*Voting Simulations*

Summary

I used Python to simulate different voting systems to see how frequently they picked the candidate that maximizes voter happiness. Score voting did so most often, Approval second most, Borda Count third, Instant Runoff fourth, and Plurality least often. For this reason, I suggest using Score voting in most situations.

Definitions

* **Plurality Voting / First Past the Post:** Each voter picks one candidate. The votes are added to each candidate's total, and whoever has the most votes wins. This is the system currently used for most US elections.
* **Ranked Choice Voting:** This refers to how voters fill out the ballot, but not how the ballots are counted. Each voter ranks candidates in order from favorite to least favorite. Instant Runoff Voting and Borda Count are both versions of RCV.
* **Instant Runoff / Alternative Voting:** A form of RCV. In the first round, everyone's first choice gets one vote. After all are counted, the candidate with the fewest votes is eliminated. The voters who had that candidate ranked first then transfer their votes to their next favorite. The next lowest is then eliminated. This continues until one candidate is left standing, who is the winner. Australia currently uses this method to elect representatives.
* **Borda Count:** A form of RCV. The highest ranked candidate is given the most points. Second place is given fewer points, third fewer still. Once all points from all voters are totaled, the candidate with the most points wins. This method is used to elect Baseball Hall of Fame-ers.
* **Approval Voting:** Each voter may approve of as many of the candidates on the ballot as they wish. If a candidate is approved of, they get a point. The winner is the candidate with the most points. Voters can only approve each candidate once.
* **Score / Range Voting:** With score voting each voter gives a number between two values (say, 1-10) representing how much they like each candidate. Candidates are given points in proportion to how highly they were rated. Candidates with the most points win. Most surveys use a similar method, minus the deciding on a winner.

Designing The Simulation

Generating Voters

For this simulation, every voter or candidate was represented by a point on a 2D plane, with the X value of the point being their opinion on one political issue and their Y value on another. I chose a 2D plane to give more detail than a 1D line but create less work than a higher dimensions. In real life, voters may be more complicated than a 2D graph, but it's good enough for my purposes.

To generate each voter and candidate I randomly chose values between -10 and 10 with two decimals (e.g., a voter might get (3.25, -8.90) as a position). With completely random values the voters are evenly distributed across the 2D space.

Chart, scatter chart

Description automatically generated

Liking Candidates

When I began to implement the voting systems, it quickly became apparent that I needed a way to determine how much each voter likes each candidate. Since the voters are represented in 2D space, it makes sense to use a measurement of distance to determine how much they like the candidates. The further a candidate is from a voter, the less that voter likes that candidate.

Two possible options occurred to me for how to measure distance:

* **Straight-Line**: Similar to drawing a straight line between points. The formula for this is:
* **End-to-End:** Add the X and Y differences between the voter and candidate. The formula for this is:

Note: The formula here is flawed, which is discussed in the limitations section.

For example, a voter at (0,0) would disagree with a candidate at (7,7) less than one at (0,11) if Straight-Line was being used, but not if End-to-End was being used. The green area in the diagram below would be included in both methods, but the red area would only be included if Straight-Line was used.

A picture containing chart

Description automatically generated

Since I couldn’t decide which to use, I opted to test both.

Once that was done, I implemented each voting system to determine who would win in each. That looked something like this:

Text

Description automatically generated

Which System Won?

At this point I had no way of comparing how well each voting system did, just which option they chose. For this reason, I added a measure of overall voter satisfaction by finding each candidate’s average distance from the voters. The candidate that should win is whoever has the smallest average distance, meaning the highest voter satisfaction.

For these early tests, I used 1000 voters and 5 candidates. Candidates are ordered 1-5, and the candidate positions, voter satisfactions, and metrics for each voting system display them in order. They looked something like this:

|  |  |
| --- | --- |
|  |  |

Notice that all voting systems in this example chose the same option, Candidate 1, who minimized the distance from voters/maximized satisfaction. However, there’d be nothing interesting to discuss if they all got the same answer all of the time. Take a look at another round of simulation:

|  |  |
| --- | --- |
|  |  |

How well a voting system does can be gauged by how closely the overall scores and rankings mirror voter satisfaction, which is the last bar chart on the bottom right. Each candidate is color-coordinated between the scatterplot and bar charts for ease of comparison. The ideal candidate is outlined with a dark blue border.

This time all voting systems save Plurality and IRV chose Candidate 5, who was best. Plurality chose Candidate 4, who was third best, and IRV chose Candidate 2, who was second best. Some voting systems frequently get a less-than-optimal candidate.

Repeated Trials

Manually looking at examples can give a *sense* of the overall performance of voting systems, but it would be more reliable to do multiple tests and let the program tally how often each gets the correct answer. For that reason, I implemented a function that would let the user do as many tests as desired, such as in the following set of trials:

Text, letter

Description automatically generated

Notice the how well each performs. In general, the order of their performance was Score, Approval, Borda, IRV, and Plurality last. This confirms what is often said about these voting systems. Looking at the following bar chart, notice that the degree of similarity to Voter Satisfaction is correlated with their performance:

Chart, scatter chart

Description automatically generated

Ideologies and Parties

One final detail that might be important to making this simulation a little more analogous to real life would be the way voters are distributed. One of the first things you might notice while looking at the scatterplots above is that they are all evenly spread. Does this seem realistic? In real life, people tend to cluster around political parties or ideologies. For that reason, having clustered ideologies would be preferable. Once that was implemented, the scatter looked more like this:

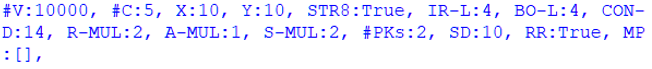
Chart, scatter chart

Description automatically generated

To do this, instead of randomly selecting candidates between two values, I used Numpy’s random module to generate normal distributions which had their peaks at randomly selected points. I also added in the ability to manually control the number of modes, where the modes are or to leave them random, and the size of the standard deviation. Each ideology will produce at least one candidate originating from it.

Simulation Parameters

Now that the model’s complete, lets change some of the parameters to see how they impact performance. Here’s what they all mean:



* **#V:** Number of voters per trial.
* **#C:** Number of candidates.
* **X/Y:** When randomly generating people, determines what the ranges of x and y values they can be found in. When using ideologies, it determines where the peaks of the distributions can lie.
* **STR8:** This value is True if the Straight-Line method of determining distance is used, False if the End-to-End method is used.
* **IR-L:** This is the number of candidates voters are allowed to list on the ballot when they rank people in IRV.
* **BO-L:** This is the same as IR-L, but for Borda Count.
* **CON-D:** This stands for ‘consideration distance’. Outside this value, voters will dislike a candidate so much that they will never vote for them, no matter how bad the alternatives.
* **R-MUL:** This stands for ‘rank multiplier’. It’s a value that gets multiplied by CON-D for certain voting systems to allow the consideration distance to be different for different systems. In this case, R-MUL applies to the two forms of ranked-choice voting systems, Borda and IRV.
* **A-MUL:** The same as R-MUL, but for Approval voting.
* **S-MUL:** The same as R-MUL and A-MUL, but for Score voting.
* **#PKs:** Number of peaks/ideologies/parties. Since candidates are chosen from each peak, this will override #C if it is larger.
* **SD:** Standard deviation of the ideologies’ peaks.
* **RR:** Stands for ‘random remaining’. If #C is greater than #PKs, the program will need to generate more candidates than there are peaks. If it’s True, they will be chosen completely at random. If it’s False, they will be randomly selected from one of the ideologies.
* **MP:** Stands for ‘manual peaks’. If the locations of ideologies’ peaks are manually chosen, they are listed here.

1. Number of Candidates

First, I tried changing the number of candidates. The following graphs have voting systems color coordinated. The heights of the bars represent the portion of the tests each style got right.

|  |  |
| --- | --- |
| Low: 2 Candidates | Medium: 5 Candidates |
|  |  |
| High: 10 Candidates |  |
|  |  |

Notice that as number of candidates increases, Plurality quickly dropped off by a significant amount. This is one of the reasons why Plurality incentivizes a two-party system. Score and Approval didn’t noticeably change in accuracy when the number of candidates increased. This suggests that they are highly resistant to vote splitting.

IRV and Borda were intermediate. While both decreased less than Plurality did, neither maintained as high of performance as Score or Approval. Borda also did a better job at maintaining its performance than IRV.

2. Number of Voters

After that, I tried changing the number of voters. For these tests I kept the number of candidates constant at 5.

|  |  |
| --- | --- |
| Very Low: 5 Voters | Low: 50 Voters |
|  |  |

|  |  |
| --- | --- |
| Medium: 500 Voters | High: 5000 Voters |
|  |  |
| Very High: 500,000 Voters |  |
|  |  |

In general, most voting systems do slightly worse when the number of voters is lower. However, Approval among all others decreased in quality the most with a small number of voters, becoming as bad as Plurality in the worst cases. This puzzled me at first, until I realized that my program wasn’t well equipped to deal with ties. Looking at the number of ties in each test, in order of largest to smallest number of voters, Approval had 0, 0, 2, 37, and 99 ties. Borda Count and Plurality also appear to suffer from a large number of ties as number of voters decreases, though not as heavily, possibly because they didn’t have as far to fall.

Since I didn’t anticipate this, the program essentially just picks the first tied candidate in the list when one occurs, which has a success rate no better than randomly picking which of the two or more tied candidates wins. Another limitation to my program is that because IRV is counted slightly differently than the rest (doing multiple rounds rather than just one) the program does not accurately describe how many ties occurred in each round. For this reason, the 0 ties listed for IRV should be ignored. Most likely, the rate of ties in IRV is higher than any of the others due to having multiple rounds, giving it more opportunities to tie.

3. Consideration Distance

When writing the code for Approval voting, I had to set a certain distance away for how close a candidate has to be in order to be approved of. The diagram below illustrates how making the radius of selection changes who a voter will or won’t approve of. The orange dots represent the candidate, green and red dots who the voter will and won’t choose, and blue circles their approval distance.

A picture containing text, vector graphics

Description automatically generated

Which distance a voter is comfortable choosing a candidate within is likely down to their individual preferences. It may not be the same for every voter and is unfortunately not something designers of voting systems can control directly.

|  |  |
| --- | --- |
| Low: 1 | Medium: 5 |
|  |  |
| High: 10 | Very High: 20 |
|  |  |

All voting systems performed poorly when the consideration distance was low. This is likely because there is very little information for the voting systems to work with. A smaller radius of consideration means that may voters will not list any candidates, or will list very few, and the voting systems won’t be able to gauge their opinions.

For Approval in particular, having a very high consideration distance also had negative effects on accuracy. This is likely because when voters pick many candidates, it can’t determine their preferences. If a voter approves of every candidate on the ballot, it has the same effect on their relative performances as if the voter had put none at all.

Ties are common for all when the consideration distance is low, and very common for Approval when consideration distance is high.

For Score, Borda, and IRV, a larger distance usually made them better. While these systems also list multiple candidates, they distinguish between them in how highly they are ranked or scored. This allows those voting systems to determine a preference, rather than leaving all listed candidates tied. For this reason, I recommend that when using any of these systems, voters opine about as many candidates as possible.

4. Rank Limit

For ranked voting systems, voters can’t list an infinite number of candidates. Often a limit is set on the number of rankings each is allowed to do. Approval and Score were removed from these tests since rank limit is not relevant to them. Keep in mind that there were 10 candidates in all these tests, rather than the usual 5.

|  |  |
| --- | --- |
| Very Low: 1 Rank | Low: 3 Ranks |
|  |  |

|  |  |
| --- | --- |
| Medium: 5 Ranks | High: 10 Ranks |
|  |  |

Notice that when only one candidate is ranked, both ranked voting systems perform no better than Plurality. As number of ranks increase, both improve.

Its also worth noting that Borda improves more quickly than IRV. IRV needs to have nearly all candidates ranked to perform at its full potential, but Borda was close to maximum performance with only half the candidates ranked.

This suggests that when creating ranked voting systems, the limit on the number of candidates that can be ranked should be as high as possible. This also applies to Score voting.

5. Distance Metrics

As mentioned earlier, there are two ways of determining voter opinion: the Straight-Line distance, or the End-to-End distance. Does this affect the relative performance of the voting methods?

|  |  |
| --- | --- |
| Straight-Line, Consider Dist. 10 | End-to-End, Consider Dist. 10 |
|  |  |

While it looks like End-to-End distance metrics make everything perform worse, especially Approval, remember that adding distances End-to-End tends to result in longer distances between voters and candidates, meaning that the relative size of consideration distance is now shorter. When adjusting for this, performance becomes:

|  |  |
| --- | --- |
| Straight-Line, Consider Dist. 14 | End-to-End, Consider Dist. 14 |
|  |  |

Notice that the situation of Straight-Line and End-to-End reverse when a longer consideration distance is used. This seems to indicate that as long as the consideration distance is appropriate for the measurement being used, changing the distance metric has no effect on overall or relative performance of the voting systems.

6. Party Polarization

Polarization occurs when there is a large difference between the major political parties/ideologies with little overlap between. Examples of this difference are shown below.

|  |  |
| --- | --- |
| Low Polarization, Standard Dev. 7 | High Polarization, Standard Dev. 3 |
|  |  |

Notice that in Plurality almost every voter voted within their own party. Centrist candidates like Blue tend to be ignored, even though Blue maximized voter satisfaction more than any other candidate. Additionally, since Blue is not many voters’ favorite choice, IRV eliminated her early on in the second example, leaving her unavailable for later rounds where she most likely would have won.

Let’s see if these polarizations change anything about the overall performance of the voting systems. Modifying polarization level can be done by increasing or decreasing the standard deviation of the parties’ distributions. The larger it is relative to the distance between the parties’ modes, the lower the polarization.

|  |  |
| --- | --- |
| High Polarization, Standard Dev. 3 | Medium Polarization, Standard Dev. 5 |
|  |  |
| Low Polarization, Standard Dev. 10 | Very Low Polarization, Standard Dev. 20 |
|  |  |

Polarization didn’t appear to affect the overall performance of the voting systems too much. Interestingly, a moderate level of polarization actually seemed to make IRV and Plurality perform better than usual, where IRV was on par with Borda, and Plurality close behind.

Approval performed worse than usual both with a low and high level of polarization. This is likely because the overall spread of the data was very small or very large when standard deviation was small or large, causing the relative size of the consideration distance to be too low and too high.

|  |  |
| --- | --- |
| Very Low Polarization: Standard Dev. 15, High Consider Dist. 23 | |
|  |  |

Once a higher consideration distance was used, Approval voting recovered its performance.

Key Takeaways

1. In their best-case scenarios, the quality of voting systems are: Score, Approval, Borda, Instant Runoff, and Plurality.
2. While Plurality performed the worst, it was better than random. In a 5-candidate election, random selection would get the optimal candidate 20% of the time, but in most cases plurality was closer to 40 or 50%.
3. Approval voting suffers the most from ties. Tiebreakers may need to be used, especially when the number of voters is small.
4. Approval voting is very sensitive to the consideration distance of the voters, which must be a moderate value.
5. All systems get worse when the number of candidates increase and the number of voters decrease, but Score and Borda are the least affected, followed by Approval, then IRV, then Plurality most of all.
6. When polarization is high, most systems don’t see any effect. IRV and Plurality actually do better when there’s a moderate level of polarization.

Limitations

* I didn’t incorporate any form of strategic voting. In real life, voters will notice the spoiler effect and vote accordingly. This can potentially change the effectiveness of the election systems, and different systems may be affected differently.
* I didn’t measure how often the voting systems got the second or third best choice. I could have also included average voter satisfaction with each voting method.
* This simulation only used two dimensions. Voters may be more complicated than this in real life, and so simulating higher dimensions would be useful.
* The code written here is not well optimized enough to do very many trials at the scale of American presidential elections. While I find it unlikely that much will change, it would be prudent to test them anyway. This is partly due to the fact that Python is not a fast language, and so hundreds or thousands of trials of hundreds of millions of voters may be out of reach.
* All ideologies or political parties in this simulation were the same size. In real life, third parties are usually much smaller than first parties, and this could potentially change the outcome.
* I didn’t incorporate any method of tiebreaking, nor did I properly count ties for IRV.
* All voters used the same consideration distance. In real life, this would likely vary along a normal curve.
* All voters perfectly estimated their own opinions, and the opinions of the candidates. Personality can often be hard to judge, and sometimes we don’t know as much about ourselves as we think we do. In reality, we make mistakes about what we truly want, as well as mistakes about politicians’ opinions (not helped by the deceptiveness of some). Rankings and scorings should include some degree of inaccuracy.
* Since writing this I realized the end-to-end method of distance calculation is flawed. This error is reflected in the code. I should have taken the absolute value of each term individually, rather than as a whole, making the proper formula:

The reason this is better is because the end-to-end method is trying to add the differences between the voter and candidates opinions on two issues. If one term were to turn out negative and the other positive, adding them before taking the absolute value would allow them to cancel out, whereas it seems more reasonable to say that more differences between the voter and candidate should decrease the candidate’s approval rating.

Having since rerun one simulation with the proper end-to-end formula, it makes no difference to the relative performances of the voting methods, but it’s worth noting. All other simulations here used the straight-line method, so it shouldn’t affect them, but take the section on end-to-end with a grain of salt.