dataset composition on model behavior.

ML is one driver of automation. Studies on the effects of automation on vocations have long identified that most of the latter will have a significant number of tasks automated, as a consequence of the ongoing fourth industrial revolution [8]. The so-called "Education 4.0" is a learning approach aligned with the context of the current industrial changes, in which human creativity and complex human interactions emerge as reinforced needs, while repetitive, non-creative tasks become more exposed to alternatives, including AI-based solutions.

Education is one sector where automation is expected to have relatively low reach [9]. Human interactions during class time, at any scholar level, are very complex and heavily dependent on hard-to-extract and act-upon contexts. Activities not so centered on immediate personal interactions, e.g., researching and writing, pose different hurdles to automation, around the delicate integration balance of comprehension, communication, and creation activities, amongst others.

Still, automation will increasingly permeate education, eventually freeing people from administrative tasks, enabling automatic assessment instruments, and complementing humans in different ways, namely via digital assistants that we see as learning-capable personalized software agents. For this to happen gradually, related techniques and technologies can benefit from inter-disciplinary approaches that somehow lower or eliminate barriers to integration opportunities.

Trigonometry in general and triangle classification in particular are one such opportunity for seeing ML in action around familiar problems. Whether dataset and model generation are trivial enough, those interested in getting exposure to the subject, or exposing others to it, may consider AmTriangle.

2 Solution Architecture and Techniques

AmTriangle is a meta-dataset for ML, in the sense that it is about having control over datasets. Its usage pattern consists in outputting a randomly generated dataset, configurable in size and automatically exported to a file, then feeding it for model training and model accuracy testing.

The dataset generator offers the possibility to output completely random collections or to enforce the equal-representation of all n classes in an n-classification supervised learning problem: for example, if 999 samples are requested, the output can contain 333 triangles of each type (“right”, “obtuse”, “acute”).

The resulting dataset can be an input for the classifier, which currently builds a model using the K-Nearest-Neighbors (KNN) algorithm [10], at its highest complexity; that is, with $K = 1$.

When asked to classify a new sample N, the model will measure the “distance” of N, relatively to all learned examples, identifying the closest or less-distant tuple in the training dataset, and assigning its classification C to N.

On lower complexity calls, i.e., $K > 1$, the algorithm looks for the “top K” closest-to-N samples, finds the majority classification C, and ranks N as C.

This means that, in the extreme case of $K =$ number of elements in the training dataset, the most frequent classification would always be the one assigned to any new sample. This would build a low complexity model, probably low precision too, risking “underfitting” or “undercomputing” [11], as the model could not fit the correct values for the training data, overgeneralizing.

KNN is an option, but the generated datasets can support any other alternatives for model creation for supervised learning, namely neural networks. Neural networks are computational graphs in which nodes perform mathematical functions, weighting

their inputs to compute an output. That output will feed nodes in subsequent layers, until reaching a final layer whose outputs aim to approximate what the examples are teaching. With neural networks, the complexity will be high if the number of nodes and/or layers also present high values.

Very complex models can generate so-called "overfitting" situations [12], in which the model performs very well with the training data, but is unable to generalize as desired in the face of new samples.

A proximity measurement algorithm, suitable and adaptable to different dimensional spaces, including the 2D space in which triangles are inscribed, is the Minkowski distance. In AmTriangle, the classifier module uses KNN with K = 1 and the Minkowski distance.

The n*th* order Minkowski distance between samples P and Q is defined as follows:

$$d(P, Q) = \left(|p1 - q1|^n + \cdots + |pn - qn|^n\right)^{(1/n)}$$

That is, for n = 2 it is the Euclidean distance, and for n = 1 the Manhattan distance, the sum of the absolute values of the differences between all the components of the vectors to be measured.

These are the current techniques in-place, but being free or "libre" software, users will have the freedom to independently adapt this approach to their needs.

3 Workflow

The dataset generator module produces JSON (JavaScript Object Notation) files, which describe randomly computed triangles. The module accepts an argument that sets how many samples per "target" or possible classification class that are to be created.

The classifier module can run from one of these JSON files. For example, here is a command-line call to generate a dataset with just one sample of each type of triangle:

```
python dsgen.py 1
```

The purpose of this singular case is only to observe the generated JSON file, depicted in Fig. 1.

The dataset is structured in sections with "keys" named "triangles", "angles", "lengths", "targets" and "files":

- "triangles" is a collection of vectors, each with 3 real numbers, corresponding to 3 points that define a graphical representation of a triangle.
- The figure shows the triangle ((458, 185), (469, 185), (458,85)), which should be imagined in a frame of reference where the point (0,0) is the upper left corner of the plane.

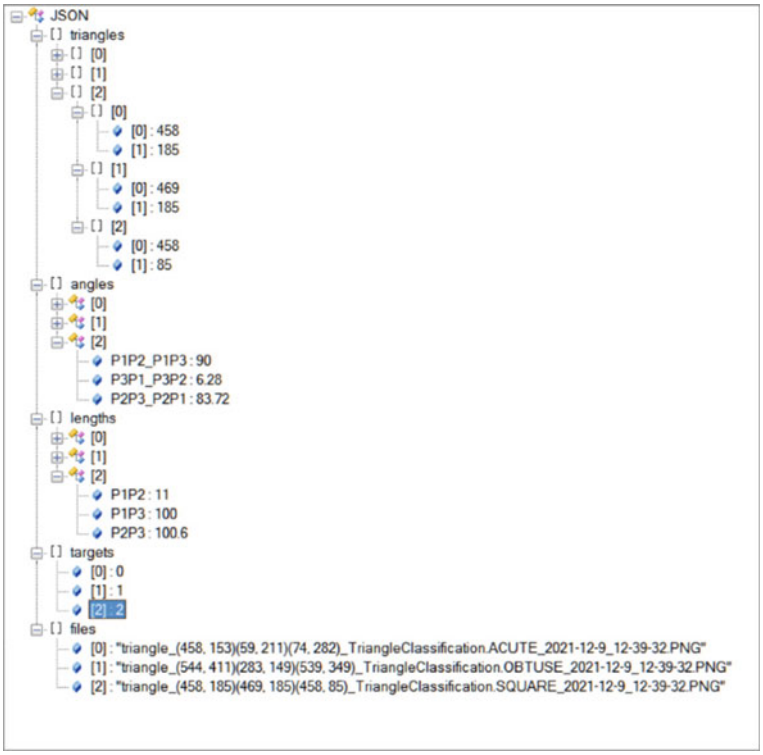

Fig. 1 A JSON for a dataset with just 3*1 samples

- "angles" is a collection of vectors, each with 3 real numbers, corresponding to the interior angles set by the sides defined by the triangle's points. The figure shows (90, 6.28, 83.72), expanded.
- "lengths" is a collection of vectors, each with 3 real numbers, corresponding to the lengths of the sides of a triangle. The figure shows (11, 100, 100.6).
- "targets" corresponds to the correct classification of each sample. In the JSON structure, it reads that the first triangle has target 0 ("acute"), the second target 1 ("obtuse"), and the third target 2 ("right").
- Finally, "files" corresponds to the names of files with pictures of the triangles if there is a need to visualize them.

In many machine learning problems, sample classification is provided by experts. For example, in Ronald Fisher's established "Iris dataset" [13], the biologist classifies 150 flower samples into 3 species/targets, based on 4 real numbers (length and width of sepals, length and width of petals); and, in the "UCI ML Breast Cancer Wisconsin Diagnostic dataset" [14], the creators classify images of tissues extracted from breast lumps as benign or malignant, based on vectors of 10 numbers.

In AmTriangle datasets, the correct classification is assured by trigonometric methods, which take the place of "experts". These methods only happen in the fabrication of the dataset for supervised learning. During the classification process, only ML techniques, based upon the examples, operate. In reality, classifying triangles does not need ML techniques—it is a problem solvable with secular trigonometry.

The rationale in the "AmTriangle" meta-dataset is the following:

- In first usages of ML, the lack of experience with peripheral areas, such as botany or medical imaging, can pose an obstacle to understanding what the samples mean and why they are classified as they are. The use of trivial samples, from a well-understood context, minimizes this barrier and frees the practitioner to focus on the specific ML tools that are in application.
- A triangle can be understood as any object with a trio of "features" or characteristics, such as, for example, a flying insect seen as (wingspan in mm, length in mm, mass in grams). Therefore, the triangles are not necessarily trigonometric—the practitioner's imagination is invited to think of something else.
- Moreover, since this is a project with free and open-source code (FOSS), the practitioner can adapt the generator module to produce n-dimensional entities, with $n > 3$, here finding an instrument that enables writing datasets of variable size [15], suitable for measurements that might help in understanding relationships between the size of datasets, and/or the number of features, and the results achieved with different models, trained differently.

In a typical workflow, first, the generator writes a dataset as a JSON file; next, the classifier inputs that file, builds and trains a model from the training data, making available an object that can classify new samples.

Here is an example of calling the classifier to learn from a dataset file, from the command line:

```
python mlclass.py 400_samples_per_target.JSON
```

The model used in the example call consists of 400 samples for each target. During the generator's operation, parameters in the code control accessory details, such as if picture-files for the triangles are being saved.

During the operation of the classifier, the following feedback happens: listing of the dataset samples, listing of each sample's corresponding classification, summary of the model, and a report on how the model performed with the training samples, including a measure of its accuracy.

4 Model Construction and Assessment

A triangle's internal angles determine its classification. Thus, one of the first operations in the classifier is the extraction of the internal angle values. Because there is extra information in the dataset, namely designations for the sides that form the

angles—useful for building graphical representations,—this data "reduction" operation returns a leaner data structure, representing only the strictly necessary values for model training and accuracy assessment.

The result is a list of vectors, each consisting of 3 angles.

Another parallel list holds the correct classification per vector.

What follows is a process of reserving a fraction of the dataset for training (75%, by default), and another fraction for gauging the training results (25%, by default).

For this purpose, the instrument "`model_selection.train_test_split`" of the module "sklearn" is used. The result is a division ("split") of the received dataset, in parts to be used for training ("train") and testing/assessing the training results ("test").

The following code snippet corresponds to the split:

```
from sklearn import model_selection as m
tupleTrainAndTestsSets = m.train_test_split(
listOfListsEachATrioOfAngles,
listOfTargetsForEachTrioOfAngles
)
X_train, X_test, y_train, y_test = tupleTrainAndTestsSets
```

It is common to capture this stage's output in the following identifiers:

- `X_train`—the list of vectors that will be used as examples for learning.
- `X_test`—a list of vectors, which will NOT be used as examples, but reserved for measuring the training accuracy, at a later stage. The machine will have to autonomously classify these test samples, without having access to their correct classification, thus allowing the ML practitioner to compare the results achieved by the model, with those that are assuredly the correct.
- `y_train`—the list of correct targets; that is, of classifications, for each of the samples in `X_train`.
- `y_test`—the list of correct targets, for classifying the samples in `X_test`. This list is essential for the process of measuring the accuracy of the model.

The model itself here discussed is a KNN classifier, using the Minkowski distance. The learning or training process occurs when the classifier's "fit" method is called.

```
knn = KNeighborsClassifier(metric="minkowski",
n_neighbors=1)
knnResult = knn.fit(X_train, y_train)
```

After "fitting" to the training data, which is learning from the examples, the programmatic object "`knnResult`" is used to make predictions about new samples.

For example, the execution of the following code assigns "`thePrediction`" a prediction of what the classification for "`someNewSample`" is.

```
thePrediction = knnResult.predict(someNewSample)
```

The prediction will be right or wrong. To determine how much can one trust the model's predictions, all elements of `X_test` will be arguments to the "`predict`"

method, and all the obtained results will be compared with the correct targets, as available in `y_test`.

If, for example, 70 out of 75 comparisons match, then the model accuracy is 70/75 = 0.9(3) ~ 93%.

5 Looking at the Generated Dataset and Model Accuracy

Each triangle consists of 3 attributes (or "features"). One visualization possibility for an entire dataset, is to work with a dispersion matrix, which is an instrument whose diagonal consists of histograms of absolute frequencies for each of the attributes under analysis: each of the internal angles, in this case. The remaining cells in the "scatter matrix", called "pair plots", are 2D representations of the interception of pairs of values of characteristics, two by two.

Figure 2 shows a "balanced" dataset with 100 samples of each triangle class, therefore a total of 300 samples, 75% of which were used for training (225) and 25% reserved for accuracy assessment (75). The "acute" triangle data points appear in red, the "obtuse" in green, and the "right" in blue. But any other colors can be used, depending on a correspondence dictionary, that follows the mathematical color model "RGBA" (Red Green Blue Opacity).

Depending on the number of samples and graphical rendering limitations, the histograms may not display all possible samples.

The AmTriangle kit includes a function for singular sample testing: the sample is subject to the model's predict method and the result is compared to the correct rating. The classifier module ends its operation by computing the model's accuracy, obtained by two different, but equivalent, ways:

- by using the score method of the model object:

```
knnResult.score(X_test, y_test);
```

- by directly comparing each prediction with the correct result.

  In the current example, the accuracy was ~ 93%:

```
Model accuracy / average matches: 0.9333333333333333
Model score by knn.score 0.9333333333333333.
```

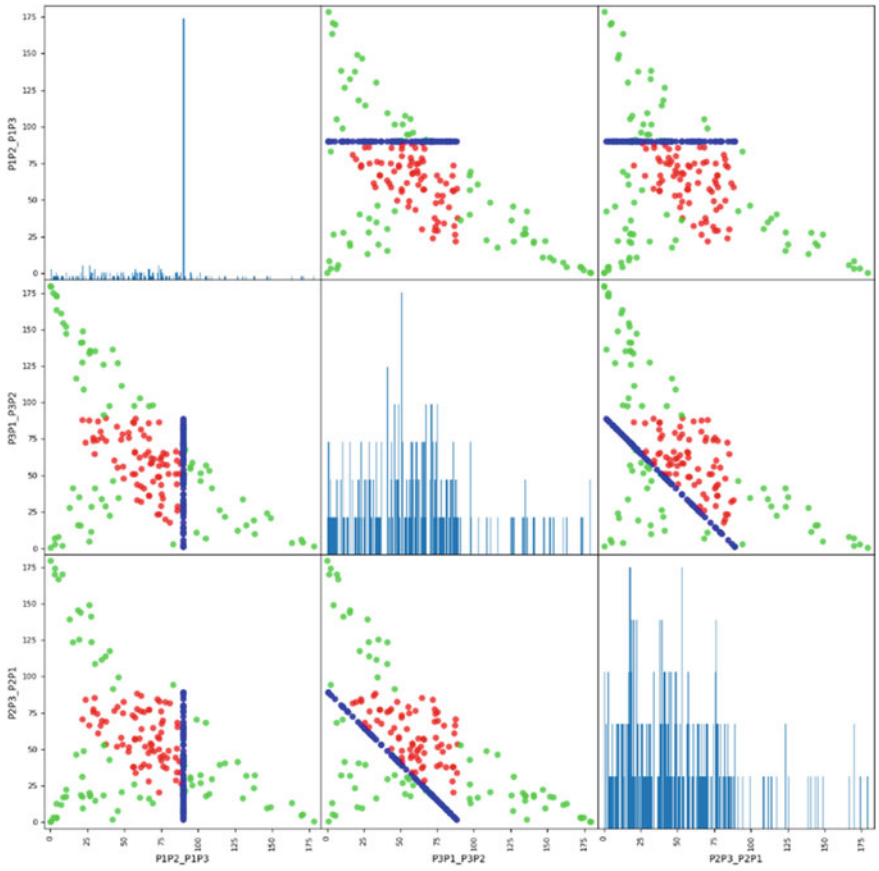

Fig. 2 Scatter Matrix for a "balanced" AmTriangle dataset with 3*100 samples

6 Discussion of Results

AmTriangle is a meta-dataset for machine learning, supporting convenient and configurable dataset production, plus a classifier that does supervised learning.

One idea is to have a never-ending source of supervised datasets—this is effectively achieved by the `dsgen.py` module. It also aims to minimize learning barriers to ML, for training models, facilitating experimentation and comparisons between different workflows—all of these can be conducted with the included `mlclass.py` tool.

Thus, this is one solution supporting educational playing with ML, around a familiar problem, defeating availability limitations and using trivial triangles as the starting samples, generalizable to tuples that can be more complex.

The need for "meta-datasets" has been previously studied [16], but in the context of benchmarking "few-shot" classification models for Deep-Learning. The AmTriangle

solution is not comparable: not a benchmark, but rather an approach to facilitate new "triangle" datasets whenever needed, and a procedure for experimenting with them on different models.

In AmTriangle, problems can happen on purpose, such as when requesting small imbalanced datasets. "Good" datasets can also be enforced, as when generating large, balanced, perfectly classified collections that can train adaptable models. In the end, it is up to the practitioner to decide what the experiments will be, assisted by the included generator and classifier tools.

References

1. Alzubi, J., Nayyar, A., Kumar, A.: Machine learning from theory to algorithms: an overview. J. Phys: Conf. Ser. **1142**, 012012 (2018). https://doi.org/10.1088/1742-6596/1142/1/012012
2. Gangadhar, S., Shanta, R.: Chapter 8—Machine learning. Handbook Stat. **38**, 197–228 (2018). https://doi.org/10.1016/bs.host.2018.07.004
3. Jaime, G.C., Ryszard, S.M., Tom, M.M.: 1—an overview of machine learning. 3–23 (1983). https://doi.org/10.1016/B978-0-08-051054-5.50005-4
4. Caruana, R., Niculescu-Mizil, A.: An empirical comparison of supervised learning algorithms. Icml 06, 161–168 (2006). https://doi.org/10.1145/1143844.1143865
5. Dridi, S.: Supervised Learning—A Systematic Literature Review (2021)
6. Barlow, H.B.: Unsupervised learning. Neural Comput. **1**(3), 295–311 (1989). https://doi.org/10.1162/neco.1989.1.3.295
7. Ghahramani, Z.: Unsupervised Learning. In: Bousquet, O., von Luxburg, U., Rätsch, G. (eds.) Advanced lectures on machine learning: ML Summer schools 2003, Canberra, Australia, February 2–14, 2003, Tübingen, Germany, August 4–16, 2003, Revised Lectures, pp. 72–112. Springer, Berlin Heidelberg, Berlin, Heidelberg (2004)
8. Insights, D.: The Fourth Industrial Revolution (2020)
9. Chui, M., Manyika, J., Miremadi, M.: Where machines could replace humans—and where they can't (yet) (2016)
10. Zhang, Z.: Introduction to machine learning: k-nearest neighbors. Ann Transl Med **4**(11), 218–218 (2016). https://doi.org/10.21037/atm.2016.03.37
11. Dietterich, T.: Overfitting and undercomputing in machine learning. ACM Comput. Surv. **27**(3), 326–327 (1995). https://doi.org/10.1145/212094.212114
12. Roelofs, R., Shankar, V., Recht, B., Fridovich-Keil, S., Hardt, M., Miller, J., Schmidt, L.: A meta-analysis of overfitting in machine learning. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F.d., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc. %U (2019). https://proceedings.neurips.cc/paper/2019/file/ee39e503b6bedf0c98c388b7e8589aca-Paper.pdf
13. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Ann. Eugen. **7**(2), 179–188 (1936). https://doi.org/10.1111/j.1469-1809.1936.tb02137.x
14. Wolberg, D.W.H., Street, W.N., Mangasarian, O.L.: Breast Cancer Wisconsin (Diagnostic) Data Set (1995)
15. Alwosheel, A., van Cranenburgh, S., Chorus, C.G.: Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis. J. Choice Model. **28**, 167–182 (2018). https://doi.org/10.1016/j.jocm.2018.07.002
16. Triantafillou, E., Zhu, T., Dumoulin, V., Lamblin, P., Evci, U., Xu, K., Goroshin, R., Gelada, C., Swersky, K., Manzagol, P.-A., Larochelle, H.: Meta-dataset: a dataset of datasets for learning to learn from few examples. In: International Conference on Learning Representations (2020)