One-Shot Voting: Machine Learning Approach

Random Forest model and features description

Contents

Scenario	**Name**	**Configuration**	q	q'	q''
A		$q > q' > q''$	TRT	DOM	DOM
B		$q > q'' > q'$	TRT	DOM	DOM
C	LB	$q' > q > q''$	TRT	WLB	DOM
D	SLB	$q'' > q > q'$	TRT	DOM	SLB
E	CMPLB	$q' > q'' > q$	TRT	CMP/WLB	DOM
F	CMP	$q'' > q' > q$	TRT	CMP	SLB

Table 1: 3 candidates scenarios

1 Methodology

In this section we describe the machine learning techniques used in order to build a black box model that accurately predicts votes given the voting configuration (pre-vote poll and preference profile) , voter identification and previous votes. We use a Random Forest classifier which is an ensemble of independent sub-classifiers that vote for a classification using the majority rule. We use feature engineering to generate a variety of features using different statistical aggregations and feature compression for dimensionality reduction. We use a variety of statistical aggregation functions such as: *Average, Median, Standard Deviation, Maximum, Minimum.* Each aggregation is done for every vote action (q, q', q'') which we name class-dependant feature aggregation as described in [1]. We also use other aggregations such as the average vote action where we view the action as a numeral ordered attribute $(q = 1, q' = 2, q'' = 3)$ and the A-ratio features where A is the action name (TRT, WLB, SLB, CMP, DOM) as described in Table 1. In some of the aggregations we ignore scenarios A and B because these scenarios doesn't hold much information about the voter compared to it's actions in the other scenarios (C,D,E,F). The idea is to bring as much information

of the voter from the training set and concentrate it uniformly in each one of the voter's votes. One problem that arise is the large number of the features which hold the information separately. To deal with the "curse of dimensionality" and the sparsity of the data, we use neural auto-encoding to reduce the dimensionality and concentrate it in a smaller number of columns. It is important to mention that sparse data might be useless for any method that relays on statistical significant.

1.1 Features description

The following are the features we use to train the Random Forest model and the outcome of the feature engineering techniques we used.

1. GAP12_pref_poll = $Votes(Q) - Votes(Q')$
2. Action_average - Average action $(1 = Q, 2 = Q', 3 = Q'')$ where $Scenario \in \{C, D, E, F\}$
3. TRT-Ratio = $\dfrac{\text{Number of times action was TRT}}{\text{Number of times TRT action was available}}$
4. VotesPref1PreVote = $Votes(Q)$
5. 1,2,3,...,20 =

   Step A: Generate a set of features as the aggregations of the features: GAP12_pref_poll, GAP13_pref_poll, GAP23_pref_poll using the aggregation functions (average, std, median, min, max, skew, kurtosis) in scenarios {C,D,E,F} for every action $\{Q, Q', Q''\}$. This generates $3 * 7 * 3 = 63$ features.

   Step B: Calculate the differences of each feature in Step A that were generated using average, median, min, max with the corresponding GAPXY_pref_poll features. For example: $GAP12\_pref\_poll - Average(\{GAP12\_pref\_poll | Action = Q' \ \& \ Scenario \in \{C, D, E, F\}\}$ This generates another $3 * 4 * 3 = 36$ features

   Step C: Use Neural Auto Encoder on Step A, B, A-ratios and GAP12_pref_poll, GAP13_pref_poll, GAP23_pref_poll features to compress the features from Step A and

Step B (and replace them) to 20 features (1,2,3,...,20). The Neural Auto Encoder has one layer of $\#features/5 = 21$ nodes with relu activation function.

6. SLB-Ratio = $\dfrac{\text{Number of times action was SLB}}{\text{Number of times SLB action was available}}$

7. PointsPreVote = Points if the winner is according to the pre-vote poll

8. CMP-Ratio = $\dfrac{\text{Number of times action was CMP}}{\text{Number of times CMP action was available}}$

9. WLB-Ratio = $\dfrac{\text{Number of times action was WLB}}{\text{Number of times WLB action was available}}$

10. Action_median = Median action $(1 = Q, 2 = Q', 3 = Q'')$ where $Scenario \in \{C, D, E, F\}$

11. Action_std = Standard deviation action $(1 = Q, 2 = Q', 3 = Q'')$ where $Scenario \in \{C, D, E, F\}$

12. Pref2 = The candidate number of the second preferred candidate.

13. GAP13_pref_poll = $Votes(Q) - Votes(Q'')$

14. VotesPref2PreVote = $Votes(Q')$

15. WinnerPreVote = The candidate number of the winner according to the pre-vote poll.

16. DOM-ratio = $\dfrac{\text{Number of times action was DOM}}{\text{Number of times DOM action was available}}$

17. Pref1 = The candidate number of the second preferred candidate.

18. Scenario = A,B,C,D,E,F determined according to the poll and preference profile of the voter.

19. VoterType = $VoterType(TRT_{ratio}, WLB_{ratio}) = \begin{cases} TRT, & \text{if } TRT_{ratio} \geq 0.9 \\ LB, & \text{if } WLB_{ratio} \geq 0.8 \\ Other, & \text{otherwise} \end{cases}$

20. VotesPref3PreVote = $Votes(Q'')$

21. GAP23_pref_poll = $Votes(Q') - Votes(Q'')$

22. Is_Random =

$$IS\_RANDOM\_VOTER(\#DOM) = \begin{cases} TRUE, & \text{if } \#DOM > 2 \\ FALSE, & \text{otherwise} \end{cases}$$

23. GAP12_poll = $Votes(leader) - Votes(runnerup)$

24. VotesRunnerup_poll = $Votes(runnerup)$

25. GameIndexInSession = The round index in session

26. Pref3 = The least preferred candidate number

27. VotesCand3PreVote = $Votes(C_3)$

28. GAP23_poll = $Votes(runnerup) - Votes(loser)$

29. RoundIndex = id of the vote (unique in all dataset)

31. VotesLoser_poll = $Votes(Loser)$

32. VotesLeader_poll = $Votes(Leader)$

33. VoterID = The Voter ID

34. VotesCand1PreVote = $Votes(C_1)$

35. VotesCand2PreVote = $Votes(C_2)$

36. WorkerId = The worker id in mTurk

37. GAP13_poll = $Votes(leader) - Votes(loser)$

38. UtilitySet = The utility set used for the voter. for example: (20,16,0)

39. Util2 = Utility if Q' is the winner

40. Util3 = Utility if Q" is the winner

41. NumVotes = Number of votes in poll

42. GameConfiguration = The number of rounds (32 or 36)

43. Datetime = Timestamp of the vote

44. VotesCand1PreVote = Number of votes for candidate number

2 RF description

In this section we describe the parameters and configurations of the Random Forest algorithm that we use to build the black-box decision models (RF). We use the *sklearn* package in *Python* to run the Random Forest model in the exact configuration that is shown in Table 3. The libraries used inside *sklearn* are shown in Table 2. Most of the parameters were the default parameters set in the library. In Table 3 we note whether the parameter is used with it's default value or not. Note we varied only one parameter which is the number of trees in the ensemble.

Algorithm	**Library**
Random Forest Classifier	sklearn.ensemble.forest.RandomForestClassifier

Table 2: Algorithms' libraries

References

[1] O. Schulte and K. Routley. Aggregating predictions vs. aggregating features for relational classification. In *Computational Intelligence and Data Mining (CIDM), 2014 IEEE Symposium on*, pages 121–128. IEEE, 2014.

Parameter	**Description**	**Value**
bootstrap	Whether bootstrap samples are used when building trees	True (default)
class_weight	Weights associated with classes	None (default)
criterion	The function to measure the quality of a split	gini (default)
max_depth	The maximum depth of the tree	None (default)
max_features	The number of features to consider when looking for the best split	auto (default)
max_leaf_nodes	If None then unlimited number of leaf nodes	None (default)
min_impurity_decrease	A node will be split if this split induces a decrease of the impurity greater than or equal to this value	0.0 (default)
min_impurity_split	Threshold for early stopping in tree growth	None (default)
min_samples_leaf	The minimum number of samples required to be at a leaf node	1 (default)
min_samples_split	The minimum number of samples required to split an internal node	2 (default)
min_weight_fraction_leaf	The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node	0.0 (default)
n_estimators	The number of trees in the forest	100
n_jobs	The number of jobs to run in parallel for both fit and predict	None (default)
oob_score	Score of the training dataset obtained using an out-of-bag estimate	False (default)
Random_state	Random number generator	np.random (default)
warm_start	Reuse the solution of the previous call	False (default)

Table 3: Random Forest parameters