Учитывая уже проделанное, напишу покороче (но так же, и с кое-какими смысловыми поправками). – Имеем дело с данными той же самой структуры, многие из настроек предиктора тоже сохраняют актуальность.

learn = актуальна

columns\_num = актуальна

predict **= НЕактуальна**

unit\_step = актуальна

Вместо # Начальное количество точек-направлений» заведем настройку max\_num\_of\_directions - как было в версии непосредственно предшествующей итоговой.

prescalers - не очень сильно нужны, но полагаю, мы оставим настройку чтобы поменьше вмешиваться в уже собранную структуру.

normalizing = on актуальна

normalize\_by\_mean\_distance = on актуальна

актуальны и ниже стоящие настройки :

input\_filename = strain2f.txt

input\_separator = tab

output\_filename = results.txt

output\_separator = tab

output\_numbers\_use\_comma = on

output\_print\_header = on

output\_func\_values = on

output\_weighted\_sum\_formula\_results = on

test\_output = on

test\_output\_filename = test\_output.txt

read\_learn\_data\_from\_file = off

learn\_data\_filename = data\_learn.csv

use\_reduction = off

use\_reduction\_size = 60

reduced\_rows\_filename = rows\_used.csv

….включительно до

debug = on

Ниже - настройки подстановки игреков исчезают за отсутствием таковой операции. …Как и скалеры (но не «прескалеры» о чем уже отмечал выше) - так как скалеры тоже теряют смысл за отсутствием соответствующих алгоритмических действий.

y\_pow = -2.0 y\_d = 0.15 - это естественно актуально, так как в основе модели остается тот же степеннОй функционал расстояний.

Соответственно, сохранятся, наверное, и какие-либо другие настройки, нужность/ненужность их, полагаю, будет ясна по смыслу.

Итак, исходя из известных координат и заданных настроек можем определить значения функционала для любой точки – как действительной (т.е. из тех что в массиве. так и вообще любой- желаемой, задав ее в данной системе координат.

Подготавливаем матрицу расстояний между точками – запоминаем ее в начальном виде, и предусматриваем возможность запоминания дальнейших изменений.

– для КАЖДОЙ точки произвели смещения на единичный ШАГ сдвига по направлениям: или тотально - до каждой из всех остальных точек, или до выборочного количества точек – смотря по настройке и реальным данным. Выполним это, остановились на варианте, где сдвиг точки дал наибольшее уменьшение функционала (суммы степенных расстояний) по сравнению со всем остальными сдвигами всех точек, **включая саму эту точку по всем используемым направлениям**. Зафиксировали этот сдвиг. (что означает получение и новой матрицы расстояний – расстояния, в которых задействована сдвинутая точка, ведь изменились) Перешли к следующей совершенно аналогичной итерации.

Повторяем все это, перемещая точки до тех пор, пока не получим:

СДВИГ (ОДНОКРАТНЫЙ ИЛИ МНОГОКРАТНЫЙ - НЕВАЖНО) **ЗАДАННОГО КОЛИЧЕСТВА ТОЧЕК**. Понятно, что количество сдвинутых точек и количество потребовавшихся на это итераций – вещи разные – ведь одни и те же точки могут совершить многократные сдвиги, - выбор ,согласно критерию может пасть на них много раз, пока не претерпит хотя б однократный сдвиг всё заданное количество точек

И, результирующий этап:

Выбираем точки, сделавшие **перемещение** за все действие алгоритма (от начального своего положения – в конечное местоположение на момент прекращения алгоритма) МЕНЬШЕ заданной пороговой величины (то есть, если мы зададим достаточно большое пороговое значение, то и вообще все точки могут попасть, но этого мы делать как раз не будем, а напротив пороговое значение выставим ск. всего НУЛЕВЫМ – чтобы попали лишь точки, которые вовсе остались неподвижными).

Для каждой такой точки делаем по вектору В, который содержит разность степенных расстояний от каждой точки до (n-1) остальных (-всех остальных) точек «learn» по матрицаM (начальная) и матрицаМ’-конечная.

Рассчитываем коэф-ты парной корреляции данных векторов (с исключением позиций, связывающих две данные точки) между собой и строим матрицу коэффициентов n\*n (матрицаК).

И выполняем ГРУППИРОВКУ точек, на основе матрицы:

*Немного подробнее:*

*Находим наибольший парный коэфф-т. Образуем из двух точек, которые им связаны, группу.*

*Находим следующий по величине парный коэфф-т: теперь смотрим, - входит ли одна из связываемых им точек в уже ранее образованную группу (-на данный момент она одна, в общем же случае правильно будет сказать «уже входит в какую-то из ранее образованных групп») … если НЕ входит, то образуем вторую группу. Если входит: проверяем, не имеет ли эта новая точка (а также (!!) точки уже объединенные с нею в группу, если таковые уже есть) с какой или какими-либо ранее включенными в эту группу точками коэфф-та НИЖЕ порогового значения. Если НЕ имеет, то просто присоединяем точку к группе. Если ИМЕЕТ, то создаем новую группу (или – в общем случае – НЕ ОБЪЕДИНЯЕМ группы).*

*/далее, как доделаем, добавим еще и группировку по «мягкому правилу» - когда напротив – чтобы точкам оказаться в одной группе. Достаточно им только между собой иметь коэф-т корр. Достаточно большой положительной величины – без проверок через третьи точки, что даст более монолитную картину групп - они, очевидно, будут более крупными и в меньшем количестве /*

*И так, - пока все (все отобранные по пороговой величине) точки, согласно коэф-ту, отражающему тесноту парной связи, не распределим по конечному числу групп.*

Далее остается два заключительных процесса алгоритма:

1) Что делаем с точкам, перемещение которых не позволило им попасть в набор первично классифицируемых по группам точек - для каждой такой точки находим группу В КОНЕЧНОМ СОСТОЯНИИ МАТРИЦЫ (а никак уж не в первоначальном) ближайшую –уже проклассифицированную в какую-нибудь группу точку.

- берем по отдельности каждую такую еще не отклассифицированную по группе точку.

Смотрим по матрице расстояний, по 15 (к примеру – это задаваемое значение) точек ИЗ КАЖДОЙ ГРУППЫ, ближайших к данной точке. Совершаем 15\*число групп(m) «проходов» данной точке вдоль по 15\*m расстояниям до каждой точки с шагом равным ШАГУ сдвига, заданному для предшествующих стадий алгоритма. Пусть m получилось =8 Если в группе не набралось 15 точек, берем значит. все имеющиеся.

Смотрим МАКСИМАЛЬНЫЕ значения функционала (суммы степенных расстояний) которые точка переживает за все шаги перемещения вдоль по каждому из 120 ребер. Там где МАКСИМУМ окажется МИНИМАЛЕН - в ту группу с точкой, принадлежащей которой соединяет найденное ребро, и классифицируем точку. И так раскидываем все непроклассифицированные из-за своей подвижности точки.

– Вот к этой группе (к которой относится эта ближайшая) и относим точку. И так для всех остальных. – Итак, в итоге чего у нас теперь вообще все точки массива распределены по группам (неверное, кластерами вернее назвать, ну неважно).

Выводя промежуточный итог, в файле результатов добавляем новый столбец - «номер группы» - просто чтобы теперь как-то отличать – куда какая точка попала. Нумерацию групп мне кажется разумнее всего вести по принципу: группе с максимальным числом точек присваиваем №1 – и далее, нумеруем по убыванию числа точек.

2) Итак всё сгруппировано - вероятно, еще после этого не помешает очистить картину от мелких случайных групп. Задаем минимальный порог количества точек в группе (ну, пусть =6) ..Пусть, всего в ходе анализа, образовалось у нас 9 групп. Находим группу с наименьшим числом точек (если таких больше одной – с одинаковым числом – берем любую произвольно, аналогичного же принципа придерживаемся и далее) - берем по отдельности каждую точку этой группы. (небольшая ремарка – обращу внимание, что и для пункта 1) и здесь – в пункте 2) теперь алгоритмы распределения точек одинаковые сделал ( до этого в первом пункте алг-м был свой) ..одинаковые за исключением лишь той разницы, что точки из первого пункта просто никак изначально не проклассифицированы, а здесь – во втором пункте речь идёт о «переклассификации»

Смотрим по матрице расстояний, по 15 (задаваемое значение) точек ИЗ КАЖДОЙ ГРУППЫ из остальных 8-ми, ближайших к данной точке. Совершаем 15\*8= 120 «проходов» данной точке вдоль по 120ти расстояниям до каждой точки с шагом равным ШАГУ сдвига, заданному для предшествующих стадий алгоритма. – Смотрим МАКСИМАЛЬНЫЕ значения функционала (суммы степенных расстояний) которые точка переживает за все шаги перемещения вдоль по каждому из 120 ребер. Там где МАКСИМУМ окажется МИНИМАЛЕН - в ту группу с точкой, принадлежащей которой соединяет найденное ребро, и классифицируем точку. И так раскидываем все точки выбранной группы. Затем, принимаемся за следующую по малости группу. И так, пока все малые (не проходящие по пороговому значению) группы не распределятся по значимым группам. Следствия: теоретически, группы изначально не проходящие по критерию малочисленности в них точек, могут в процессе отстоять свою сохранность - за счет пришедших в них точек из еще более малых групп, перейти через критический порог по количеству точек; теоретически, одна и та же точка может и несколько раз перепрыгнуть из группы в группу – так как изначально она может перейти в группу, тоже относящуюся к малым – затем, когда дело дойдет уже до этой группы (и окажется, что она так и не выбралась из категории малых) точку перекинет еще в какую-нибудь другую и т.д.

- По итогам, формируем файлы окончательных результатов, где перешедшие точки (если таковые случатся) соответственно изменят номер группы.

Последнее, что осталось из важного - пару моментов :

1) добавляем «боевой» режим - режим классификации в группы новых точек. которые появляются после того, как процесс классификации завершен. – Создаем для этого отдельный файл - в него (с помощью каких то внешних программ, или вручную) вносится строчка(строчки) формата, аналогичного имеющемуся – тоже число столбцов – id-шник и координаты. Нажимаем условную «кнопку» - на деле, это пока просто будет включение в конфиге в состояние «on» соответствующей настройки – в результате чего этот файл прочекивается и находящиеся в нем точки классифицируются в уже созданные группы, способом совершенно аналогичным описанному для непрклассифицированных или – неудовлетворительно классифицированных точек (см.выше). Главный смысл – не выполнять опять-сначала выявление групп заново ради новых точек, а пользоваться уже готовыми результатами, которые где-то для этого сохранены.

После каждого «нажатия» : или создается по новому файлу или переписывается некий файл – где точки заносятся и одновременно - этим новым точкам присваивается номер группы, в которую она определяется. После чего (пока вручную, потом – отдельно подумаем) строчку можно перенести в основной файл результатов – формат строк совпадает, а id-шник , чтобы был уникален, можно пока вручную перезадавать.

2) Из предиктора стоит взять элемент «обучения» пункт «3» с настройкой # Число лучших точек-направлений с предыдущего шага directions\_best\_from\_prev\_step - адаптировав под новый алгоритм – то есть : в рамках одной точки, если речь идет не о тотальном, а о выборочном подходе, запоминаем набор наилучших (дающих наилучшее **падение** функционала) направлений и пробуем эти направления на очередной попытке сдвига данной точки, наряду с направлениями из новой случайной выборки –короче, всё как и сейчас реализовано. Здесь же - по остальным аналогиям из скалерных доделок – пункт «2» смысл естественно теряет – так как в этом алгоритме интересует уже теперь только и именно максимизация ПАДЕНИЙ функционала. – Пункт «1», говорящий о том, что каждое направление рассматривается как прямое, так и как обратное – т.е. «от» точки - вполне может быть перенесен и в этот алгоритм.

…Еще вероятно, что коэф-т парной корреляции по итогам тестов заменим чем-то немного иным, если неэффективно будет показывать себя, но это проще сначала потестировать, прежде чем делать выводы.

Всё – весь процесс описан : формирование матрицы расстояний – формирование матрицы значений основного функционала (сумм степенных расстояний) на основе которых производится анализ - процесс вариации сдвигами точек, чтобы определить направления «скатываний», что напоминает градиентные методы, только в дискретном варианте - группировка точек, произведенная на основе анализа выявленных направлений скатываний - заключительные уточнения в виде «догруппировки» и перегруппировки.

Жду отзывы и вопросы.