Predicting the winners of Borda, Kemeny and Dodgson elections with supervised machine learning

Hanna Kujawska, Marija Slavkovik and Jan-Joachim Rückmann

University of Bergen, Norway
{marija.slavkovik,Jan-Joachim.Ruckmann}@uib.no,
Han.Kujawska@gmail.com

Abstract. Voting methods are widely used in collective decision making, not only among people but also for the purposes of artificial agents. Computing the winners of voting for some voting methods like Borda count is computationally easy, while for for others, like Kemeny and Dodgson, this is a computationally hard problem. The question we explore here is can winners of Kemeny and Dodgson elections be predicted using supervised machine learning methods? We explore this question empirically using common machine learning methods like XGBoost, Linear Support Vector Machines, Multilayer Perceptron and regularized linear classifiers with stochastic gradient descent. We analyze elections of 20 alternatives and 25 voters and build models that predict the winners of the Borda, Kemeny and Dodgson methods. We find that, as expected, Borda winners are predictable with high accuracy (99%), while for Kemeny and Dodgson the best accuracy we could obtain is 85% for Kemeny and 89% for Dodgson.

Keywords: Computational social choice · voting · machine learning application

1 Introduction

Voting theory is concerned with the design and analysis of methods that choose a winner for elections. An election is a pair (C,V) where C is a finite set of candidates (also called options or alternatives) and V is a set of voters, each represented as a total, strict, asymmetric order $\succ_i$ over C. As a collective decision making method, voting theory finds its implementation not only in politics, but also in automated decision making [22]. One of the computationally simplest methods is, for example, plurality where the winner is the alternative top ranked by the highest number of voters. It is well documented that with some voting methods, such as Kemeny and Dodgson computing the winners is computationally hard, winner determination in the case of both voting methods is Θ_2^p [2,12,11].

Efficient computing of a representative rank of alternatives is also of interest to recommender systems, specifically in collaborative filtering [20]. Unlike voting theory, which has extensively studied what it means for a collective rank to be representative of individual ranks, in collaborative filtering, the representativeness of a "collective" rank is measured by user satisfaction, which is not always available. Social choice methods, like the Kemeny and Dodgson methods, may have valuable properties [16][1], but it is the

[1] To be fair, the value of the properties that Dodgson satisfies has been disputed [3].

high computational complexity that deters from their use. It is therefore interesting to explore whether some "precision" can be "traded off" for computing time. Clearly this is not of interest where exactly the Kemeny/Dodgson winners are sought, but for situations, such as collaborative filtering, where imprecision can be afforded while gaining "representability" of the collective rank in a classic social choice sense.

We ask whether machine learning, specifically classification with supervised learning, can be used to predict the winners of Dodgson and Kemeny elections.

Classification is the problem of assigning a label to a given data point , using a set of labeled data points as examples, called a *training set*. A data point is a vector of values, where each value is associated with a *feature*. Features are used to build a factorized representation of an entity or event. A label, or class, of a data point is the feature we are trying to predict. Training the classifier over a dataset is the process of building a model that when presented with feature values of an unknown instance (example or datapoint) on input, outputs a classification (or a label) for that instance.

We are motivated by the work of Devlin and O'Sullivan [7] who treat satisfiability (SAT) of a Boolean formula[2] as a classification problem. They do this by labeling examples of formulas as satisfiable or unsatisfiable (by having calculated them) and using then these examples as a training data set to build a model that predicts the satisfiability of a formula. They accomplish 99% accuracy for hard 3-SAT problems, and accuracy in excess of 90% for "large industrial benchmarks".

For a given set of candidates C and number of voters $|V|$, it is possible to calculate all the profiles of voters that can occur, that is combinations (with repetition) of strict, total, asymmetric orders over C. For a nontrivially large C and $|V|$ this number can be very big. Can we then, calculate the election winners for some of these profiles, use the so calculated winners as a label for the profile, and use these labeled profiles as a training data set to build a winner prediction model? This is the question we address here. We use a data set of 12360 profiles for 20 candidates and 25 voters and for each of this profiles we calculate the Borda, Kemeny and Dodgson winners.

We hypothesize that the Borda winners can be predicted with high accuracy and use Borda as a kind of "benchmark". We test the predictability of Kemeny and Dodgson winners, as these are computationally hard to calculate. We would like to point out that, it was shown in [18] that the class of scoring rules, of which the Borda method is a member, is efficiently probably approximately correct (PAC) learnable.

The first problem of feeding profiles as training examples to a machine learning algorithm is to represent the voting profiles as data points, i.e., use factorized representation of collections of preference orders. There are different ways in which the *factorization* of profiles can be accomplished and the way a profile is factorized can affect the classifier performance. We consider three different factorizations and analyze their fitness for Borda, Kemeny and Dodgson respectively.

Voting rules often are irresolute, namely, for some elections they identify more than one winner in a tie. We intend to use the winner as a label that needs to be predicted, however to do this we need to address the problem of ties. We break ties using a lexicographic tie-breaking, which is common in the voting literature, to choose a label for an

[2] The satisfiability problem SAT is the problem of deciding whether a given a Boolean formula is admits a truth assignment to each of its variables such that the formula evaluates true.

irresolute profile. However we find out that ties do matter in the accuracy of predicting winners.

To find the right methods for classification, we explored the large pool of all available machine learning classification methods in the scikit-learn library[3] through a process of trial and error. We lastly settled using on ten different classifiers and compared their performance.

Code and datasets for this paper are given at https://github.com/hanolda/learning-election-winners.

This paper is structured as follows. In Section 2 we introduce the basic concepts and definitions from voting and machine learning. In Section 3 we give an overview of the data sets we used in our experiments. In Section 4 we present how profiles can be represented as data sets for supervised machine learning. In Section 5 we describe our experiments, results and evaluation methods, while in Section 6 we discuss the experiments' outcomes. Related work is discussed in Section 7. Lastly, in Section 8 we make our conclusions and outline directions for future work.

Our contribution is twofold. Our results in predicting Kemeny and Dodgson winners are promising, but admittedly explorative. We have established feasibility of the approach and are thus opening the possibility to use machine learning for predicting winners in voting and social choice theory. Once learned, a model for a particular size of candidate set and voter set, can be reused for any election of that "size". Predicting the outcomes of functions that can be computed, however computationally inefficient, allows us to check at any point how well the predictor is performing which opens exciting opportunities for predicting instead of calculating voting outcomes when precision is not critical, for example, for the purposes of collaborative filtering, or in pre-election estimation of election results in politics.

2 Background

We first introduce the basic definitions and concepts from voting theory and supervised machine learning.

2.1 Voting theory

An election is a pair (C,V) where C is a finite set of alternatives (or candidates) $C = \{c_1, c_2, \ldots, c_m\}$ and V is a finite collection of agents or voters. Each voter i is represented with her *preference relation* i.e., a strict, complete, transitive and antisymmetric order $\succ_i$ over the set C of alternatives. The top-ranked candidate of $\succ$ is at position 1, the successor at position 2, while the last-ranked candidate is at position m. A collection of preference orders $V = (\succ_1, \ldots, \succ_n)$ is called a preference *profile*. A voter i prefers candidate c over candidate c' iff $c \succ_i c'$. A voting rule F is a mapping from an election E to a non-empty subset of C called winners of the election.

The Borda method, or Borda count, is *a positional scoring rule*. Each candidate in C is associated with a score that is obtained by the position that candidate has in the

[3] https://scikit-learn.org/stable/

preference orders of the profile. A candidate $c \in C$ receives $m-1$ points from each $\succ_i$ where it is top ranked, $m-2$ points from all $\succ_i$ where it is second ranked, and so on, receiving 0 points when it is the last ranked alternative. Borda winners are the candidates that have a maximal sum of points, which is called Borda score.

The Kemeny method is a *distance-based rule*. For two preference orders $\succ_i$ and $\succ_j$ we can define the *swap* distance, also called Kendall Tau distance, as the minimal number of pairwise swaps required to make the two orders the same. Formally we can define the swap distance d between two orders $\succ$ and $\succ'$ over a set of candidates C with $c_i, c_j \in C$:

$$d(\succ,\succ') = |\{(c_i,c_j) : (c_i \succ c_j \wedge c_j \succ' c_i) \vee (c_j \succ c_i \wedge c_i \succ' c_j)\}|. \tag{1}$$

The collective preference order of a profile V is the order $\succ$ for which the sum of swap distances from $\succ$ to each $\succ_i \in V$ is minimal. This collective preference order is called a Kemeny ranking of election E. Kemeny winners are the top-ranked alternatives in a Kemeny rank (there could be more than one rank "tied"). Formally we can define the Kemeny method as follows. Let $\mathcal{C}$ be the set of all total, strict and antisymmetric orders that can be constructed over a set of alternatives C.

$$\text{Kemeny}(V) = \underset{\succ \in \mathcal{C}}{\text{argmin}} \sum_{\succ_i \in V} d(\succ,\succ_i) \tag{2}$$

Before defining the Dodgson method, we need to introduce the concept of Condorcet winner. The Condorcet winner of an election E is the candidate that defeats all other candidates in a pair-wise comparison. Condorcet winners do not exist for every election, but when they do exist, they are unique. Let $\mathcal{V}$ be the set of all profiles for $|V|$ agents that can be formed from $\mathcal{C}$. The Dodgson winner of a profile V is either its Condorcet winner when it exists, or the Condorcet winner of a profile V' which is obtained from V by a minimal number of adjacent swaps. Formally

$$\text{Dodgson}(V) = \text{Condorcet}(\underset{V' \in \mathcal{V}}{\text{argmin}} \sum_{\substack{\succ_i \in V \\ \succ'_i \in V'}} d(\succ_i,\succ'_i)), \tag{3}$$

where V is the set of profiles that have a Condorcet winner and d is the swap distance defined in (2).

Example 1. Let us consider an election E with four candidates $C = \{a;b;c;d\}$ ($|C| = 4$) and $|V| = 7$ voters. Table 1 presents the preference profile of the V voters. Each row represents the preference order of a subset of voters, where the first column is the number who voted with this preference order, and the following column is the order of the vote. For instance, 3 voters have the preference order: $a \succ b \succ c \succ d$.

The Borda winner is b, the Kemeny winner is a, and the Dodgson winner is b. This profile does not have a Condorcet winner because a defeats b and c but not d; b defeats only d, c defeats only d and d defeats only a, in a pairwise comparison.

number of voters	preference order
3 voters	$a \succ b \succ c \succ d$
1 voter	$d \succ b \succ a \succ c$
1 voter	$d \succ c \succ a \succ b$
1 voter	$b \succ d \succ c \succ a$
1 voter	$c \succ d \succ b \succ a$

Table 1: Preference profile example.

2.2 Machine learning

We now introduce the machine learning methods we use in our experiments. These are: XGBoost, Linear Support Vector Machines (SVM), Multilayer Perceptron and regularized linear classifiers with stochastic gradient descent (SGD). These approaches were chosen through a process of trial and error that considered all available machine learning classification methods in the scikit-learn library[4].

Support Vector Machines (SVM). Conceptually, a data point, for which all feature values are real numbers, can be seen as a point in hyperspace. Binary classification would then be the problem of finding a hyper-plane that separates the points from one class from the points of the other class. SVM's find this hyper-plane by considering the two closest data points from each class. Since not all datasets can be separated with a hyper-plane, SVM's use *kernels* to transform the dataset into one that can be split by a hyper-plane. Both linear and nonlinear kernels can be used. The efficiency of a machine learning method can be improved by tuning the so called hyper-parameters of an SVM classifier. The SVM we used provides one hyper-parameter to tune: cost of miss-classification of the data on the training process.

Gradient Boosted Decision Trees (GB) are among the most powerful and widely used models for supervised learning. Predicting a label can be done with a decision tree built using the training data. To increase the prediction performance, GB builds an ensemble of decision trees in a serial manner: each new tree is built to correct the mistakes of the previous one. GB's offers a wide range of hyper-parameters that can be tuned to improve prediction performance, among else the number of trees ($n\_estimators$) and $learing\_rate$, which controls the degree to which each tree is allowed to correct the mistakes of the previous trees.

Multilayer Perceptrons (MLP) are feed-forward neural networks. MLPs are often applied to supervised learning problems: they learn to model the correlation (or dependencies) in two phases process: *forward* pass - the training data flow moves from the input layer through the hidden layers to the output layer (also called the visible layer). There, the prediction of the output layer is measured against the target labels. The error can be measured in a variety of ways, e.g. root mean squared error (RMSE). In the *backward* pass, $backpropagation$ is used to make model parameters, i.e. *weigh* and *bias* adjustments relative to the error. That act of differentiation, based on any gradient-based optimization algorithm, gives us a landscape of error. During the convergence state, the parameters are adjusted along the gradient, which minimize the error of the model.

Regularized linear classifiers with stochastic gradient descent (SGD) SGD is very efficient approach in the context of large-scale learning. For classification pur-

[4] https://scikit-learn.org/stable/

poses, regularized linear classifiers use a plain stochastic gradient descent learning routine which supports different regression *loss functions*, that measures the model (mis)fit and the *penalties*, regularization term that penalizes model complexity. SGD is fitted with the training samples and the target values (class labels) for the training samples and for each observation updates the model parameters: *weights* and *bias* (also called offset or intercept). A common choice to find the model parameters is by minimizing the regularized training error.

3 Datasets and preprocessing

We originally intended to use datasets from the `preflib.org`, however we found them unsuitable for various reasons such as there were too few candidates, or too few profiles in the data set. We used a dataset of 361 profiles of ranked lists of music tracks (songs) from Spotify[5] as a basis to generate a high-dimensional dataset of 12360 profiles. The such obtained datasets consist of profiles for $|C| = 20$ candidates with $|V| = 25$ voters per profile.

The Spotify dataset consists of daily top-200 music rankings for 63 countries in 2017. Ranked are 20 music tracks (songs) described by *position*, *track name*, *artis*t, *streams* and *URL*. *A single voter's* preference order represents one ranking for one day in one country. We considered 25 countries on 361 days, as some days had to be removed due to preference order incompleteness. We also only considered profiles that have a unique winner under the three voting methods we studied, as is was our intention to not consider ties.

Analyzing the Spotify dataset, however, we observed that the profiles' labels are not uniformly distributed, namely not all candidates are winners of an approximately equal number of the profiles in the dataset. Specifically, the Spotify dataset contains many profiles where the winner is candidate number 16 or candidate 18. We handled this issue by generating a synthetic dataset with the same number of candidates $|C|$ and voters $|V|$ as in Spotify. In the newly built dataset, we kept only those profiles that ensured class-balanced output, i.e., the same number of profiles for each possible winner. The synthetic dataset extends the Spotify dataset into a total of 12360 profiles, which were generated by creating permutations of $|C| = 20$ alternatives and joining them into $|V| = 25$ combinations with repetitions.

For the synthetic dataset we further created three datasets: separately for Borda, Kemeny and Dodgson winners. Each dataset consists of preference ranks (the same for all voting methods) along with labels denoting the winner, as per voting method. For Borda we used all of the 12360 labeled profiles, synthetic and from Spotify. To label the profiles with Kemeny and Dodgson winners, we calculated the winners using DEMOCRATIX[6][5]. DEMOCRATIX was not able to process all of our 12360 profiles. For Kemeny we were able to label 10653 profiles, and for Dodgson we had 11754 labeled profiles.

We split the dataset into a training, validation and testing sets ($70/15/15[\%]$), using the *Stratified ShuffleSplit* cross-validator from the Model Selection module of *scikit-*

[5] https://spotifycharts.com/regional

[6] `http://democratix.dbai.tuwien.ac.at/index.php`

learn library[7], which creates a single training/testing set having equally balanced (stratified) classes. Table 2 presents the train, validation and test split of profiles in the generated dataset. The detailed final distribution of total and training samples per candidate is presented in Table 3.

Dataset	Training set	Validation set	Test set
Borda	6666	1429	1429
Kemeny	6921	1484	1483
Dodgson	7362	1578	1578

Table 2: No. profiles for training, validation and test.

Candidate	as Borda winner	in training	as Kemeny winner	in training	as Dodgson winner	in training
1	483	338	597	418	483	338
10	358	250	299	209	551	386
11	481	337	419	293	511	358
12	307	215	322	225	460	322
13	491	344	397	278	542	379
14	544	381	401	281	519	363
15	522	365	387	271	499	349
16	404	283	381	267	1156	809
17	387	271	360	252	206	144
18	479	335	496	347	513	359
19	513	359	585	409	522	365
2	497	348	613	429	515	361
20	553	387	598	419	500	350
3	494	346	563	394	502	352
4	550	385	612	428	489	342
5	453	317	565	396	501	351
6	475	332	522	365	535	375
7	508	355	633	443	512	358
8	484	339	590	413	513	359
9	541	379	548	384	489	342

Table 3: Borda, Kemeny and Dodgson winner distribution.

4 Factorization of profiles

A profile of votes is a list of total orders, typically encoded as a nested list of ordered lists representing the voters. To be able to use profiles as training data in ML algorithms, we need to find a way to model the profiles using factorized representation. There are several ways in which this can be done and the choice of representation can have a substantial impact on the winner prediction. We explored three different approaches. The approaches differ in what is considered a *feature* of a voting profile.

4.1 Labeling profiles

Each of the voting methods we consider, Borda, Kemeny and Dodgson, admit non-unique election winners. Namely, more than one candidate can appear in a tie as a winner. We use supervised learning with the profile of votes as a data point and the winner for a given election as the class or label for that profile.

Typically, voting methods are accompanied by tie-breaking mechanisms. There are numerous approaches to tie-breaking including randomly choosing one among the winners or pre-fixing an order over C that will be used for breaking ties. The tie-breaking mechanism does have an impact on the election outcome and the properties of the method [1,15]. We here applied a *lexicographic tie-breaking* and used this winner to label the profiles in the training set.

Other possible approaches we could take, which we leave for future work, is to use the whole set of winners as a label. In this case the set of possible labels for a profile would be the power set of C rather than C as is the case now. Allowing a datapoint to be labeled as a member of more than one class is a subject of study of multi-label classification methods [21]. Both of these approaches pose considerable challenges.

[7] The seed for the random number generator during the split is equal to 42.

Using subsets of C as labels makes the feature engineering and data pre-processing task more elaborate, while multi-class classification methods are not as off-the-shelf wide spread as classification methods.

Observe that the Borda winners can be exactly computed from all three representations, the Kemeny winners from Representation 3, while the Dodgson winners cannot be exactly computed from any of the three.

4.2 Representation 1

In this representation the set of features is the set of candidates C and the value for each feature is the Borda score of the feature. The profile from Example 1 is given in Table 4.

f-a	f-b	f-c	f-d	label
11	12	9	10	b

Table 4: The profile from Example 1 in Representation 1.

The shape of the new representation is:

$$(\#profiles, \#candidates).$$

Here, the shape of the Spotify model is $(361, 20)$ and of our generated model is $(12360, 20)$.

The main drawback of this factorization is that it forces anonymity on the profile. Namely the factorized representation loses the information about who voted for whom. The main advantage is that the original profiles don't have to be the same size (number of voters) to transform the dataset to this feature representation. That means that once learned, the prediction model can be used for winner prediction of any new election.

4.3 Representation 2

In this representation there is one feature for each candidate-possible_rank pair. The value of the feature is the number of voters ranking the candidate at the featured position. In other words, we count the number of times each of the candidates is ranked at each position. This is the so called *positional information* of a profile. For example, if the set of candidates has four candidates, we obtain 16 features. The profile from Example 1 is given in Table 5.

f-a1	f-a2	f-a3	f-a4	f-b1	f-b2	f-b3	f-b4	f-c1	f-c2	f-c3	f-c4	f-d1	f-d2	f-d3	f-d4	label
3	0	2	2	1	4	1	1	1	1	4	1	2	2	0	3	b

Table 5: The profile from Example 1 in Representation 2.

The shape of the new representation is:

$$(\#profiles, |C| \cdot |C|).$$

Thus, for given $N = 20$ candidates and length of each vote (ranking) equal to 20 positions, we obtain 400 features. Here, the shape of the Spotify model is $(361, 400)$ and of the generated model: $(12360, 400)$.

The main drawback of this transformation is that we lose the information about the sequence of individual preferences. Although time efficiency was not something we directly were interested in increasing, this representation was observed to lead to a significant reduction in model training time, in particular in when predicting Kemeny winners.

4.4 Representation 3

As a third way to factorize profiles we consider the set of features to be the set of unique pairs of candidates. The value of the feature then is the number of voters that prefer the first candidate to the second in the pair. This is the so called *pairwise majority matrix*. The profile from Example 1 is given in Table 6.

ab	ac	ad	ba	bc	bd	ca	cb	cd	da	db	dc	label
4	4	3	3	5	4	3	2	5	4	2	3	b

Table 6: The profile from Example 1 in Representation 3.

For given $N = 20$ candidates we have 380 possible combinations (without repetition, order matters). The shape of the representation is:

$$(\#profiles, \binom{|C|}{2}).$$

The shape of the Spotify model is: $(361, 380)$ and the generated model: $(12360, 380)$.

5 Experiments and testing

The three different profile representations yielded nine dataset, three for each voting methods. We considered ten different classifiers. We tested the performance of the classifiers on each representation using cross-validation testing. As evaluation metrics to assess the performance of the ten classifiers we used accuracy and F1-score. Accuracy is the proportion of correctly classified instances from the total number of instances. The F1-score is calculated for each class (category) separately and the average is taken as the final F1-score. For each class the F1-Score is $2 * \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$, where precision $= \frac{t_p}{t_p + f_p}$, recall $= \frac{t_p}{t_p + f_n}$, t_p is the number of true positive, namely the number of samples correctly predicted as positive and f_p is the number of false positive, namely number of samples wrongly predicted as positive.

5.1 Borda results

Table 7 summaries the performance of the classifiers using Representation 1. We obtain the best accuracy by using the Gaussian Naive Bayes classifier (100%) and XGBclassifier (99,5%). We noticed that the top-performing classification models are those generated by the group of algorithms capable of generating probability predictions. These also had the highest F1-scores. The 100% accuracy of the Naive Bayes classifier could

be the result of overfitting, however we did not succeed in underfitting without reducing the size of the training dataset.

To understand better the "behavior" of the models we also considered wether the miss-predictions are actually other winners that were in a tie which were not used for labeling due to the lexicographic tie-breaking we used. If the predicted winner was a true winner in a tie we counted that datapoint as true positive. That improved the accuracy results by 0-6%. It is interesting to mention that we noticed that GaussianNaiveBayes tended to push probabilities of the likelihood of a candidate being a winner to 0 or 1. The reason for it is because it assumes that features are conditionally independent given the class, which is the case in this dataset in Representation 1 containing not redundant features.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
GaussianNB	100	100	100	100	0	0	100
XGBClassifier	99.5	66.56	66.37	66.47	4	0.22	99.79
RandomForestClassifier	90.29	33.34	31.88	32.52	2	0.11	90.4
SGDClassifier	85.6	49.58	46.82	47.92	26	1.4	87
SVC(kernel='linear')	76.86	27	23.05	24.79	20	1.08	77.94
RidgeClassifier	70.01	26.25	21.1	21.95	5	0.27	70.28
DecisionTreeClassifier	66.07	49.94	57.14	52.33	0	0	66.07
RandomForestClassifier	4.91	25.04	17.54	19.37	44	2.37	57.28
LinearSVC(C=1.0)	52.32	25.82	16.05	16.64	38	2.05	54.37
AdaBoostClassifier	48.33	33.21	42.86	35.58	0	0	48.33
MLPClassifier	43.37	19.18	13	15.16	111	5.99	49.36
SVC(C=1, kernel='rbf')	22.55	86.3	22.49	15.87	0	0	22.55

Table 7: Representation 1: Borda predictions after hyper-parameter tuning.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
XGBClassifier	100	100	100	100	0	0	100
GaussianNB	100	100	100	100	0	0	100
DecisionTreeClassifier	66.07	49.94	57.14	52.33	0	0	66.07
SVC($kernel =' linear'$)	64.72	33.04	29.97	30.75	0	0	64.72
AdaBoostClassifier	48.33	33.21	42.86	35.58	0	0	48.33
LinearSVC	41.96	19.57	13.33	15.58	2	0.11	42.07
SGDClassifier	40.45	20.3	14.38	16.18	0	0	40.45
RidgeClassifier	30.04	17.12	9.09	11.45	52	2.8	32.84
MLPClassifier	29.29	11.43	9.74	9.87	1	0.05	29.34
SVC($kernel =' rbf'$)	22.55	86.3	22.49	15.87	0	0	22.55
RandomForestClassifier	17.31	9.4	5.2	6.4	151	8.14	25.45

Table 8: Representation 2: Borda predictions after hyper-parameter tuning.

Table 8 and Table 6 respectively summarize the ML performance under Representations 2 and 3. We also observed best performance for the ML methods that performed well with Representation 1.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
XGBClassifier	100	100	100	100	0	0	100
GaussianNB	100	100	100	100	0	0	100
SVC(kernel='linear')	81.55	26.87	24.49	25.57	2	0.11	81.66
LinearSVC	70.12	26.04	21.07	23.2	9	0.49	70.61
DecisionTreeClassifier	66.07	49.94	57.14	52.33	0	0	66.07
SGDClassifier	60.46	29.33	19.66	19.67	20	1.08	61.54
AdaBoostClassifier	48.33	33.21	42.86	35.58	0	0	48.33
MLPClassifier	46.44	18.87	13.92	15.74	72	3.88	50.32
RandomForest(depth = 5)	45.47	19.74	13.66	15.47	76	4.1	49.57
RidgeClassifier	38.78	20.57	11.65	14.16	36	1.94	40.72
RandomForestClassifier	31.45	17.29	9.4	11.17	76	4.1	35.55
SVC(kernel='rbf')	22.55	86.3	22.49	15.87	0	0	22.55

Table 9: Representation 3:Borda predictions after hyper-parameter tuning.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
XGBClassifier	51.81	38.96	37.51	37.4	101	6.32	58.13
GaussianNB	44.43	35.39	36.51	34.88	95	5.94	50.37
RandomForestClassifier	42.55	34.31	35.42	33.47	138	8.64	51.19
SVC(kernel='linear')	40.43	32.57	31.91	31.08	136	8.51	48.94
RidgeClassifier	35.48	36.86	33.36	28.43	205	12.83	48.31
LinearSVC	35.23	40.81	27.36	24.62	97	6.07	41.3
RandomForest(depth = 5)	34.61	31.47	27.64	26.26	130	8.14	42.75
SGDClassifier	25.34	30.36	23.42	20.94	93	5.82	31.16
AdaBoostClassifier	24.97	21.46	18.59	17.7	472	29.54	54.51
MLPClassifier	18.46	19.86	19	15.02	253	15.83	34.29
SVC(kernel='rbf')	8.64	90.14	10.46	10.42	151	9.45	18.09
DecisionTreeClassifier	5.94	16.57	12.84	6.88	109	6.82	12.76

Table 10: Representation 1: Kemeny predictions.

5.2 Kemeny results

Tables 10, 11 and 12 respectively, summarize the performance of the classifiers on predicting Kemeny winners. All the measures are calculated after hyper-parameter tuning. Here the results are discouraging compared with the Borda winner predictions. We obtained the best results for SVC(kernel='rbf') and the RandomForestClassifier when using Representation 3. However, after analyzing the miss-predictions and comparing them with the tied winners, we observed that the SGDClassifier actually has an accuracy of 85%, which is still low, but a considerable improvement. The percentage of the correctly predicted winner involved in the ties varied considerably across classifiers.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
XGBClassifier	21.53	24.42	21.8	18.02	188	11.76	33.29
LinearSVC	19.27	19.89	19.28	16.71	149	9.32	28.59
SVC(kernel='linear')	19.21	21.67	19.19	16.49	201	12.58	31.79
SGDClassifier	17.83	24.42	18.43	16.14	91	5.69	23.52
RidgeClassifier	17.65	19.46	17.6	15.23	151	9.45	27.1
GaussianNB	16.21	22.71	18.24	14.7	189	11.83	28.04
RandomForestClassifier	9.64	15.38	10.75	9.09	278	17.4	27.04
SVC(kernel='rbf')	8.64	90.14	10.46	10.42	151	9.45	18.09
MLPClassifier	7.32	11.25	8.93	6.38	241	15.08	22.4
AdaBoostClassifier	6.7	15.65	7.98	5.46	402	25.16	31.86
RandomForest(depth = 5)	5.51	9.43	8.14	5.15	272	17.02	22.53
DecisionTreeClassifier	3.5	7.04	5.41	2.88	189	11.83	15.33

Table 11: Representation 2: Kemeny predictions.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
SGDClassifier	2.44	0.12	5.0	0.24	1771	83.11	85.55
RandomForestClassifier	60.86	57.18	61.32	55.78	128	6.01	66.87
SVC(kernel='rbf')	60.58	95.26	61.82	71.49	82	3.85	64.43
GradientBoostingClassifier	59.5	55.43	58.66	54.09	79	3.71	63.21
XGBClassifier	26.94	38.13	28.27	25.07	175	8.21	35.15
RandomForest(depth = 5)	7.74	37.15	14.93	8.26	348	16.33	24.07
AdaBoostClassifier	6.29	7.04	7.32	5.04	232	10.89	17.18
DecisionTreeClassifier	5.02	10.71	6.6	3.65	252	11.83	16.85
GaussianNB()	6.34	8.28	7.72	5.37	219	10.28	16.62
RidgeClassifier	4.27	8.11	6.99	3.09	245	11.5	15.77
MLPClassifier	4.41	2.54	4.71	1.7	115	5.4	9.81
LinearSVC	6.62	0.33	5.0	0.62	15	0.7	7.32

Table 12: Representation 3: Kemeny predictions.

5.3 Dodgson results

Surprisingly, we obtained better results predicting Dodgson winners than Kemeny winners, but still with a relatively lower accuracy than Borda winners: 87% with the GradientBoostingClassifier under Representations 1 and 3. Tables 13, 14 and 15 respectively summarize our results. Again, all the measures are calculated after hyper-parameter tuning. Here again, after analyzing the miss-predictions, we observe that the GradientBoostingClassifier actually correctly predicts winners that are among ties.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
GradientBoostingClassifier	87.2	81.37	75.34	77.95	51	2.17	89.37
RandomForestClassifier	84.22	75.05	73.68	73.87	54	2.3	86.52
XGBClassifier	77.63	65.96	48.98	53.13	90	3.83	81.46
GaussianNB	66.31	42.34	41.84	41.63	120	5.1	71.41
RandomForest(depth = 5)	66.57	55.43	36.24	39.26	88	3.74	70.31
SVC(kernel='rbf')	68.06	97.07	61.14	73.34	10	0.43	68.49
RidgeClassifier	63.16	40.37	38.84	38.73	114	4.85	68.01
MLPClassifier	57.64	43.33	44.4	38.79	113	4.81	62.45
AdaBoostClassifier	55.98	35.27	36.61	35.32	117	4.98	60.96
SGDClassifier	50.91	27.14	30.64	23.01	129	5.49	56.4
LinearSVC	34.11	34.02	25.55	19.77	116	4.93	39.04
DecisionTreeClassifier	23.52	17.17	12.22	8.5	25	1.06	24.58

Table 13: Representation 1: Dodgson prediction.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
GradientBoostingClassifier	80.77	68.75	73.07	70.46	59	2.51	83.28
MLPClassifier	77.16	63.44	71.98	66.57	68	2.89	80.05
XGBClassifier	69.72	53.67	57.13	54.12	97	4.13	73.85
SVC(kernel='rbf')	68.06	97.07	61.14	73.34	10	0.43	68.49
LinearSVC	62.91	51.52	55.26	50.62	125	5.32	68.23
RandomForestClassifier	61.93	52.59	62.03	52.66	141	6.0	67.93
RidgeClassifier	63.16	46.38	51.67	47.73	102	4.34	67.5
GaussianNB	57.59	42.61	51.65	44.99	120	5.1	62.69
SGDClassifier	53.21	48.54	40.55	37.82	93	3.96	57.17
AdaBoostClassifier	25.1	19.39	22.26	18.23	195	8.29	33.39
DecisionTreeClassifier	21.23	9.1	9.0	5.52	66	2.81	24.04
RandomForest(depth=5)	22.59	1.13	5.0	1.84	19	0.81	23.4

Table 14: Representation 2: Dodgson predictions.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
GradientBoostingClassifier	87.28	80.85	77.62	78.91	50	2.13	89.41
XGBClassifier	85.92	77.83	78.41	77.9	66	2.81	88.73
RandomForestClassifier	71.08	59.15	69.32	60.71	105	4.47	75.55
GaussianNB	65.76	50.52	45.52	45.62	115	4.89	70.65
SVC(kernel='rbf')	68.06	97.07	61.14	73.34	10	0.43	68.491
RidgeClassifier	62.14	42.01	43.51	41.85	113	4.81	66.95
MLPClassifier	54.4	39.04	42.54	38.8	110	4.68	59.08
SGDClassifier	50.87	36.31	29.72	22.74	128	5.44	56.31
LinearSVC	41.56	46.0	19.17	20.69	51	2.17	43.73
RandomForest(depth = 5)	26.8	62.52	10.2	10.87	23	0.98	27.78
DecisionTreeClassifier	17.31	10.01	18.27	9.93	192	8.17	25.48
AdaBoostClassifier	15.06	11.9	14.83	9.21	172	7.32	22.38

Table 15: Representation 3: Dodgson predictions.

6 Discussion

A learning curve is a graphical representation of model's learning performance[8] over 'experience' or time. We used learning curves here as an ML diagnostic tool for the behavior (underfitting, overfitting or good fit) of ML models and as a diagnosis tool to evaluate if the datasets (training and validation) are representative of the problem

[8] algorithms that learn from a training dataset incrementally

domain. We omit including all the learning curve graphs due to space restrictions and only highlight the most interesting observations.

Borda model behavior. We identified underfitting in the following models: SVM with RBF kernel and Random Forest, where the training loss remains flat regardless of experience. We identify under-fitted models by analyzing the learning curve of the training loss. A learning curve shows under-fitting if : (i) the training loss continues to increase until the end of the training, i.e. premature halt occurs or (ii) the training loss remains flat regardless of training. Some of the ML methods showed over-fitting in their learning curves. A learning curve shows over-fitting if: (i) the training loss continues to increase until the end of training, i.e. premature halt occurs, which we observed with the MLP and SGD classifiers; or (ii) validation loss increases in order to decrease again, which we observed with the Random Forest classifier at its late state. The learning curves' plot showed a good fit for XGBoost, linear SVM and the AdaBoost classifiers, where we observed (i) the training loss decreases to the point of stability and (ii) the validation loss increases to a stability point but there remained a small gap with the training loss. Here, the suitable model fit was presented by the XGBoost, Linear SVM with SGD classifiers. The Representation 2 training set was easier to learn compared to Representation 1, namely more algorithms achieved 100% accuracy for the training set while at the same time increasing accuracy for the test set. Figure 1 presents an example of the train and validation learning curves for models learned in Representation 2.

Kemeny and Dodgson miss-predictions.

We observed that in the case of predicting Kemeny winners, a model using Representation 3 was very well learnable by the SGD classifier. Here, however, the accuracy of the original dataset was very low ca. 2% and the number of miss-prediction was very high – ca. 83 % were wrongly classified (false positive). After checking the miss-classified profiles, we found that 83% of the original miss-predictions were found in ties, meaning a true winner was predicted, but not the winner that was selected by the lexicographic tie-breaking. That analysis revealed that the model accuracy is actually ca. 85%. Figures 2 and 3 (in the Appendix) present an example of the train and validation learning curves for models learned in Representation 3. For predicting Dodgson winners, we obtained the best accuracy with the Gradient Boost classifier learned using Representations 1 and 3. Curiously, here although some winners were correctly classified, but not used as labels due to the tie-breaking rule, their numbers did not make for such a drastic difference as with Kemeny winner predictions. Figures 4, 5 and 6 presents an example of the train and validation learning curves for models using Representations 1, 2 and 3 respectively.

7 Related work

Machine learning, in the context of social choice, has been used to predict the missing components of preference orders (ballots) in [8]. Machine learning has also been used to do political sentiment analysis, namely predict winners of elections given Twitter data (not preference rankings) in [19]. Data science techniques for handling large elections have been used in [6].

Most similar to our work is perhaps Neural networks are used in [4] where the authors train the network to classify profiles with their unanimity winner, their Condorcet winners and unique Borda winner (only profiles that have such a respective winner are used). Burka et al use, what we call representation 3 and profiles in which the number of voters equals 7, 9 or 11, while the number of alternatives equals 3, 4 or 5. Their best Borda accuracy is 92.60%, while their best Condorcet winner accuracy is 98.42%.

Procaccia et al [18] consider the problem of PAC-learnability of scoring methods. We present methods in which the winners of a scoring method, Borda, can be predicted.

Apart from the work we already discussed [7], machine learning, specifically neural networks, have been applied to solve NP-complete problems, specifically the traveling salesman problem [17]. Prates et. al accomplish 80% accuracy with their approach.

8 Conclusions

We asked if winners of computationally hard to compute voting methods can be predicted using machine learning classification. We considered two voting methods for which it can be hard to compute the winners: Kemeny and Dodgson and also one method, Borda, for which winners can be efficiently computed. We considered ten different machine learning classifiers. We constructed considered elections with 20 alternatives for 25 voters. For training machine learning classifiers, we constructed 12360 profiles which we factorized three different ways and for which we computed the Borda, Kemeny and Dodgson winners. Predicting election winners is a robust approach. Once a model is created for an election of a particular size, it can be reused for any election of that size regardless of what the options or who the voters are.

Our answers are, as expected, Borda winners can be predicted with high accuracy. Kemeny and Dodgson winners can be predicted with relative accuracy of 85%-89%. It is important to emphasize that the 12360 profiles we used in training comprise less than 0.01% of all possible profiles for 20 alternatives and 25 agents. The total number of all possible profiles for $|C|$ alternatives and $|V|$ voters is: $\frac{(|V|+|C|!-1)!}{|C|!(|V|-1)!}$. Better accuracies can be obtained with a higher percentage of labeled profiles for training. Further experiments are needed, with different sets of candidates and voters (sets of different sizes that is) to explore the impact of the data set dimensions on the performance of classifiers.

Surprisingly, the Kemeny winners, which in the worst case are computationally easier to compute that the Dodgson winners, are predictable with a lower accuracy than the Dodgson winners. Also surprisingly, our models were able to predict the correct winners for elections when the winners were in a tie (and not used to label a profile because the lexicographic tie-breaking mechanism did not select them).

It would be interesting to see where the miss-predictions occur in the aggregated preference order of Kemeny and Dodgson, are they from among the top ranked or low ranked alternatives or whether there is no correlation between the prediction and a specific position in the Kemeny/Dodgson rank. DEMOCRATIX calculates just the winner(s) not a full preference order, so we were not able to do this analysis at present. In the future we will consider building prediction models for smaller profiles for which the full Kemeny and Dodgson ranks can be computed for analysis.

An interesting avenue to explore is the learnability of collective judgment sets in judgment aggregation. Judgment aggregation studies how the opinions of different individuals on the truth value of a issues can be aggregated into a collective judgment set when the issues that need to be decided upon are possibly logically interconnected [14,9]. Judgment aggregation is known to generalize voting, namely the problem of finding winners for an election can be represented as the problem of finding collective judgment sets [14,9]. Furthermore, Borda, Kemeny and Dodgson are generalizable to judgment aggregation methods [13,10]. The complexity of finding collective judgements using these generalized "voting" methods is typically higher than that of calculating election winners. Furthermore, while the models that predict election winners can be reused for elections of same size of candidate and voter sets, the logic relations between the issues on which opinions are given, would prevent this re-usability in judgment aggregation.

Acknowledgements. The research was partly supported by the project "Better Video workflows via Real-Time Collaboration and AI-Techniques in TV and New Media", funded by the Research Council of Norway under Grant No.: 269790

References

1. Aziz, H., Gaspers, S., Mattei, N., Narodytska, N., Walsh, T.: Ties matter: Complexity of manipulation when tie-breaking with a random vote. In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence. pp. 74–80. AAAI'13, AAAI Press (2013), http://dl.acm.org/citation.cfm?id=2891460.2891471
2. Bartholdi, J., Tovey, C.A., Trick, M.A.: Voting schemes for which it can be difficult to tell who won the election. Social Choice and Welfare **6**, 157–165 (1989), http://dx.doi.org/10.1007/BF00303169
3. Brandt, F.: Some remarks on Dodgson's voting rule. Math. Log. Q. **55**(4), 460–463 (2009). https://doi.org/10.1002/malq.200810017, https://doi.org/10.1002/malq.200810017
4. Burka, D., Puppe, C., Szepesvary, L., Tasnadi, A.: Neural networks would 'vote' according to borda's rule. Tech. rep., Karlsruher Institut für Technologie (KIT) (2016). https://doi.org/10.5445/IR/1000062014
5. Charwat, G., Pfandler, A.: Democratix: A declarative approach to winner determination. In: Proceedings of the 4th International Conference on Algorithmic Decision Theory - Volume 9346. pp. 253–269. ADT 2015, Springer-Verlag, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-319-23114-3_16
6. Csar, T., Lackner, M., Pichler, R., Sallinger, E.: Winner determination in huge elections with mapreduce. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA. pp. 451–458 (2017), http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14894
7. Devlin, D., O'Sullivan, B.: Satisfiability as a classification problem. In: Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science. http://www.cs.ucc.ie/ osullb/pubs/classification.pdf (2008)
8. Doucette, J.A., Larson, K., Cohen, R.: Conventional machine learning for social choice. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. pp. 858–864 (2015), http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9340

9. Endriss, U.: Judgment aggregation. In: Brandt, F., Conitzer, V., Endriss, U., Lang, J., Procaccia, A.D. (eds.) Handbook of Computational Social Choice. Cambridge University Press (2016)
10. Endriss, U.: Judgment aggregation with rationality and feasibility constraints. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018. pp. 946–954 (2018), http://dl.acm.org/citation.cfm?id=3237840
11. Hemaspaandra, E., Hemaspaandra, L.A., Rothe, J.: Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. J. ACM **44**(6), 806–825 (Nov 1997). https://doi.org/10.1145/268999.269002, http://doi.acm.org/10.1145/268999.269002
12. Hemaspaandra, E., Spakowski, H., Vogel, J.: The complexity of Kemeny elections. Theoretical Computer Science **349**(3), 382 – 391 (2005). https://doi.org/https://doi.org/10.1016/j.tcs.2005.08.031, http://www.sciencedirect.com/science/article/pii/S0304397505005785
13. Lang, J., Slavkovik, M.: Judgment aggregation rules and voting rules. In: Algorithmic Decision Theory - Third International Conference, ADT 2013, Bruxelles, Belgium, November 12-14, 2013, Proceedings. pp. 230–243 (2013). https://doi.org/10.1007/978-3-642-41575-3_18, https://doi.org/10.1007/978-3-642-41575-3_18
14. List, C., Polak, B.: Introduction to judgment aggregation. Journal of economic theory **145**(2), 441–466 (2010)
15. Mattei, N., Narodytska, N., Walsh, T.: How hard is it to control an election by breaking ties? In: ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014). pp. 1067–1068 (2014). https://doi.org/10.3233/978-1-61499-419-0-1067, https://doi.org/10.3233/978-1-61499-419-0-1067
16. Nurmi, H.: Voting procedures: A summary analysis. British Journal of Political Science **13**(2), 181–208 (1983), http://www.jstor.org/stable/193949
17. Prates, M.O.R., Avelar, P.H.C., Lemos, H., Lamb, L.C., Vardi, M.Y.: Learning to solve np-complete problems: A graph neural network for decision TSP. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. pp. 4731–4738 (2019). https://doi.org/10.1609/aaai.v33i01.33014731, https://doi.org/10.1609/aaai.v33i01.33014731
18. Procaccia, A.D., Zohar, A., Peleg, Y., Rosenschein, J.S.: The learnability of voting rules. Artif. Intell. **173**(12-13), 1133–1149 (2009). https://doi.org/10.1016/j.artint.2009.03.003, https://doi.org/10.1016/j.artint.2009.03.003
19. Rodríguez, S., Allende-Cid, H., Palma, W., Alfaro, R., Gonzalez, C., Elortegui, C., Santander, P.: Forecasting the chilean electoral year: Using Twitter to predict the presidential elections of 2017. In: Meiselwitz, G. (ed.) Social Computing and Social Media. Technologies and Analytics. pp. 298–314. Springer International Publishing, Cham (2018)
20. Shakirova, E.: Collaborative filtering for music recommender system. In: 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). pp. 548–550 (Feb 2017). https://doi.org/10.1109/EIConRus.2017.7910613
21. Tsoumakas, G., Katakis, I.: Multi-label classification: An overview. International Journal of Data Warehousing and Mining **2007**, 1–13 (2007)
22. Zwicker, W.S.: Introduction to the theory of voting. In: Handbook of Computational Social Choice, pp. 23–56. Cambridge University Press (2016). https://doi.org/10.1017/CBO9781107446984.003, https://doi.org/10.1017/CBO9781107446984.003

Appendix -Figures

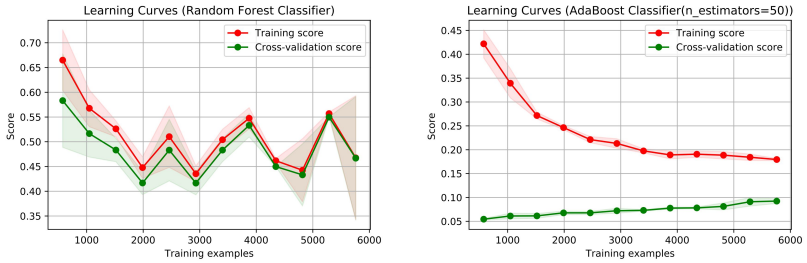

Fig. 1: Train and validation learning curves for Borda models learned in Repres. 2.

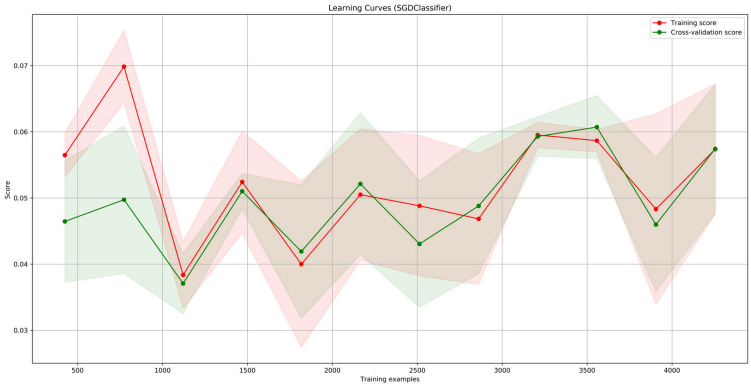

Fig. 2: Train and validation learning curves for Kemeny using Repres. 3., SDG classifier.

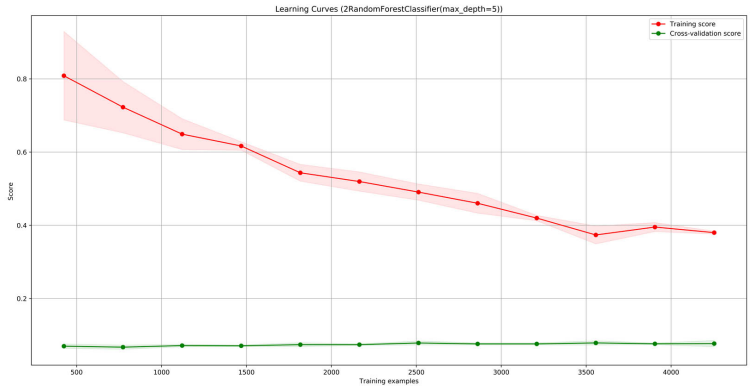

Fig. 3: Train and validation learning curves for Kemeny using Repres. 3., SDG classifier top, 2Random Forest classifier bottom.

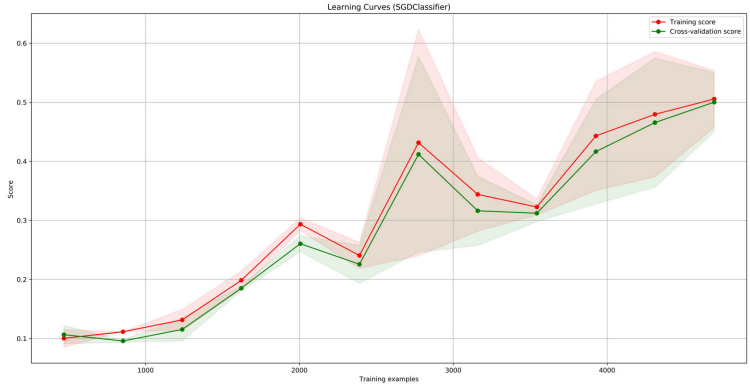

Fig. 4: Learning curve for Dodgson using Representation 1

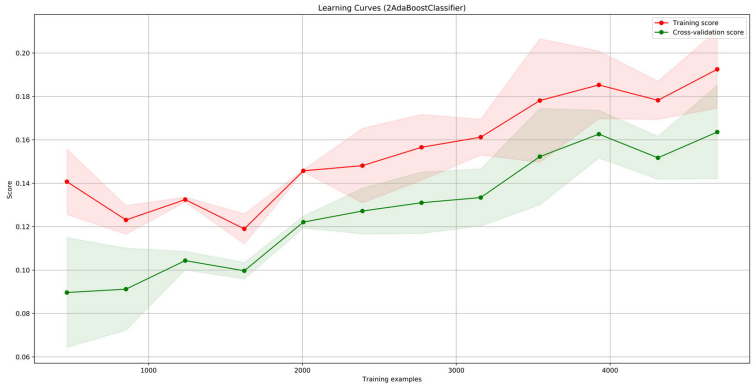

Fig. 5: Learning curve for Dodgson using Representation 2

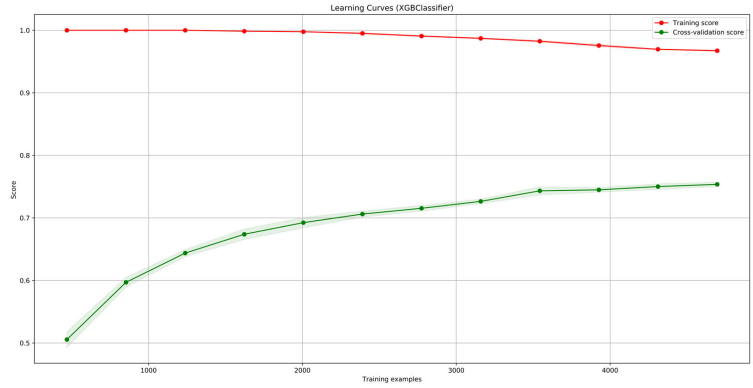

Fig. 6: Learning curve for Dodgson using Repres. 3