SphereNet

Table of Contents

1 Some facts about the classifier

- Has been written with numpy[1] (linear algebra library) and numba[2], a JIT compiler[3] for python and been built to be able to handle large scales of data using highly optimized parallelized algorithms. So it should be fast, even with a lot of data.
- Iintroduces new concepts and ideas (tunable hyperparams) to SphereNet which can be configured as well as the possibility to use SphereNet as ensemble learning[4] method with multiple classes (MultiSphereNet).
- Can be used in the style of an scikit-learn[5] classifier (easy to tune and practicability).

1 https://numpy.org/
2 https://numba.pydata.org/
3 https://en.wikipedia.org/wiki/Just-in-time_compilation
4 https://en.wikipedia.org/wiki/Ensemble_learning
5 https://scikit-learn.org/stable/

2 How does it work?

Notice that every time running one of the functions for the first time it will take a little longer because of the JIT compilation[6].

2.1 SphereNet

2.1.1 Parameters

- *in_class* [default: 1]: The class that will be classified as True or inside the spheres.
- *min_dist_scaler* [default: 1.0]: The value every sphere radius will be scaled by after calculating (before optimization).
- *min_radius_threshold* [default: -1]: Minimum radius threshold value. If a sphere radius is smaller it will be thrown out (before optimization). Set to -1 to accept any sphere radius.
- *optimization_tolerance* [default: 0]: Optimization tolerance (see below).
- *optimization_repetitions* [default: 1]: Number of times to repeat optimization algorithm. Should only be higher than 1 if using parallelized algorithm.
- *optimization_parallel* [default: False]: Parallelizing the optimization will make it a lot faster depending the number of your CPU cores, but can lead (because of race conditions[7]) to more spheres produced (i.e. not keeping the best classifying ones but two slightly worse instead).
- *min_num_classified*: [default: 2]: The minimum number of points from *in_class* that have to be classified by this sphere for it to be kept. Prevents from creating single spheres for every outlier or smaller outlier clusters if higher.
- *max_spheres_used* [default: -1]: Maximum number of spheres used. The optimal *n* spheres will be kept for classification. Set to -1 to keep all spheres after optimization.
- *metric* [default: 'euclid']: The metric used for distance measurement.
    - std: Standard deviation of x - y
    - minkowski
    - euclid
    - hamming
    - max: maximum of |x - y|
    - cosine
    - jaccard
    - dice
    - canberra
    - braycurtis
    - correlation: correlation coefficient between x and y
    - yule
    - havensine
    - sum: sum of x - y
    - mse: mean squared error
    - ots: 1 / sum((x - y) ^ 2)
- *p* [default: 2]: The *p* value (order of magnitude) when using the minkowski distance (*p*=1 equals manhattan distance, *p*=2 equals euclid distance).
- *standard_scaling* [default: False]: Apply standard scaling on data with StandardScaler[8].
- *normalization* [default: False]: Apply normalization on data with Normalizer[9].

6 https://en.wikipedia.org/wiki/Just-in-time_compilation
7 https://en.wikipedia.org/wiki/Race_condition
8 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
9 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html

- *remove_training_outliers* [default: False]: Remove outliers with LocalOutlierFactor[10].
- *verbosity* [default: 0]
    - 0: show no logs at all
    - 1: show text logs
    - 2: show text logs and progress bars

2.1.2 Training

Training the algorithm consists of three steps:

2.1.2.1 1. Preprocess

- Apply scaling if set by *standard_scaling*.
- Apply normalization if set by *normalization*.
- Remove outliers if set by *remove_training_outliers*.

2.1.2.2 2. Generating the spheres

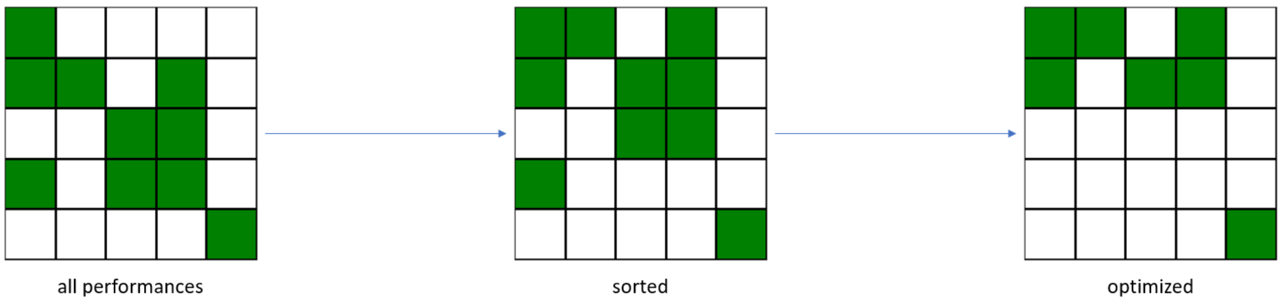
- For every point of the *in_class* find nearest (using the defined *metric*) point not belonging to the *in_class* (belonging to *out_class*). Use the *in_class* point as center and the distance to this nearest outside point as radius of a new sphere classifier.
- Scale each radius with *min_dist_scaler* and filter out poinyd with *min_radius_threshold*.
- Create a performance boolean matrix (later used by optimization) and filter with *min_num_classified*. Each row of the performance matrix stands for a point of the *in_class* used as center with it's associated radius as classifying sphere. Its columns depict for each point of the *in_class* (here used as test points) a boolean value whether it is inside the sphere or outside using the defined *metric* (*short*: rows depict points used as sphere-classifiers and columns depict for each point (here used as test-point) if it is classified by the row or not).

2.1.2.3 3. Optimization

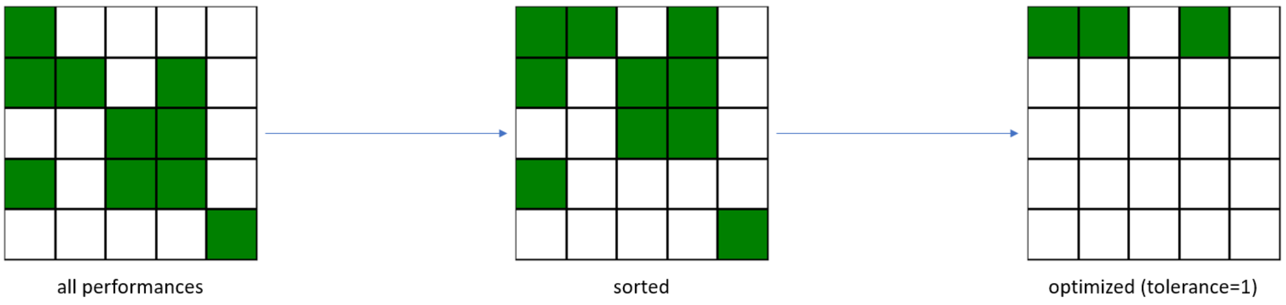
The second step will produce a huge performance matrix, only a small amount of points will be filtered out to be used as classifying sphere. This step will filter out a huge amount of spheres to be used as classifiers.

The main idea behind the optimization algorithm is to remove all rows, which's true-values are already contained in any combination in one of the other activated rows (each column, where the row that is currently being tried to remove contains a true-value, has a true-value in one of the other rows in the corresponding column). What this means is, if all points a sphere classifies can be classified by another sphere-classifier or a combination of other spheres, this row can be removed. For the algorithm to be as fast as possible, the performance matrix will be sorted by the amount of true-values each row has. When optimizing, it will first be attempted to remove rows with less true values (more probable to be removed) and the searching for other true-values is done first in the rows with most true-values, since they are more probable to contain the column's respective true-value. To note is, that once a row is deactivated (optimized out / removed) by the algorithm, it will not be searched anymore.

10 https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html



The parameter *optimization_tolerance* defines the number of spheres that is already assumed to be found for each row before searching for other true-values in the matrix. This way, the higher *optimization_tolerance* is set, the more spheres will be removed. This can prevent overfitting, too large / complex models but also decrease accuracy.



2.1.3 Testing

A new point will be classified as the inside class, if it is inside one of the spheres that are left over after training. Else it will be classified as belonging to the outside class. This will typically increase

2.1.4 Reducing algorithm size

If the algorithm has a too large number of spheres, another possibility than playing around with the hyperparameters is to call the method *reduce_size* after training the classifier. This will first calculate the distances between all for the classification used spheres' centers with the defined *metric*. Those centers, that are from both points' perspectives their respective nearest neighbor will be merged to one point by averaging their radius and coordinates. This process can be repeated multiple times (set the *repetitions=n* parameter). The smaller the complexity of the classificator, the more accuracy it will probably lose, but this is typically not too high of a price for the gained weight loss.

2.1.5 Notices

- The algorithm is **extremely prone to outliers**. It is advised to **filter, generalize and scale data as good as possible beforehand**, this can have quite a big effect (actually not only on this classifier, this is a general notice actually).
- Typically, the algorithm scales (produces more spheres) the larger the dataset.
- Data augmentation can help when using small datasets (with noise), but will also create very fast a very high complexity.
- The algorithm **overfits very fast**. Can be prevented with *optimization_tolerance* and *min_num_classified* as well as appropriate preprocessing beforehand.

- Scaling and normalization helps seldom, but sometimes.
- The *metric* hyperparameter is probably the most important to be tuned (there will be surprises!).
- Using this with a non-euclidean distance metric means using it in a non euclidean room, so these are not really hyperspheres anymore!
- Don't try to understand this algorithm only in 2d or 3d, for CoSmA these are 16 or 32 dimensional hyperspheres and have nothing to do anymore with just a 3d interior detection like you would imagine it with thresholds etc (especially when using a non-euclidean metric!).

2.2 MultiSphereNet

Whereas the normal SphereNet can only be used for binary classification, using MultiSphereNet will use SphereNet as ensemble method[11] (creating one SphereNet for every class).

2.2.1 Parameters

All parameters of SphereNet are applicable here as well. Additional parameters are:

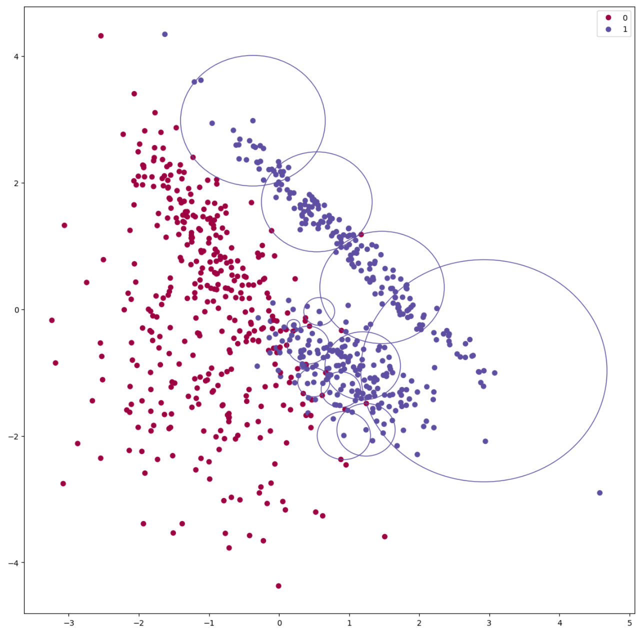
- *pred_mode* [default: 'conservative']: The prediction mode
    - conservative: Fastest classification mode. If there is a conflict between classifications (point to be classified is in spheres of multiple classes) it will not be resolved. Points that are classified as outside by all SphereNets will be marked as class -1. This mode should be advised to not be used in production cases.
    - careful: If there are classification conflicts, those will be resolved by comparing max inside distance of the respective classifying sphere. The class of the sphere the point is further inside of will be used for class prediction. Points that are classified as outside by all SphereNets will be marked as class -1. This can improve the accuracy. The more conflicts there are to be resolved, the longer the classification takes.
    - force: Same as careful mode, but a class is forced if the point is outside of all SphereNets by using the class of the sphere border the point is nearest to. This can improve the prediction accuracy by quite a lot. The more conflicts and complete outside points there are to be resolved, the longer the classification takes. Reducing the *min_dist_scaler* distance can improve in this mode the classification accuracy, but also make the model more complex. This will also produce more cases of complete outside points, means more classification calculations to be done.

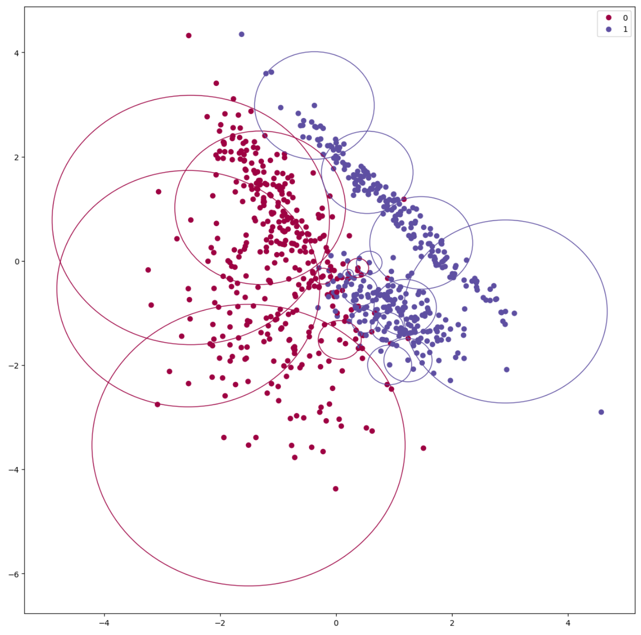
2.2.2 Notices

- What makes the classification take longer by most is the number of spheres, not the classification mode.
- MultiSphereNet with *pred_mode* 'force' is normally larger, but in almost all cases better than using only one SphereNet.

11 https://en.wikipedia.org/wiki/Ensemble_learning

2.3 Visual comparison of MultiSphereNet and SphereNet on an example





These are randomly generated data points belonging two different clusters: Class 0 (red) and class 1 (blue). The left picture displays, which spheres the normal SphereNet implementation (with *metric*='euclid' and *min_num_classified*=3 and *in_class*=1) would generate. Everything inside these spheres will be classified as belonging to class 1, everything outside as belonging to class 0. This creates an predictions accuracy of 93.2%. In the right picture, two SphereNets (a MultiSphereNet ensmble) has been used with *pred_mode*='force'. This improved the prediction accuracy to 95.6%.