# An Explanation of the Correlation Eliminator Algorithm

## 1 Introduction

This explanatory note explains the 'correlation_eliminator' function in the data cleaning functions folder. The purpose of this algorithm is to identify the fewest number of variables that must be removed in order to eliminate all pairwise absolute correlations above a defined cutoff between the variables in some design matrix. Being highly correlated above a cutoff is a reflexive, symmetric, but not necessarily transitive relationship, and eliminating every high pairwise correlation would reduce the relationship to a purely reflexive one,[1] so solving the more general problem of how to reduce any reflexive and symmetric relationship to a purely reflexive one will also solve the correlation problem. Even better, solving this problem will allow us to resolve any entangled data whose entanglement is defined by some fuzzy measure, e.g. 'having x proportion of the same data points as another variable.' This is exactly what the 'isolator' function (which the 'correlation_eliminator' function uses) does, and the following sections justify the rules that the isolator function uses.

It was recently pointed out to me[2] that the problem the isolator function is solving is functionally the same as the minimum vertex cover problem. The minimum vertex cover problem is an NP complete problem in which one must identify the smallest set of vertices in a graph such that every edge of the graph ends in at least one of the vertices. As described in the next section, the problem that the isolator function is solving can be represented by a graph, and the isolator function identifies a minimal set of vertices (which I call points) within a graph such that removing all the edges (which I call connections) ending in at least one of the vertices identified removes all connections. It should be clear from this description that solutions to the problems the isolator function solves and solutions to minimum vertex cover problems will be equivalent if the minimum vertex cover problem is equivalent to the graphical representation of the problem solved by the isolator function.

Accordingly, the output of the isolator and correlation eliminator functions is some number of what I call 'optimal removal sets.' Formally, these are sets that satisfy two conditions. 1) If the elements within these sets are removed from consideration, then no connection (high correlation) between two distinct variables will exist, and 2) there exists no set of strictly lower cardinality that satisfies (1). We could alternately think about 'optimal stubborn sets' which contain the greatest number of variables such that no connection exists between any two, and are complements of the optimal removal sets, but I will take an optimal removal set approach here.

[1] A purely reflexive relationship is also strictly speaking symmetric but by 'purely reflexive' I mean that no two distinct points are related.

[2] Thanks go to my cousin Gus Kasper.

The algorithm serves to remove near duplicate information from a dataframe of explanatory variables, and I personally used it prior to principal component analysis in order to avoid skewing the importance of the principal components. To summarise how the algorithm actually works, it takes the following steps:

1. The inputted dataframe and correlation cutoff are used to create a representation of the problem.

2. The representation is fed to the isolator function, which starts by repeatedly removing whole swathes of points that are optimal to remove.

3. Once it can no longer find optimal points, it creates two branch solutions, one of which must be optimal, and resumes removing optimal points until it can no longer find any.

4. This bifurcation and removal continues until a branch solution that solves the problem is the shortest branch under consideration.

5. If the problem gets out of hand and the number of branches under consideration balloons past a user defined level, the algorithm defaults to a quasi-greedy approach.

The rest of this document proceeds by, first, discussing representations of the problem and making general observations. Then, I discuss a logical solution to the problem, and finally I reformulate one of the elements of the logical approach for computational efficiency. I conclude by explaining how the points discussed fit into the algorithm.

## 2 Representations of the Problem

There are two ways to represent the problem being solved that we should explore ...

1. As a dictionary where, for each variable, there is an accompanying list of variables with which it is highly correlated (including itself).

2. As a graph, where variables are represented as points and high pairwise correlations are represented as strings connecting the points. Reflexive connections, i.e. loops connecting points to themselves, are left out for ease of representation. An example of this illustration is given in Figure 1.

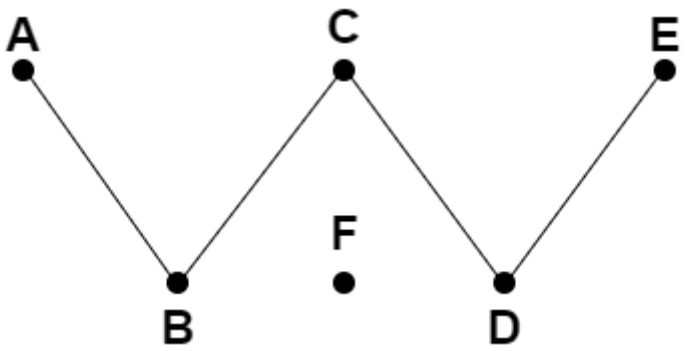


Figure 1: A visual representation of an example problem

The first representation is the one the algorithm actually uses to keep track of the state of the problem, though it is instantiated in a pandas series rather than a dictionary. The second representation can help us think of a physical analogy of the problem: we need to come up with rules for how to remove the least number of pins on a pin-board whilst removing all strings connecting the pins, for any arrangement of pins and strings, where removing a pin removes any string attached to the pin. From this analogy it is obvious that any solitary pins such as F in figure 1, i.e. variables highly correlated only with themselves, can safely be ignored, since there is never any point in removing them and they cannot affect whether another point must be removed. I take advantage of this fact when running the algorithm computationally, as I drop such singleton points to clean up the series representation.

Using the second representation, we can see from figure 2 below that a greedy approach in which we successively remove the pin that gets rid of the greatest number of strings can be sub-optimal even for a simple problem. The greedy approach suggests that, for the problem in figure 1, either B, C, or D should be removed first. However, removing C first would necessitate the removal two more points. Removing B and D is the only optimal solution and the greedy approach would identify this solution two thirds of the time.

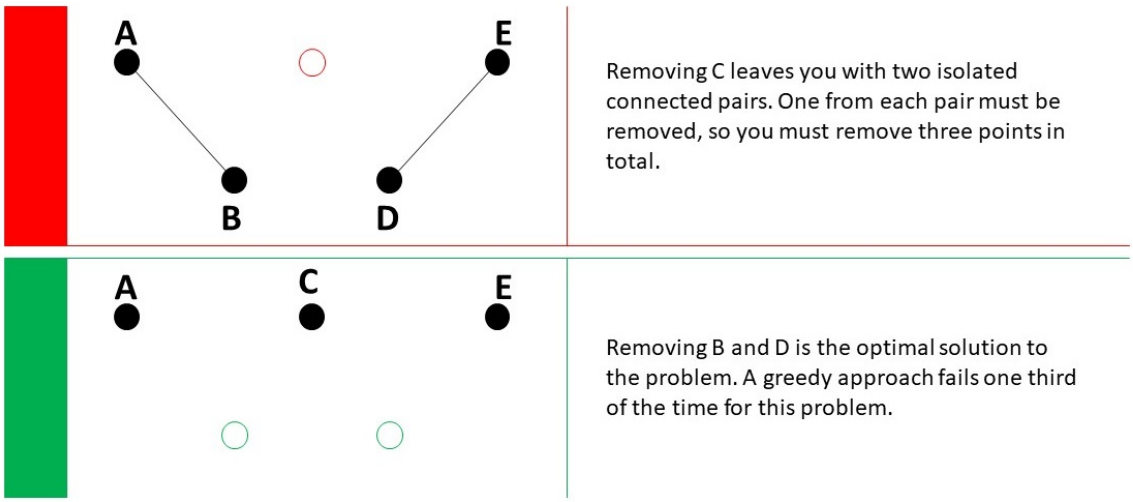


Figure 2: Demonstrating the failure of a greedy approach

Two final observations are worth mentioning before stating the rules that can solve this problem. 1) So long as we are consistent, relabelling a representation of a problem will not change the set of optimal removal sets or optimal stubborn sets in any meaningful way - there will be the same number of optimal sets each of the same cardinality. 2) If we have identified a subset of one of the optimal removal sets and consider the rump problem with the points identified in that subset removed, then the union of the optimal removal set of this latter problem with the aforementioned subset will be an optimal removal set of the original problem. In simple language this means we can identify some points that should be removed, remove them, and then consider the problem that remains. To see why this is the case, we can use a proof by contradiction. If an optimal removal set of the stump problem had a higher cardinality than the set difference of the optimal removal set of the original problem and the subset, then that optimal removal set could not be optimal because there exists a set with lower cardinality that solves the problem, to wit,

the set difference. If an optimal removal set of the stump problem had a lower cardinality than the set difference of the optimal removal set of the original problem and the subset, then the optimal removal set of the original problem cannot be optimal, since there exists one of lower cardinality formed by the union of the subset and the optimal removal set of the stump problem. An optimal removal set must always exist (since you can at the very least always remove pairwise correlations by eliminating all but one variable[3]), so this proof is not trivial and we can attack a problem bit by bit.

# 3 The Logical Approach

## 3.1 The Subset Rule

*The subset rule states that it is optimal to successively remove points that share at least all the connections of another.*

'Share at least all the connections of another' means that the point has an associated list of variables in the first representation that is a superset of the associated list of another point. Having established that we can proceed sequentially, the subset rule follows from two premises:

P1: For any two connected points one or the other must feature within an optimal removal set.

P2: If a point has at least all the connections of another, then if the latter features in an optimal removal set, so must the former.

The first premise can be proved by contradiction. If neither point featured in an optimal removal set, then there would still exist a connection and the optimal removal sets cannot be optimal. The second is the case since the rump problem formed by dropping the point with fewer connections is functionally the same[4] as the rump problem formed by dropping the point with more connections and then adding some number of connections equal to the difference in the number of connections between the point with greater connections and the point with fewer.[5] This is illustrated in figure 3, which compares the rump problem formed by removing point A in figure 1 versus removing point B and adding a connection. The preliminary label swap is just to make the comparison all the more obvious.

Clearly, a rump problem that is equivalent (or has fewer connections) to another cannot have optimal removal sets of higher cardinality, hence the second premise.

## 3.2 Bifurcation

The subset rule runs into trouble if it cannot find any satisfactory points. Fortunately, we can use similar reasoning as in the subset rule to create a fairly efficient branching

[3] In the case of trivial problems with no points or no connections, the optimal removal sets are the empty set.

[4] Functionally same in the sense that the cardinality of optimal removal sets of the rump problem are the same.

[5] In the case of equivalent points, the number of connections added is thereby zero.

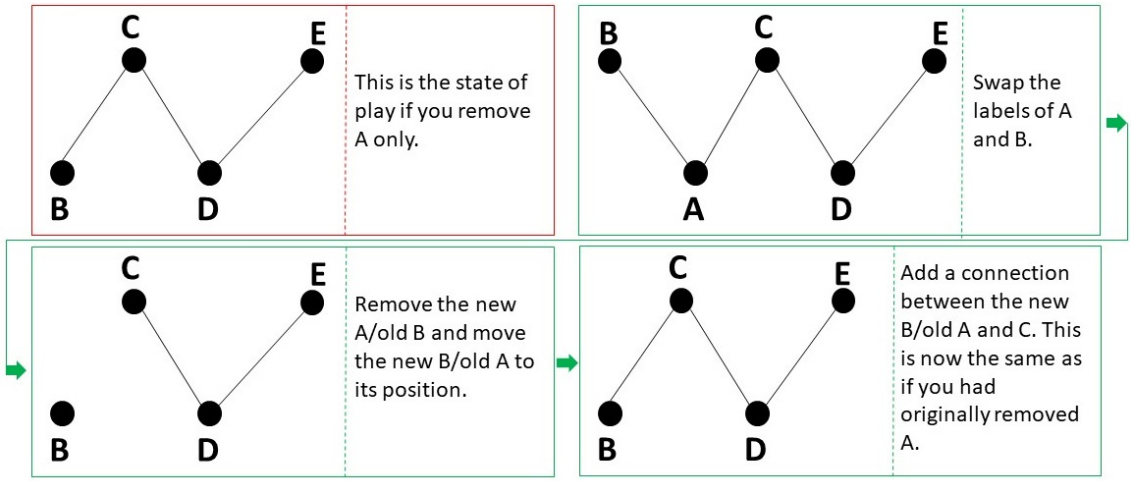


Figure 3: Comparing removing point A and point B in the example problem

mechanism. For any point, either it or all the points it is connected to must feature in some optimal removal set. Again, if this were not true, then a connection would still exist between it and one of the points it is connected to, which is not allowed. We can therefore create two branches when the subset rule fails and the problem remains unsolved. The first branch removes the 'greedy point' with the greatest number of connections, whereas the second branches removes the complimentary set of points. If we have so far removed a set of points that is a subset of one of the optimal removal sets, then either the addition of the greedy point or its compliment will still leave us with a subset of one of the optimal removal sets. We can then proceed with the subset rule on each of these branches until we need to bifurcate again, and so on. As soon as a branch that solves the problem (ie. removes all connections) is the shortest branch under consideration, it must be an optimal removal set, since one of the branches under consideration must end up being an optimal removal set, and any branches that are still not sufficient to solve the problem will not get any shorter.

# 4 Reformulating The Subset Rule for Computational Efficiency

The subset rule cannot be applied all at once, since doing so would remove unnecessary points. To see this, consider the rump problem in the top pane of Figure 2. Removing all points that share at least all the connections of another all at once would result in the removal of all points, when we only need to remove two. We can deal with this issue by splitting the subset rule into two rules, one which deals with points that have exactly the same connections, and one that deals with points that have strictly more connections. So long as we can show that we can achieve the same result from a (successive) application of the subset rule, we can be assured that optimality is maintained.

## 4.1 Rule 1: Congruence Rule

*The congruence rule states that for all sets of multiple points that have the exact same connections, we can remove all but one point*

If we consider a problem in which only one such 'congruence set' of points exists, it is easy to see that an application of the congruence rule is equivalent to an application of the subset rule in which a point from the congruence set is successively removed until only one remains. We can then generalise to a problem with multiple distinct congruence sets by noting that no two points within two distinct congruence sets can possibly be connected. If they were, then every point in one congruence set must be connected to every point in the other, forming one congruence set. Consequently the removal of all but one point within one congruence set cannot make another congruence set cease to be a congruence set, and therefore we could achieve the congruence rule with some successive application of the subset rule even with multiple distinct congruence sets. Therefore, the congruence rule will identify a subset of an optimal removal set, just like the subset rule.

## 4.2 The Proper Subset Rule

*The proper subset rule states that all 'PS points' with all the connections of some other 'PS partner' and at least one more, can be removed.*

Let's consider a *successive* application of the proper subset rule (which itself is an application of the subset rule). Any PS point originally identified before successive removal occurs will only cease to be a PS point if its PS partner is removed or if its extra connections over its partner are removed.

In the former case, a PS partner will only be removed if it is a PS point itself. In that case, the PS partner must itself have a PS partner, and since the relationship between PS points and PS partners is transitive, this new PS partner must be a PS partner of the original PS point. Consequently, no PS point will cease to be a PS point through successive application by virtue of having its PS partner removed, since it will still have a spare PS partner.

In the latter case, a PS point that has its extra connections removed will form a congruence set with its erstwhile PS partner and therefore can still be validly removed by an application of the subset rule once the successive application of the proper subset rule has run its course. A simultaneous removal of all PS points can therefore be achieved by an application of the subset rule, and is therefore optimal.

## 4.3 The Congurence Rule and Proper Subset Rule Together

The set of PS points not in a congruence set that can be identified before and after an application of the congruence rule is the same set. We can use the same reasoning as above to see this. If every point in a congruence set is a PS partner, then removing all but one will still leave a PS partner. Conversely, if the points in a congruence sets are the extra connections of a PS point, it will still have an extra connection once all but one are removed.

It is possible that the points within a congruence set are also PS points, so the set of points identified for removal by the congruence rule and proper subset rule might have

some overlap, but so long as this is dealt with so as to not cause running errors, we can be confident that removing unique points identified by these two rules could be achieved by the subset rule and is therefore optimal.

# 5 Conclusion

Looking back at the summary of the algorithm given in the introduction, we can now clarify that the 'swathes of points' being removed are identified by adding together the sets of points identified by the proper subset rule and those identified by the congruence rule. Since removing this joint set can create new congruence sets, the algorithm uses a while loop to continue this removal procedure until no more points can be found. The 'branches' are created by the bifurcation technique described in section 3.2 and each branch is extended and bifurcated until the shortest branch under consideration solves the problem.

To conclude, this document has introduced the correlation eliminator and isolator algorithms, and summarised how each operates. The problem that these algorithms (especially the isolator algorithm) solve is distilled and represented, and then a logical solution is explained. Finally, an element of the logical approach is reformulated for computational efficiency.