Bowdoin College
Bowdoin Digital Commons

Honors Projects | Student Scholarship and Creative Work

2022

Exploiting Context in Linear Influence Games: Improved Algorithms for Model Selection and Performance Evaluation

Daniel Little
Bowdoin College

Follow this and additional works at: https://digitalcommons.bowdoin.edu/honorsprojects

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation
Little, Daniel, "Exploiting Context in Linear Influence Games: Improved Algorithms for Model Selection and Performance Evaluation" (2022). *Honors Projects*. 388.
https://digitalcommons.bowdoin.edu/honorsprojects/388

This Open Access Thesis is brought to you for free and open access by the Student Scholarship and Creative Work at Bowdoin Digital Commons. It has been accepted for inclusion in Honors Projects by an authorized administrator of Bowdoin Digital Commons. For more information, please contact mdoyle@bowdoin.edu, a.sauer@bowdoin.edu.

Exploiting Context in Linear Influence Games: Improved Algorithms for Model Selection and Performance Evaluation

An Honors Paper for the Department of Computer Science

By Daniel Little

Bowdoin College, 2022
© Daniel Little

Contents

Acknowledgments v

Abstract vi

1 Introduction 1

1.1 Literature Review: Model Tuning and Evaluation . . . 1

1.1.1 Holdout: Train/Test Split . . . 1

1.1.2 Holdout: Train/Validate/Test Split . . . 1

1.1.3 Cross-Validation . . . 2

1.1.4 Bootstrap . . . 3

1.1.5 Nested Cross-Validation with Tuning . . . 4

1.1.6 Bootstrap Bias Corrected Cross-Validation . . . 4

1.2 Contributions . . . 6

2 Linear Influence Games 6

2.1 Machine Learning . . . 8

2.2 Roll Call Data . . . 10

2.3 Sponsorship and Cosponsorship Data . . . 10

2.4 Ideal Points and Bill Polarity . . . 10

3 Model Selection and Performance in LIGs 12

3.1 Nested cross-validation with tuning . . . 13

3.2 Bias Bootstrap Corrected CV . . . 15

4 Automated Model Selection 16

5 Estimating Polarity of Held Out Bills 17

6 Future Work 22

A Appendix 26

List of Figures

1 PSNE in a Linear Influence Game . . . 8
2 Distribution of learned ideal points . . . 11
3 10-fold CVT results . . . 18
4 Box-and-Whisker plot of different polarity estimation methods . . . 21
5 Distribution of polarity estimation methods . . . 22
6 10-fold CVT results across a wider range of hyperparameter values . . . 26
7 Mean results for different polarity estimation methods . . . 28
8 Box-and-Whisker plot of polarity estimation distance from learned values . 28
9 Distribution of polarity estimation distance, $\rho = 0.002, \rho' = 0.00013$ . . . 29
10 Mean polarity estimation distance, $\rho = 0.002, \rho' = 0.00013$ . . . 29
11 Box-and-Whisker plot of polarity estimation distance, $\rho = 0.002, \rho' = 0.00013$ 30
12 Distribution of polarity estimation distance, $\rho = 0.002, \rho' = 0.00013$ . . . 30
13 Mean polarity estimation distance, $\rho = 0.002, \rho' = 0.00013$ . . . 31

List of Tables

1 Summary of LIG model metrics 14
2 Results of Nested CVT 15
3 Comparing Results of BBC-CV and Nested-CVT 16
4 Metrics and Desired Values for Automated Model Selection 17
5 Ranked Choices of Automated Model Selection 17
6 Summary of LIG model metrics 21

Acknowledgments

I would like to thank Professor Irfan for his help and guidance in this project. Throughout this project, Professor Irfan has taught me an immense amount about game theory, machine learning, conducting research, scientific writing, and many more disciplines. This project would not be possible without him, and I am incredibly grateful for his thoughtfulness, mentorship, and time. I would also like to thank my friends, family, and other professors for their support in this project, and more broadly, for supporting me throughout my Bowdoin career.

Abstract

In the recent past, extensive experimental works have been performed to predict joint voting outcomes in Congress based on a game-theoretic model of voting behavior known as *Linear Influence Games*. In this thesis, we improve the model selection and evaluation procedure of these past experiments.

First, we implement two methods, Nested Cross-Validation with Tuning (Nested CVT) and Bootstrap Bias Corrected Cross-Validation (BBC-CV), to perform model selection and evaluation with less bias than previous methods. While Nested CVT is a commonly used method, it requires learning a large number of models; BBC-CV is a more recent method boasting less computational cost. Using Nested CVT and BBC-CV we perform not only model selection but also model evaluation, whereas the past work was focused on model selection alone.

Second, previously models were hand picked based on performance measures gathered from CVT, but both Nested CVT and BBC-CV necessitate an automated model selection procedure. We implement such a procedure and compare its selections to what we otherwise would have hand picked.

Additionally, we use sponsorship and cosponsorship data to improve the method for estimating unknown polarity values of bills. Previously, only subject code data was used. This estimation must be done when making voting outcome predictions for a new bill as well as measuring validation or testing errors. We compare and contrast several new methods for estimating unknown bill polarities.

1 Introduction

In any machine learning project, performance estimation is an important part of the pipeline. After learning a model one will want to know how well to expect the model to perform when making predictions on previously unseen data. Raschka [2018] provides reasons for the importance of performance estimation: in addition to being aware of how frequently predictions will be correct in practice, one must also have a performance estimate to tell whether tweaks are improving or worsening a model's predictive performance, or to pick between a group of models or learning algorithms. There are various methods of achieving this goal, each with a different set of trade-offs. We start by reviewing the literature on model tuning and evaluation.

1.1 Literature Review: Model Tuning and Evaluation

1.1.1 Holdout: Train/Test Split

One basic, yet prevalent method of accomplishing this is known as the Train/Test Split, or the Holdout method [Kohavi et al., 1995]: split the dataset into two pieces, for example, putting a random 80% of the examples into a training set and the remaining 20% into a testing set. A model is learned with the training set, and its predictions on the testing set $\hat{y}$ are compared to the testing set's known values y. This comparison computes some measure of loss $l(y, \hat{y})$, such as a squared error or 0-1 loss. To this learned model, the testing set essentially is unseen data; therefore, this testing set performance works well as a proxy for the model's performance on new data. In practice, one can use this performance measure of the learning method or configuration used learned to use the model, rather than the specific model learned on the training set. In this way, one can estimate the error of the learning method, but then learn a final model on all training data to hopefully further increase performance. One potential issue with this method relates to the independence of the chosen training and testing sets: if the examples selected in the training set are not a representative sample, the model's accuracy decreases [Kohavi et al., 1995].

1.1.2 Holdout: Train/Validate/Test Split

Getting a desired model as well as an estimation of its performance on new data becomes more complicated when hyperparameters are involved: hyperparameters are values provided to the model learning process which affect learning in some way. When used to learn models, the ideal hyperparameter values will produce better performing models. But these ideal values are not known ahead of time. One such example is a regularization parameter, which essentially allows one to alter the sensitivity of the learning method to the training data. A model can be learned with each in a range of hyperparameter values, with the best performing value being selected for the final model. Extending the idea of the train/test split, one can split the dataset into three pieces: a training, validation, and testing set, holding, for example, 60%, 20%, and 20% of the examples. One model can be learned for each potential hyperparameter value, all using the training set. Each model is then evaluated with the validation set and given some error measure, or *validation error*. One model is learned with hyperparameter value with the lowest validation error, and this "validated"

model can be tested with the testing set, with the resulting test error measure being used as a performance estimation for the model. One might wonder why the validation error itself can't be used as a performance estimation, avoiding having to hold out more data to calculate a test error; both errors seem conceptually similar. The reason is that hyperparameter values were decided based on this validation set and its error, but the test set is meant to be seen essentially only once for the purpose of getting an accurate error measure before training a model with the chosen configuration on all of the data. Using the validation set as a test set, and likewise, iterating over the test set to try improve learning can lead to an optimistically biased final error measure [Raschka, 2018].

1.1.3 Cross-Validation

Another such method is K-fold Cross-Validation (CV). This involves splitting up the dataset D into a some number k of folds and picking one fold k_i to be a testing set, and the remaining folds $k-i$ as a training set. This step is repeated several times until all folds have been used as a testing set. On each iteration, a model is learned with the training data and prediction error is calculated on the testing set. Each of these intermediate errors can be averaged together as a prediction error estimate, as shown in Pseudocode 1. The number of folds to use in CV can greatly affect the results: if the minimum number of folds, two, is used, the training set will be half of the dataset. In experiments with small datasets, this might not be a large enough training set to effectively learn a model. Using a very large number of folds will improve the learned models, but the testing set becomes much smaller. In this case, the variability of the testing set will increase, although there are more total error measures to combine. The extreme version of this procedure, Leave-One-Out CV, is used regularly. Additionally, using a large number of folds can be increasingly computationally expensive, as an additional model needs to be learned for each fold. Kohavi et al. [1995] show experimentally that ten is the optimal number of folds to use, even if the dataset and computational resources would allow for using more folds. Kim [2009] also discusses repeating this procedure multiple times (repeated CV), and report that it results in lower variability in error. We use the notation $M(D)$ to mean the predictions made by the model M on the labels of data D.

Algorithm 1 K-fold Cross-Validation

```
procedure CV(Dataset D, learning method f)
    k ← number of folds
    Split D into k folds
    for each fold i in k do
        testSet ← fold i of D                              ▷ held out fold
        trainingSet ← all folds k − i of D                 ▷ all remaining folds
        M ← f(trainingSet)
        testErr_i ← l(testSet, M(testSet))                 ▷ calculate the loss for the testing set
    end for
    return (1/k) Σ_{n=1}^{k} trErr_i                       ▷ mean training error across all folds
end procedure
```

After getting a training error from cross-validation, one could then train a final model on all data with a more robust out-of-sample error estimate.

In addition, a modification to CV can turn it into a model selection procedure called Cross-Validation with Tuning (CVT), shown in Pseudocode 2: CV is performed on each of a number of possible model configurations (e.g. hyperparameter values). The error on each round of CV is then used not as testing error, but as validation error, and the optimal hyperparameters are chosen based on the validation error calculated in CV. However, the CV-returned error for the chosen hyperparameter value should again not be used as a prediction error estimate in this case. In this example, the learning method (and thus the CV procedure) also takes a configuration C_i.

Algorithm 2 Cross-Validation with Tuning

1: **procedure** CVT(D f, C)
2: $k \leftarrow$ *number of folds*
3: **for** C_i `in` C **do**
4: $validationError_i = CV(D, f, C_i)$
5: **end for**
6: $i* = \underset{i}{\operatorname{argmin}}\, validationError_i$
7: **return** $\langle C_{i*}, validationError_{i*} \rangle$
8: **end procedure**

After running the CVT procedure, one final model can be trained with the whole dataset and the configuration C_i returned by CVT. But as with the Train/Test/Validate split, the returned validation error would be a biased estimation of model performance [Raschka, 2018].

CV is particularly helpful for small datasets: if one's dataset is large, then a randomly sampled training set calculated by the Holdout method is likely to be representative of the whole dataset. This becomes less likely with a smaller number of data points, and so a training or validation error calculated via Holdout may have a higher bias. CV essentially repeats the Holdout method such that at the end of the procedure, the average error calculated across all folds includes errors from models both trained with, and tested against, every data point available.

1.1.4 Bootstrap

The Bootstrap method is based on the idea that sampling with replacement from one's dataset can effectively create new samples, allowing one to make statistical inferences and better estimate the true distribution and its parameters [Efron and Tibshirani, 1993]. A general version of this procedure, not limited to model evaluation or tuning, can be summarized as such: we create a new sample of data called bootstrap sample by repeatedly sampling from our dataset with replacement until the bootstrap sample is the same size as full dataset. Then we calculate some statistic on this bootstrap sample, such as the sample mean. We repeat this procedure a number of times (often hundreds), and use the average mean across bootstrap samples as an estimate of the population parameter.

In the context of model evaluation Efron and Tibshirani [1993] describe several possible procedures using the bootstrap, most of which involve training a model on each bootstrap sample, calculating the error, and averaging these training or resubstitution errors across all bootstrap samples generated. This procedure alone is likely to be optimistically biased [Efron and Tibshirani, 1993], as it is more or less using training error as an estimation of model performance. There are some methods to estimate what this bias is likely to be and correct for it in the error calculation. Raschka [2018] also describes a commonly used modification of the bootstrap called the Leave-One-Out Bootstrap (LOOB) which involves collecting the data that were not sampled for the bootstrap sample and using them as a testing or validation set for a model trained on the bootstrap sample. This is shown in Pseudocode 3.

Algorithm 3 Leave-One-Out Bootstrap

1: **procedure** CVT(D, b)
2:     **for** i = 1:b **do**
3:         $bootstrap_i \leftarrow$ `sample` $|D|$ `times from` D `with replacement`
4:         $bootstrap_{-i} \leftarrow$ `samples in` D `not in` $bootstrap_i$
5:         $M \leftarrow f(bootstrap_i)$
6:         $trainErr_i \leftarrow l(bootstrap_{-i}, M(testSet))$
7:     **end for**
8:     **return** $\frac{1}{B}\sum_{n=1}^{B} trainErr_n$ ▷ mean training error across all bootstrap samples
9: **end procedure**

1.1.5 Nested Cross-Validation with Tuning

Methods like CVT and LOOB provide model selection but do not return an unbiased measure of out-of-sample testing error. Ostertag-Hill [2020] suggest Nested Cross-Validation with Tuning (Nested CVT) as one path forward of improving performance estimation. Nested CVT involves nesting the CVT procedure within CV: the outer loop withholds one fold of data from the inner loop to use as a testing set after the inner loop completes, and the inner loop performs CVT and returns a validated model configuration. The outer loop will then learn a model with the (outer) training set, and test it against a testing set that was completely held out from all models learned in the inner loop. This algorithm is demonstrated in Pseudocode 4. Note that each inner loop may return a different selected model, meaning the procedure can return anywhere from 1 to k different validated model configurations, each with its own testing error calculated in the outer loop. Varma and Simon [2006] demonstrate experimentally that doing a nested cross-validation reduces the bias in the reported error rate on an independent test set.

1.1.6 Bootstrap Bias Corrected Cross-Validation

Tsamardinos et al. [2018] present a novel method for model selection and performance estimation. Bias Bootstrap Corrected Cross-Validation (BBC-CV) first performs CVT, but saves the *out-of-sample* predictions made by all model configurations when calculating validation error in a matrix Π for use later on to correct for the bias of the validation er-

Algorithm 4 Nested Cross-Validation with Tuning

```
1: procedure NESTED-CVT(X, y f, C)
2:     k ← number of folds
3:     Split X and y into k folds
4:     for each fold i in k do
5:         testSet ← fold i of X                          ▷ held out fold
6:         trainingSet ← all folds k − i of X             ▷ all remaining folds
7:         C*_i ← CVT(trainingSet, f, C)
8:         testErr_i ← l(testSet, M(testSet))
9:     end for
10:    return ⟨allC*, corresp.testError values⟩
11: end procedure
```

ror calculated in CVT. These predictions are bootstrap-sampled from Π, creating samples across every model configuration. Some model selection procedure is employed (possibly the same one as CVT, simply picking the configuration with the smallest loss across the bootstrap sample). Depending on the data points in the bootstrap sample, the procedure may select the same configuration returned by CVT, or it may select a different configuration. Regardless, the model's loss on the *predictions omitted from the bootstrap sample* is calculated, and averaged across each loop of this procedure. The resulting average loss L_{BBC} is returned as an estimation of test error.

Algorithm 5 Bootstrap Bias Corrected Cross-Validation

```
1: procedure BBC-CV(D, b, f, C)
2:     ⟨validationError, C*, Π⟩ ← CVT(D, f, C)
3:     for i = 1:b do
4:         Π^b ← sample with replacement N rows of Π
5:         Π/b ←  rows of Π not in Π
6:         j ← select(Π^b, labels for corresponding data)
7:         L_i ← l(Π^{/b}(j), labels corresponding data)
8:     end for
9:     return (1/b) Σ_i^b L_i
10: end procedure
```

BCC-CV requires learning orders of magnitude fewer models than nested CV, and, in fact, requires only learning the models for CVT. Tsamardinos et al. [2018] also verifies experimentally that the bias of the loss returned by BBC-CV is a significant improvement on just using the CVT validation error, and is comparable to the bias of average testing error returned by Nested CVT.

BBC-CV explores an interesting concept about combining model selection and evaluation: the error estimation isn't estimating the error of the particular selected model configuration; it's estimating error for whatever heuristics are being used to do the selection. This can be seen in the fact that the expected error of one configuration is being estimated with the

error of potentially different configurations.

1.2 Contributions

We apply model selection and error estimation techniques to the Linear Influence Game used to model and predict senate votes, introduced by Irfan and Ortiz [2011] and developed further by Irfan and Gordon [2018], Phillips et al. [2021], and Ostertag-Hill [2020]. Previously the model had only biased estimations of error, as well as other metrics such as the distance of senators' learned ideal points from a known reference [Poole et al., 2015]. We utilize techniques to reduce this bias, and in particular, adopt one technique to the requirements of this project.

In the course of implementing these techniques, we also develop an algorithm to automatically select model configurations based on their measured performance in a procedure like CVT; previously models were always picked by hand after running CVT.

We also investigate several new methods for estimating bill polarity, which is required for both predicting voting outcomes of bills and measuring model performance on a held out set. These new methods use sponsorship and cosponsorship data, whereas previous methods only used subject codes of bills.

2 Linear Influence Games

This project builds on a base of prior experiments which aim to predict senate voting outcomes. Irfan and Ortiz [2011] introduce the Linear Influence Game (LIG), a type of non-cooperative, graphical game inspired by diffusion and threshold models. In a LIG, individuals are represented as nodes on a directed graph, and a node i's payoff, u_i, is dictated by weights of incoming edges as well as a node's threshold parameter. Any node j might have edges going onto i with weight $w_{ij} \in \mathbb{R}$, and similarly, i may have outgoing edges with weight $w_{ji} \in \mathbb{R}$ to any other node j. These weights represent an influence factor from one player in a game to another. Note that these weights can be negative or positive, as well as asymmetric: two nodes can influence one another differently. If i has an edge going to j with a positive weight large in magnitude, then player j will be more inclined to match i's action taken. The opposite is true for a negative weight. Nodes additionally have a threshold value $b_i \in \mathbb{R}$ which generally represents how resistant a node is to influence from other nodes.

Each node i has a choice of binary actions $x_i \in \{-1, 1\}$. In non-cooperative game theory, a player's best response is the action that maximizes their payoff. In the case of an influence game $\mathcal{G}$, a player's best responses can be determined with the *best response correspondence* [Irfan and Ortiz, 2011] $\mathcal{BR}_i^{\mathcal{G}} : \{-1, 1\}^{n-1} \to 2^{\{-1,1\}}$, which is defined as:

$$\mathcal{BR}_i^{\mathcal{G}}(\mathbf{x}_{-i}) \equiv \underset{x_i \in \{-1,1\}}{\arg\max}\; u_i(x_i, \mathbf{x}_{-i}). \tag{1}$$

To understand how a best response would be determined in an real example of a game, we need to define a player i's payoff function u_i. The payoff of a player's action is determined by the influence function [Irfan and Ortiz, 2011], which determines combines a node i's influences and threshold value. The influence function for node i, parameterized by an action x_i and all other nodes actions $\mathbf{x}_{-i}$ is

$$f_i(\mathbf{x}_{-i}) \equiv \sum_{j \neq i} w_{ij} x_j - b_i. \tag{2}$$

The influence value $f_i(\mathbf{x}_{-i})$ is calculated as the sum of all products of incoming influence factors and the influencing node's action $x_j \in \{-1, 1\}$ and finally subtracting threshold value b_i. A player's best response will be the affirmative action, 1, if the influence value is positive. If the value is negative, the node's best response will be the action -1, and if the value is 0, the node is indifferent between the two actions; either one is a best response. This results in the best response behavior [Irfan and Ortiz, 2011]:

$$\begin{aligned} f_i(\mathbf{x}) > 0 &\Longrightarrow x_i^* = 1, \\ f_i(\mathbf{x}) < 0 &\Longrightarrow x_i^* = -1, \text{ and} \\ f_i(\mathbf{x}) = 0 &\Longrightarrow x_i^* \in \{-1, 1\}. \end{aligned}$$

Alternatively, this can be shortened into a compact syntax of a single payoff function

$$u_i(x_i, \mathbf{x}_{-i}) = x_i f_i(\mathbf{x}_{-}i). \tag{3}$$

by multiplying the action x_i by the value returned by the influence function, such that the payoff is positive if the player chooses the action with the same sign as the value of the influence function. This payoff function is equivalent to the best response behavior.

The general picture of the game comes together: each node will attempt to maximize their payoff by voting in accordance with their own threshold value and neighboring players' actions, as well as the influences of these neighbors. But Irfan and Ortiz [2011] utilize the LIG not to predict one player's action, but to predict a *joint* outcome using a learned game. This challenge of finding out the likely outcomes of the entire game is significantly more difficult, but it starts with the pure-strategy Nash equilibrium, or PSNE. A PSNE can be defined succinctly as a simultaneous best response; A given joint action is a PSNE if after everyone played the joint action, no player's payoff would increase after the fact if they switched their vote. This concept is one of many for calculating stable outcomes of a game. Compared to a mixed-strategy Nash equilibrium, in which stable joint actions can consist of a player playing a number of different actions with some probability, a players in a PSNE choose play exactly action all of the time. Figure 1 demonstrates several PSNE within a small LIG.

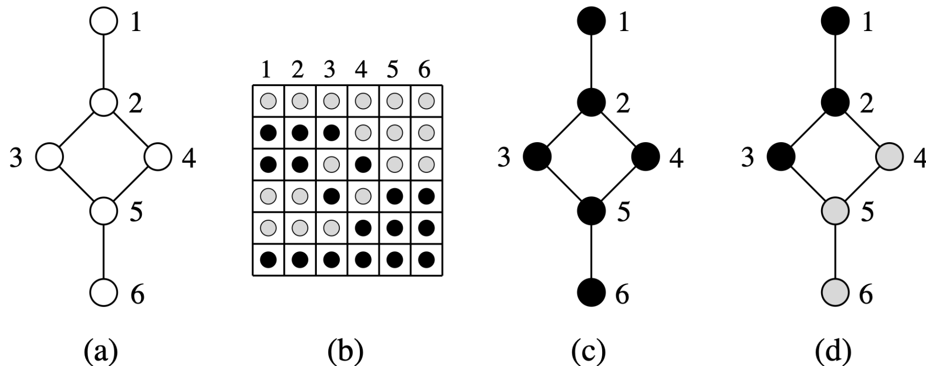


Figure 1: This example of a PSNE in a LIG, reproduced with permission from Irfan and Ortiz [2011], is a particular variety of LIG called a majority game in which all weights are 1 and thresholds are 0, meaning that any node's best response is simply to match the action of the majority of its neighbors. In this example, (a) is a visual representation of the game structure, and (b) lists all PSNE: each row has each node 1-6 choosing one of two actions $\{-1, 1\}$, represented by grey and black. Note that if a node's neighbors are split by vote, the node can choose either option; both are a PSNE. (c) and (d) are graphical representations of the last and second rows of (b), respectively.

2.1 Machine Learning

Irfan and Ortiz [2011], Irfan and Gordon [2018], Phillips et al. [2021] and this paper use the learning algorithm developed by Honorio and Ortiz [2015]. The algorithm is a generative mixture model which has a LIG $\mathcal{G}$ pick a joint action $\mathbf{x}$ from the set of a game $\mathcal{G}$'s set of PSNE, or $\mathcal{NE}(\mathcal{G})$, with probability q, and pick a joint action $\mathbf{x} \notin \mathcal{NE}(\mathcal{G})$ with probability $1 - q$. A model is learned with a dataset made from training examples of joint actions with the implication that these joint actions are "real-world" PSNE for whatever the game which best models the real world would be. The algorithm balances two goals in learning a game: the first is generating a game with a minimal number of PSNE, and the second is generating a game that best represents the real-world data. The first is important because if a model generates a massive number of PSNE, it has no real predictive ability.[1] To quantify the first goal, the term $\pi(\mathcal{G})$, or the *true proportion of equilibria*, is defined as

$$\pi(\mathcal{G}) \equiv |\mathcal{NE}(\mathcal{G})|/2^n. \tag{4}$$

meaning the proportion of all possible joint actions which are PSNE for the particular game $\mathcal{G}$. Similarly, the second goal is quantified with $\hat{\pi}(\mathcal{G})$, or the *empirical proportion of equilibria*

[1]For example, in a trivial game [Honorio and Ortiz, 2015] with the matrix of weights $\mathbf{W} = 0$ and biases $\mathbf{b} = 0$, every joint action of the possible 2^n would be a PSNE, which is clearly unhelpful in the realm of prediction.

$$\hat{\pi}(\mathcal{G}) \equiv \frac{1}{m}\sum_{l} 1[\mathbf{x}^{(l)} \in \mathcal{NE}(\mathcal{G})], \tag{5}$$

meaning the proportion of the real-world "training" data that are represented in $\mathcal{NE}(\mathcal{G})$. Honorio and Ortiz [2015] then derive the maximum likelihood estimation formulation of the problem as such. The formulation defines the optimal game $\hat{\mathcal{G}}$ and mixture parameter $\hat{q}$ as the pair which minimizes the log likelihood $\hat{\mathcal{L}}(\mathcal{G}, q)$:

$$(\hat{\mathcal{G}}, \hat{q}) \in \arg\max_{(\mathcal{G},q)} \hat{\mathcal{L}}(\mathcal{G}, q), \tag{6}$$

where $KL(p1\|p2)$ is the *Kullback-Leibler divergence* of two Bernoulli distributions, which finds the statistical difference between two distributions, and:

$$\hat{\mathcal{L}}(\mathcal{G}, q) = KL(\hat{\pi}(\mathcal{G})\|\pi(\mathcal{G})) - KL(\hat{\pi}(\mathcal{G})\|q) - n\log 2 \tag{7}$$

Intuitively, this formulation is stating that we want to minimize the first term by maximizing $\pi(\mathcal{G})$, the proportion of real-world data which are PSNE in the game $\mathcal{G}$, in relation $\hat{\pi}(\mathcal{G})$, the proportion of all joint actions generated by $\mathcal{G}$ which are PSNE, such that $KL(\hat{\pi}(\mathcal{G})\|\pi(\mathcal{G}))$ is minimized. We then want to minimize the second term by making the KL divergence between $(\hat{\pi}(\mathcal{G}))$ and q as small as possible, meaning that the optimal mixture parameter $\hat{q} = \hat{\pi}(\mathcal{G})$. However, calculating $\pi(\mathcal{G})$ requires computing the entire set of $\mathcal{NE}(\mathcal{G})$, which is shown to be $\#P - complete$ [Irfan and Ortiz, 2011]. Honorio and Ortiz [2015] show that with a high probability, finding a game $\mathcal{G}$ which maximizes $\hat{\pi}(\mathcal{G})$ is equivalent to minimizing the loss $\ell(z) = 1[z < 0]$ of a game across all real-world bills:

$$\sum_{l} \max_{i} \ell\big[x_i^{(l)}(\mathbf{w}_{i,-i}^T \mathbf{x}_{-i}^{(l)} - b_i)\big] \tag{8}$$

Equation 8 essentially counts the total instances across all labels of any player not playing a best response for a given game $\mathcal{G}$. This count can be divided by the total size of the dataset M to get the proportion of labels which are *not* captured as PSNE by $\mathcal{G}$, equivalent to $1 - \hat{\pi}(\mathcal{G})$, and the values of $\mathbf{W}, \mathbf{b}$ which minimize this summation of loss constitute the optimal game. An ℓ_1-norm regularizer is added as a hyperparameter, which when increased, biases the learning process toward models with smaller weights and therefore fewer edges.

$$\underset{\mathbf{W},\mathbf{b}}{min} \frac{1}{M} \sum_{l} \max_{i} \ell\big[x_i^{(l)}(\mathbf{w}_{i,-i}^T \mathbf{x}_{-i}^{(l)} - b_i)\big] + \rho\|\mathbf{W}\|1. \tag{9}$$

The implementation of this minimization is converted to use logistic regression with a smooth upper bound, $\ell(z) = \log(1 + e^{-z})$:

$$\min_{\mathbf{W},\mathbf{b}} \frac{1}{m} \sum_{l} \log(1 + \sum_{i} e^{-x_i^{(l)}(\mathbf{w}_{i,-i}^T \mathbf{x}_{-i}^{(l)} - b_i)}) + \rho\|\mathbf{w}\|_1. \quad (10)$$

2.2 Roll Call Data

The models used by Irfan and Ortiz [2011], Irfan and Gordon [2018], Phillips et al. [2021], Ostertag-Hill [2020] all utilize the LIG and the machine learning algorithm to instantiate games such that each node represents one senator. Senate voting outcomes are used as labels, and the machine learning learning process produces a game that will best be able to model the relationships between senators to produce PSNE, i.e. joint voting outcomes, which match the voting outcomes from the real world.

A number of different senates have been used for the LIG and training examples; the most recent project, Ostertag-Hill [2020], used the 114th and 115th senates combined. We incorporate the 116th senate into the dataset as well, increasing from 722 to 917 total bills to learn models with. We also increase the number of senators in the model from 103 to 110. We use techniques developed by Phillips et al. [2021] to combine different senators into one game: When an incoming senator replaces one of the same party, the two are "merged." Alternatively, if a replacement is from the opposing party, both senators stay in the dataset with their votes being extrapolated to be the party mean. Senators Blackburn, Braun, Cramer, Hawley, Kelly, McSally, and Rosen are added as new senators to the model. Senator Simena was merged with senator Flake, and senator Romney was merged with senator Hatch. This senate and bill data are collected by "Congress" repository from the @unitedstates github account.

2.3 Sponsorship and Cosponsorship Data

In this project, we use sponsorship and cosponsorhip data to better estimate the polarities of unknown bills. In that process, we make our dataset more complete by filling in previously missing sponsorship and cosponsorship data for amendments. Thus in addition to the @unitedstates github data, we add a new data source (`senate.gov`) to add in the sponsors of every bill, including amendments. The amendment sponsors were scraped using Selenium and saved in the dataset. This sponsor data is important to the project as new methods for estimating unknown bill polarity rely heavily on bill sponsorship and cosponsorship, and future polarity estimation improvements can be made using this data.

2.4 Ideal Points and Bill Polarity

LIGs have a large capacity to capture group dynamics in the senate, but they model neither individuals' political beliefs nor the political leaning of individual bills. This means that a LIG model doesn't differentiate any bill from another: $\mathcal{NE}(\mathcal{G})$ will always be identical for two bills, even if those bills have wildly different voting outcomes in the senate.

Inspired by Gerrish [2013]'s experiments with ideal point models and the US senate, Irfan and Gordon [2018] add to the LIG model the concept of ideal points. In addition to incoming weights and a threshold value b_i, nodes were also given an ideal point value p_i, and each bill is assigned a polarity value a_l. Both variables represent a left-right political leaning; by convention, negative values represent a left-lean and positive values a right-lean. Additionally, ideal points and polarities are learned as the model is learned, similar to weights and thresholds.

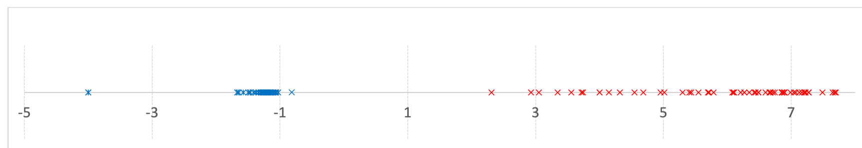


Figure 2: Distribution of learned ideal points for a model with the BBC-CV method's chosen best configuration of $\rho = 0.0024$ and $\rho' = 0.00038$.

This alters Equation 2 by adding the term $(p_i \cdot a_l)$ in the summation:

$$f_i(\mathbf{x}_{-i}) \equiv \sum_{j \neq i} w_{ij} x_j + (p_i \cdot a_l) - b_i. \tag{11}$$

This term functions as such: when the bill's polarity leans the same political direction as the senator, a positive term is added whether both are positive or negative. The added positive term means the senator is more likely to vote *yea*, as it adds to their total influence value. If the bill's polarity and the senator's ideal point disagree, the resulting term is negative, and the senator is more likely to vote *nay*. The smaller both values are, the smaller the resulting term.

The machine learning algorithm is also modified to learn an ideal point for each senator, as well as a polarity for each bill. The minimization is changed, and an additional $\ell 1$-norm regularizer ρ' is added to control the magnitude of learned ideal points and polarities.

$$\min_{\mathbf{W},\mathbf{b},\mathbf{p},\mathbf{a}} \frac{1}{m} \sum_l \log(1 + \sum_i e^{-x_i^{(l)}(\mathbf{w}_{i,-i}^T \mathbf{x}_{-i}^{(l)} + (p_i \cdot a_l) - b_i)}) + \rho \|\mathbf{w}\|_1 + \rho' \|\mathbf{p} \cdot \mathbf{a}\|. \tag{12}$$

The magnitudes of learned senator ideal points and bill polarities are dimensionless and theoretically unbounded, so choosing a good value of ρ' helps to keep the values within a reasonable range. Figure 2 shows the distribution for a model configuration chosen in model selection of $\rho = 0.0024$, $\rho = 0.00038'$.

The aim in adding ideal points and polarities is to allow the model to capture individual preferences and political leanings in addition to group dynamics and influence. Reference ideal point values learned by Poole et al. [2015] are also used in the model selection process

to bias the final model to have learned ideal points which match a known reference point. Critically, a bill's polarity is needed to predict its joint voting outcomes, but bill polarities are only learned as the model is learned. If some bills are held out for testing or validation purposes, their polarity is never learned. Irfan and Gordon [2018] utilize a technique developed by Poole et al. [2015] to estimate the polarity of a bill given a set of learned bills by comparing the subject codes of the unknown bill to the known bills. We discuss this technique and attempt to improve upon it in Section 5.

Phillips et al. [2021] takes the LIG model from Irfan and Ortiz [2014] and adds a bill-clustering step; bills are clustered using subject codes and fuzzy k-means clustering, and then a separate LIG model is learned for each cluster, or sphere. New bills can be clustered using their subject codes and predictions can be made with a model for the appropriate cluster. These more specialized models are hopefully tuned to better predict outcomes within each sphere. Ostertag-Hill [2020] combines the two additions from Irfan and Gordon [2018] and Phillips et al. [2021], with clustering bills into different spheres and learning models with additional parameters to capture individual political beliefs. We don't incorporate clustering into this research, but the new model evaluation procedures will make it easier to compare different versions of LIGs; one example experiment could be comparing the performance measures of models learned with and without clustering.

3 Model Selection and Performance in LIGs

Previous experiments using LIGs have used CVT to select the optimal hyperparameters. One straightforward way to do this would be to go through every model configuration, meaning every combination of hyperparameters ρ and ρ', and perform K-fold CV for each. Within CV, one LIG would be learned on the training folds, and for each held out bill in the validation set, the LIG would be used to predict the joint voting outcomes. This prediction could be compared to the known voting outcome of the bills in the validation set, and an average error could be calculated for each model configuration. The problem with this is that making a prediction is NP-hard [Irfan and Ortiz, 2011], and practically, can take up to hours or days depending on the bill being predicted. Rather than using predictions, cross-validation has been tweaked for these experiments to output a several descriptive metrics about the model which do not require prediction to calculate. The optimal model configuration is then chosen by hand based on these metrics. For example, CVT returns for each model configuration the number of edges, the euclidean distance between learned ideal points and the known reference, the proportion of bills in the validation set which are PSNE, the proportion of senators not playing best responses across the validation set (validation best-response error), and several more metrics. A researcher then sorts through this data to find a reasonable choice of values for ρ and ρ'. Each of these metrics are important in some way; if model selection were performed only on the basis of, for example, validation best-response error, the chosen values of ρ and ρ' might learn a very dense model with too many edges and making any predictions would take an unreasonably long time. Therefore, in addition to validation best-response error, number of edges and several other metrics must be taken into account for the selection of hyperparameter values.

Algorithm 6 Cross-Validation with Tuning for LIG

```
1: procedure CVT-LIG(D, f, set of all combinations of ρ, ρ′R)
2:     for i in R do
3:         Split D into k folds
4:         validationError_i ← CV(D, f, R_i)        ▷ proportion of senators not voting BR
5:         // other metrics, e.g. number of edges in the graph, are saved
6:     end for
7:     Store mean validation error and other metrics and associated ρ, ρ′
8:     Manually select best ρ and ρ′ configuration
9: end procedure
```

Adapted to the LIG, Pseudocode 6 shows the CVT procedure performed for model selection. The metrics calculated for each held-out fold are used to select a model configuration (i.e. values for ρ and ρ'). Therefore, the selected model configuration is likely to perform worse on new data than the metrics suggest. These metrics are optimistically biased the ideal model configuration was chosen based on them.

The metrics collected are summarized in table 1. We also tweak these metrics in a couple of ways in this paper. First, the calculation of ideal point distance was changed in a significant way: the reference ideal points are all scaled between -1 and 1, whereas our learned ideal points are unbounded. The ideal point distance is calculated as the euclidean distance between the reference and learned ideal points. A consequence of this is that in this metric, ideal points are penalized for being large in magnitude, even if they are proportional to the reference values. This made the value cumbersome to use when hand selecting models, and infeasible to use in automated model selection: if you try to minimize the ideal point distance, the selected configuration would end up picking a value for ρ' that learns ideal points to be zero, because this distance is empirically smaller than any model with nonzero learned ideal point values. We change the calculation to calculate euclidean distance of the reference values and normalized learned ideal points such that they sit between -1 and 1. This turns the ideal point distance into a much more practically usable metric.

In addition, we measure two new metrics: the proportion of learned ideal points and polarities which equal zero. In automated model selection, we wanted to explicitly bias the selection procedure against configurations which tend not to learn meaningful values for ideal point and polarity. Rather than trying to use the ideal point distance as a proxy, we decided to explicitly measure this to use in the automated procedure.

3.1 Nested cross-validation with tuning

We implement Nested CVT on the LIG model to perform model selection. Because Nested CVT requires the inner CVT procedure to automatically select a model, we also develop an algorithm for selecting an LIG model based on the calculated metrics, discussed further in Section 4. The Nested CVT procedure returns any number from 1 to k different validated model configurations, where k is the number of folds chosen for the outer loop, as well as an unbiased estimation of metrics such as validation error, number of edges, etc. for each

Metric	Definition
Training error	Percent of senators not voting $\mathcal{BR}$ across a training set
Validation error	Percent of senators not voting $\mathcal{BR}$ across a validation set
Testing error	Percent of senators not voting $\mathcal{BR}$ across testing bills
Training PSNE	Percent of training set bills represented as PSNE for LIG
Validation PSNE	Percent of validation set bills represented as PSNE for LIG
Testing PSNE	Percent of testing set bills represented as PSNE for LIG
Number of edges	Number of edges in a learned LIG
Ideal Point distance	Euclidean distance of learned ideal points to a references
Proportion IP = 0	Proportion of all learned ideal points equal to 0
Proportion Polarity = 0	Proportion of all learned polarities equal to 0

Table 1: A summary of metrics captured in the Cross-Validation process. These metrics are used for model selection and evaluation. For example, the validation error would be used to pick among ρ and ρ' values to find a model configuration that is likely to model the data used for learning.

configuration. See Table 1 for the list of all metrics collected.

For the senate-based LIG model, learning a model is nontrivial; it may take 30-40 seconds to learn a single model with a large proportion of the 917 bills in the training set. Performing CVT alone can take considerable time: 40 seconds $\cdot$ 10 folds $\cdot$ 100 model configurations (10 ρ values $\cdot$ 10 ρ' values) can easily take up to 12 hours or more. Nesting this procedure in an additional loop of ten folds could take over 120 hours. When performing Nested CVT, the number of configurations tested needs to be limited (to in the 30-50 range in our runs) as well as the number of outer loops to keep the execution times manageable.

Experimental Results

The Nested CVT procedure was run with $k = 5$ outer folds and returned five different model configurations.

These results are promising for both Nested CVT and the automated model selection procedure. Several of the selected values of hyperparameters ρ and ρ' are very similar to what was selected in Irfan and Gordon [2018] ($\rho = 0.00225, \rho' = 0.0004$). Additionally, the values of testing error returned by Nested CVT are close to the validation error returned by CVT in Irfan and Gordon [2018] (17.0). More experimentation is needed to verify the difference in bias between the two, as several other variables were changed (e.g. the senators making up the LIG).

ρ	ρ'	trErr	testErr	trPSNE	testPSNE	# Edges	IP Dist.	%IP = 0	%Pol = 0
0.0026	0.00026	5.39	16.15	17.16	12.56	1229	3.01	0	0.017
0.0024	0.00038	5.13	17.70	17.32	12.5	1613	3.03	0	0.028
0.003	0.00026	5.39	16.34	18.01	7.60	1039	2.90	0	0.013
0.0022	0.00038	5.10	14.79	18.25	6.55	1483	3.11	0	0.028
0.0024	0.00044	5.09	15.28	17.16	10.92	1532	2.96	0	0.036

Table 2: Nested Cross-Validation with Tuning Results. Each model, picked by the new automated selection procedure, is generally within the desired range for all metrics.

3.2 Bias Bootstrap Corrected CV

Nested CVT requires a significant amount of CPU time; we also implement BBC-CV, which provides model selection and evaluation while learning magnitudes fewer models than Nested CVT. However, BBC-CV relies on the CV procedure generating predictions which, as previously stated, our adaptation of CV does not do. We adapt Tsamardinos et al. [2018]'s procedure to utilize the LIG metrics rather instead predictions to calculate unbiased metrics. Rather than a matrix of out-of-sample predictions Π, we use a matrix of calculated metrics, T. We also use the notation of Tsamardinos et al. [2018] for bootstrap samples, T^b, and the set of data held out from the samples, $T^{/b}$.

Algorithm 7 Bootstrap Bias Corrected Cross-Validation

```
1: procedure BBC-CV-LIG(D, b, f, C)
2:     ⟨C*, T⟩ ← CVT − LIG(D, f, C)
3:     for i = 1:b do
4:         T^b ← sample with replacement N rows of Π
5:         T^/b ← rows of T not in T^/b
6:         j ← selectModel(T^b)      ▷ automated procedure selects the best configuration
7:         t_i ← T(j, :)    ▷ store average metrics for all models learned with configuration j
8:     end for
9:     return (1/b) Σ_{i=1}^{b} t_i
10: end procedure
```

Experimental Results

The BBC-CV procedure was run with $b = 500$. The procedure returned one model configuration selected by CVT and the automated selection procedure, along with estimations of all metrics for the configuration. The selected configuration, $\rho = 0.0024, \rho' = 0.00038$, was also in the set of configurations chosen by Nested CVT, and their estimations of metrics are compared in Table 3.

Again, the results of the BBC-CV procedure are promising. The metric values calculated by both procedures are generally similar, with a discrepancy on the reported test PSNE

Method	trErr	testErr	trPSNE	testPSNE	# Edges
BBC-CV	5.13	19.34	16.94	6.3	1476
Nested-CVT	5.13	17.70	17.32	12.5	1613

Table 3: A comparison of a subset of metric estimations generated by BBC-CV and Nested-CVT, with the hyperparameter value $\rho = 0.0024, \rho' = 0.00038$. Most metrics are quite similar, but the testing PSNE values (the percent of held-out testing bills which are identified as PSNE) differ.

value. Given that BBC-CV essentially only requires the same time as performing CVT, BBC-CV would be able to run on a much larger number of hyperparameter values to get a finer grain for hyperparameter values to select.

4 Automated Model Selection

In previous LIG research the process of model selection involved running the CVT procedure, capturing all resulting metrics from every configuration of ρ, ρ' tested, hand-generating graphs to chart the metrics, and manually deciding on the optimal values of ρ and ρ' to use on the final models. The error estimation procedures implemented here require model configuration selection to happen every time CVT is performed. This may be feasible for BBC-CV when the selection is only required once, but less so for Nested CVT: somebody would have to select a model every time an outer loop and resume execution of the program.

In order to make Nested CVT and BBC-CV simple to run, as well as to save researcher time, we implement a configurable algorithm for ranking models that can be called directly in Matlab code, making CVT, BBC-CV, and other model selection methods completely autonomous. The procedure works as such: each model configuration is penalize based on its metric values by comparing each metric to a predefined desired range of values, and scaling the penalty amounts differently per-metric. See Table 1 for a summary of all metrics calculated. Some metrics are not penalized at all as long as they sit in the desired range, whereas others aim to be minimized or maximized, and are penalized for being above the minimum for example, and further penalized for being outside of the desired range. Table 1 outlines how each metric is configured in the algorithm; these configurations should be tweaked if using a different dataset or group of senators, as these values were hand-tuned such that the algorithm made reasonable decisions for this particular set of roll call data.

Experimental Results

The algorithm's rankings of five model configurations performed in one run of CVT is shown in Table 5. A researcher would decide on the best model configuration by looking at the charts in Figure 3. We compare the algorithm's choices to what a researcher might decide based on looking at the charts.

The automated model selection procedure has made a sensible ranking. All of its choices put the number of edges in the desired range of 1000-1500. No decisions are made which

Metric	Range	Optimization	Outside penalty	Inside penalty
Training error	0-6	Min	4	0.5
Validation error	0-30	Min	4	0.25
Ideal point distance	0-5	Min	4	1
# Edges in LIG	1000-1500	Any	10	0
Training % PSNE	0-50	Max	2	.25
Validation % PSNE	0-50	Max	2	.25
Ideal points % = 0	0-50	Min	20	2
Bill polarities % = 0	0-50	Min	20	2

Table 4: The configuration of each metric within the automated model selection algorithm. Each metric is given a desired range, a value indicating whether they should be minimized or maximized (or if indifferent), a penalty scaling value for being outside of the range, and a penalty scaling value for being inside of the range.

rho	rhoPrime	penalty	trErr	valErr	trPSNE	valPSNE	nEdges	ipDist	ipZero	polZero
0.0024	0.00038	1.255	5.203	15.160	17.654	10.024	1451.3	3.152	0.00	0.032
0.0026	0.00044	1.256	5.266	14.956	17.194	10.254	1434.9	3.128	0.00	0.042
0.0028	0.00026	1.259	5.421	15.611	17.218	9.927	1178.5	3.009	0.00	0.020
0.003	0.00032	1.259	5.483	15.824	17.085	10.251	1087.5	2.967	0.00	0.024
0.003	0.00026	1.260	5.502	16.159	17.085	9.600	1100.8	2.922	0.00	0.020

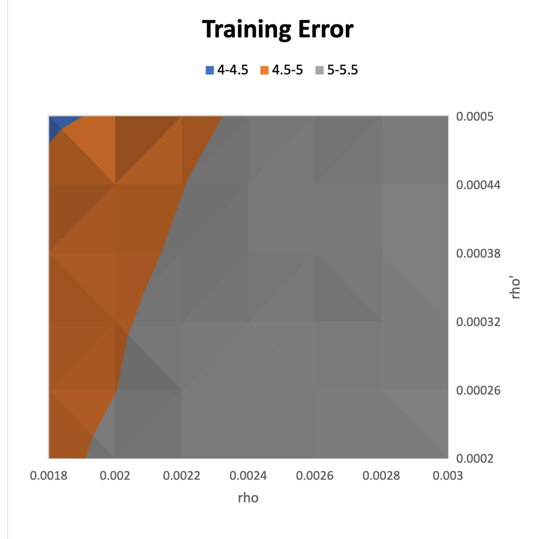
Table 5: The ranked choices of the automated model selection algorithm, from top to bottom. The "penalty" value is added for clarity, although it is not a metric.

completely disregard any metric in particular; all decisions are roughly middle-ground. However, there are a range of hyperparameter values in the top five ranked models. This may be indicative of the fact that there could be a number of local maxima for optimal hyperparameter values. The algorithm correctly avoids lower ρ values, which cause graphs that are too dense and have too many edges. It seems to generally be picking higher values within the range of possible ρ' values, possibly to try to find a middle ground between minimizing training and validation error, maximizing training and validation PSNE, and minimizing ideal point distance.

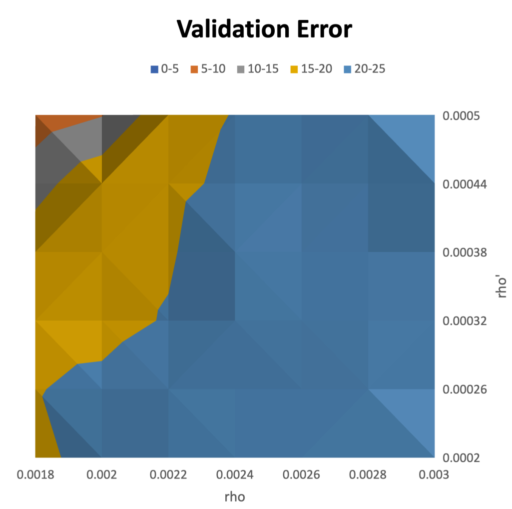
5 Estimating Polarity of Held Out Bills

In order to measure the best-response error or training/validation PSNE of a model, the polarities of all bills with which the model will be tested are needed. In the context of, for example, one fold of cross-validation, after a model is trained with the training folds its

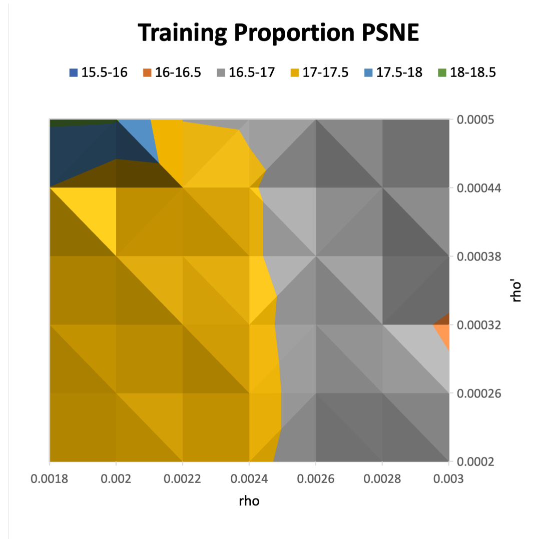
Figure 3: Results of several salient metrics for one inner 10-fold CVT run of nested CVT. Whereas these graphs are normally used for model selection, we compare the results of an automated model selection algorithm to what a person might otherwise select based on these figures.



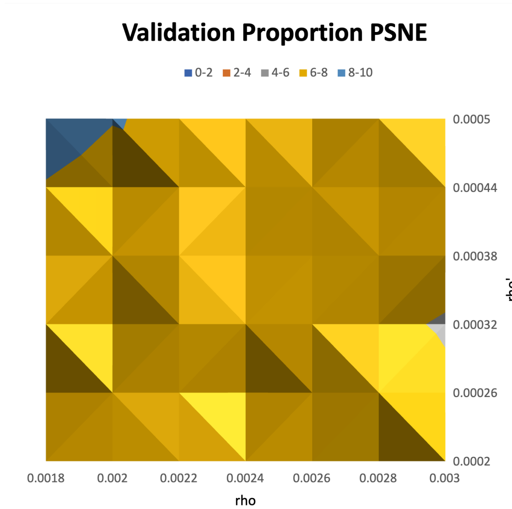
(a) Training error for CVT with $0.0018 < \rho < 0.003$, $0.0002 < \rho' < 0.0005$. This value should be minimized in selection.



(b) Validation Error for CVT with $0.0018 < \rho < 0.003$, $0.0002 < \rho' < 0.0005$. This value should be minimized in selection.

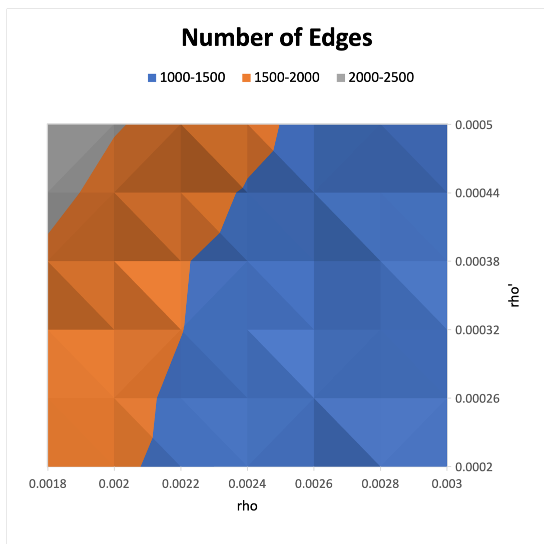


(c) Training PSNE for CVT with $0.0018 < \rho < 0.003$, $0.0002 < \rho' < 0.0005$. This value should be minimized in selection.

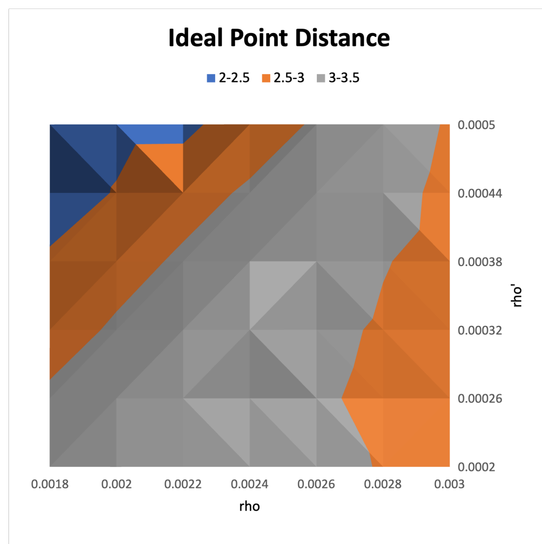


(d) Validation PSNE for CVT with $0.0018 < \rho < 0.003$, $0.0002 < \rho' < 0.0005$. This value should be minimized in selection.

validation error will be measured with the held out fold. However, only the polarities of the bills in the training set have been learned. Similarly, unlearned polarities of bills are



(e) Number of edges for CVT with $0.0018 < \rho < 0.003$, $0.0002 < \rho' < 0.0005$. The number of edges is acceptable within the 1000-1500 range.



(f) Ideal point distance to a known reference for CVT with $0.0018 < \rho < 0.003$, $0.0002 < \rho' < 0.0005$. This value should be minimized, but is acceptable within the 0-4 range.

needed for making predictions for voting outcomes on those bills. Although we don't know the polarity of some bills, we still know some existing data about all bills, such as subject codes, which is used to estimate the polarities of these held-out bills.

Irfan and Gordon [2018] use a straightforward way of estimating the unknown polarity of a bill l' using a whichever training set of bills was used to learn the model. The method is to compare the subject codes of the unknown bill l_u with the subject codes of each bill l_k with a known polarity, and use the known polarity of the bill l_k^* with subject codes that most closely match the subject codes of l_u as an estimate for its polarity. The similarity of two bills' subject codes is measured by taking the euclidean distance between bit vectors representing the presence or absence of each subject code in a bill. This method uses subject codes to substitute bill polarity; we call this method *Subject Code Substitution*, or *SC-S*.

The two processes where polarity must be estimated, model learning and prediction, are central to the project. If this estimation measure could be improved, the accuracy of results would improve as well. One potential problem with this subject code-based method is that, for example, a right-leaning bill l_k could have a left-leaning amendment l_u with unknown polarity, but both would have the same subject codes. A subject code based estimation algorithm might simply choose the polarity a_{l_k} as the estimation for a_{l_u}, which would clearly be a bad polarity estimate. We compare several new methods of unknown bill polarity estimation to this subject code-based version.

In addition to subject codes, another piece of information we have about all bills whether or not they have been trained with is which senators sponsor a bill. This sponsorship and cosponsorship information can be paired with the learned ideal points of each senator to, for example, average all sponsors' ideal points to get an rough party leaning estimate for a

bill. The first new method attempted is called *Subject Code Mean Ideal Point Substitution*, or *SC-MI-S*. This is a simple modification to the procedure above: only consider bills where the average sponsor ideal point leans in the same political direction as the unknown bill. For example, if $a'_l = -2$, use the polarity of the closest bill via subject code euclidean distance only among bills with $a_l < 0$. This procedure gets at the idea that we only want to substitute the polarity of one bill for another if they are similar in terms of political leaning.

The next method, *LRM*, is a linear regression model, where the presence or absence of each subject code as well as each senator was a feature. With 845 subject codes and 110 senators, the linear model has a total of 955 features. The model is trained on polarities learned when learning an LIG. Rather than calculating the mean sponsor ideal point, this model encodes each sponsor and their ideal point individually to make more data available for the model to learn on.

An additional concept that we use is rather than selecting the single bill which was closest by some measure and using its polarity as a stand-in, we calculate a weighted average of all known bill polarities with the weights determined by a similarity measure. In this next method, *Subject Code, Mean Ideal Point Weighted Average*, or *SC-MI-W*, we multiply the average sponsor ideal point p by the subject code bit vector $\mathbf{s}$, and take the euclidean distance of the two vectors. This distance measure, $dist(k, u) = euclidean(S_k \cdot \mathbf{p}_k, S_u \cdot \mathbf{p}_u)$, is then used in a power law-like formula to calculate the weighted average, where j is the number of known-polarity bills:

$$\sum_{n=1}^{j} pol_k \cdot \delta^{dist(k,u)} / \sum_{n=1}^{j} \delta^{dist(k,u)}. \tag{13}$$

The next method, *Subject Code Ideal Point Weighted Average*, or *SC-I-W*, utilizes subject code data as well as all sponsors' individual ideal point values. The bill sponsorship bit vector, i.e. the vector of length equal to the number of senators where a 1 represents that senator being a bill sponsor or cosponsor, is element-wise multiplied by a vector of all senators' ideal points. This scales the sponsoring senators' 1 by their ideal point. This vector is then element-wise multiplied by the subject code vector, resulting in a matrix where each row represents a subject code, each column a potentially sponsoring senator. If the bill does not have a subject code, then its row is full of zeros; otherwise its row is the sponsorship bit vector scaled by each sponsor's ideal point. The distance measure in *SC-MI-W* is replaced by the euclidean distance between these two matrices, in which all subject code, sponsorship, and ideal point data is encoded. The same weighted average is calculated.

The final method uses the same matrix calculated in *SC-I-W*, but uses substitution and just picks the polarity of the bill with the closest distance rather than calculating a weighted average. Thus it is called *SC-I-S*.

Full Name	Acronym
Subject Code Substitution	*SC-S*
Subject Code Mean Ideal Point Substitution	*SC-MI-S*
Linear Regression Model	*LRM*
Subject Code Mean Ideal Point Weighted Average	*SC-MI-W*
Subject Code Ideal Point Weighted Average	*SC-I-W*
Subject Code Ideal Point Substitution	*SC-I-S*

Table 6: A summary of metrics captured in the Cross-Validation process. These metrics are used for model selection and evaluation. For example, the validation error would be used to pick among ρ and ρ' values to find a model configuration that is likely to model the data used for learning.

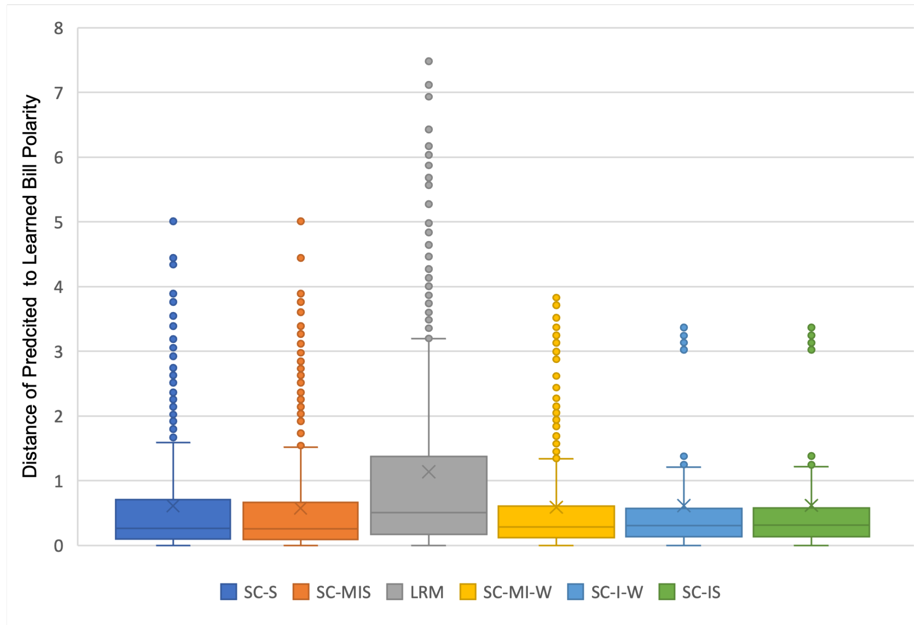


Figure 4: Box-and-Whisker plot of the distribution of polarity estimation distance from learned values. Polarities were learned with hyperparameter values $\rho = 0.002, \rho' = 0.0002$, and estimations calculated with $\delta = 0.5$.

Experimental Results

Figure 4 provides a summary of descriptive statistics for each algorithm implemented, the X-axis showing the distance of estimated polarities to their learned values. *SC-MI-S* makes a slight improvement on *SC-S*, slightly improving the IQR, but doesn't seem to alter the predictions very much. *LRM* performs poorly compared to all other versions; this is likely because we force complex, potentially nonlinear data into a linear model. The model pre-

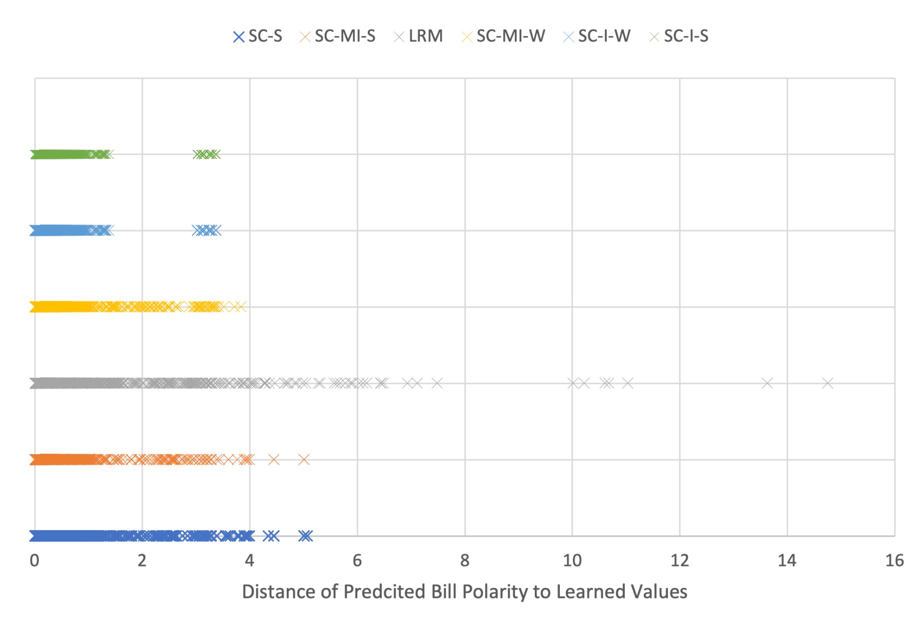


Figure 5: Distribution of polarity estimation distance from learned values. Polarities were learned with hyperparameter values $\rho = 0.002, \rho' = 0.0002$, and estimations calculated with $\delta = 0.5$.

dicts a relatively large number of outliers. *SC-MI-W* narrows the distribution and reigns in outliers compared to *SC-S*, and appears to make slightly improved predictions. *SC-I-W* and *SC-I-S* appear to have notably better performance than *SC-S* as well as *SC-MI-W*, despite an odd gap in distance values. However, after checking the raw data it was determined that both versions tend to predict polarity values near 0 on almost every bill. More investigation is required to determine exactly why these last two models behave this way, but for this reason, we are not considering using them to replace *SC-S*. At the very least, this finding demonstrates that there is much more room for improvement in the realm of estimating unknown bill polarity.

SC-MI-W appears to have better outlier performance and a preferable distribution of polarity estimate errors relative to learned values than *SC-S*. Figure 5 confirms this result. In the areas where polarity estimates are used, mainly calculating the training and validation errors of a model and equilibrium computation, the algorithm *SC-MI-W* was substituted in for the previous algorithm *SC-S*. These methods were also tested with a range of δ values as well as several different model configurations (see A for figures); the results are consistent across all δ values and configurations.

6 Future Work

With more robust error estimation methods, robust comparisons between previous iterations of LIG projects can easily be made. Either Nested CVT or BBC-CV can be used to

compare, for example, the performance metrics of the LIG models with and without ideal points added, or with and without clustering. These comparisons are allowed by the fact that the reported metrics from BBC-CV or Nested CVT will be less biased than simply using CVT. Additional work could include testing the biases of Nested CVT and BBC-CV's metrics with artificial datasets, as well as comparing their reported values to the validation measured returned by CVT.

Additionally, these methods could be applied to making LIG predictions. This would need to be done thoughtfully and carefully, as the computational hardness of computing equilibria is proven, and practically can take very a long duration. But these methods could be applied as an improvement on, for example, holding out a subset of three bills as was done in Irfan and Gordon [2018]; folds of unseen bills could be predicted rather than just a select few. If improvements are made on the computation of equilibria, or if the LIG model gains new variables which allow it to further decrease $\pi(\mathcal{G})$, then equilibria computation will be reliably faster and these methods can be utilized with prediction error.

Lastly, it is clear that bill polarity estimates can be further improved. Logistic regression or otherwise nonlinear models may be able to better model the political spectrum of bill polarities and their subject codes and sponsor ideal points. Text-analysis of bills could also make further improvements.

References

Bradley Efron and Robert Tibshirani. *An introduction to the bootstrap.* Number 57 in Monographs on statistics and applied probability. Chapman & Hall, 1993. ISBN 978-0-412-04231-7.

Sean M Gerrish. *Applications of latent variable models in modeling influence and decision making.* PhD thesis, Princeton University, 2013.

Jean Honorio and Luis E Ortiz. Learning the structure and parameters of large-population graphical games from behavioral data. *Journal of Machine Learning Research*, 16(1): 1157–1210, 2015.

Mohammad Irfan and Luis Ortiz. A game-theoretic approach to influence in networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 25(1):688–694, 2011. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v25i1.7884. URL https://ojs.aaai.org/index.php/AAAI/article/view/7884.

Mohammad T Irfan and Tucker Gordon. The power of context in networks: Ideal point models with social interactions. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 910–918, 2018.

Mohammad T. Irfan and Luis E. Ortiz. On influence, stable behavior, and the most influential individuals in networks: A game-theoretic approach. *Artificial Intelligence*, 215:79–119, 2014. ISSN 00043702. doi: 10.1016/j.artint.2014.06.004. URL https://linkinghub.elsevier.com/retrieve/pii/S0004370214000812.

Ji-Hyun Kim. Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics & Data Analysis*, 53(11):3735–3745, 2009. ISSN 01679473. doi: 10.1016/j.csda.2009.04.009. URL https://linkinghub.elsevier.com/retrieve/pii/S0167947309001601.

Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

Luca Ostertag-Hill. Ideal point models with social interactions applied to spheres of legislation. Honors project report, Bowdoin College, 2020.

Andrew C. Phillips, Mohammad T. Irfan, and Luca Ostertag-Hill. Spheres of legislation: polarization and most influential nodes in behavioral context. *Computational Social Networks*, 8(1), 2021. ISSN 2197-4314. doi: 10.1186/s40649-021-00091-2. URL https://computationalsocialnetworks.springeropen.com/articles/10.1186/s40649-021-00091-2.

Keith Poole, Howard Rosenthanthal, and Christopher Hare. Alpha-NOMINATE applied to the 114th senate, 2015. URL https://voteviewblog.com/2015/08/16/alpha-nominate-applied-to-the-114th-senate/.

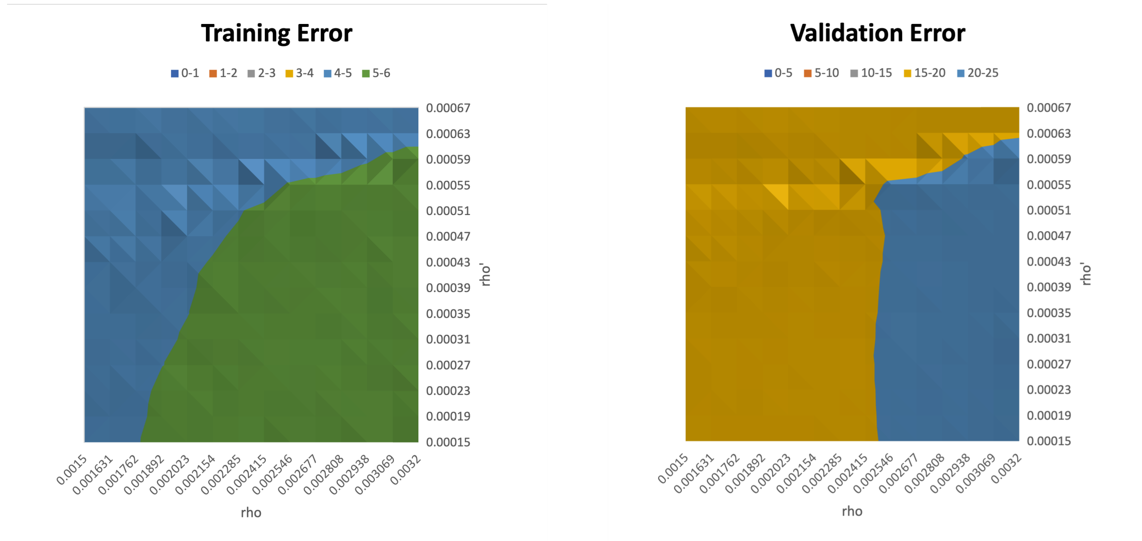
Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*, 2018.

Ioannis Tsamardinos, Elissavet Greasidou, and Giorgos Borboudakis. Bootstrapping the out-of-sample predictions for efficient and accurate cross-validation. *Machine Learning*, 107(12):1895–1922, 2018. ISSN 0885-6125, 1573-0565. doi: 10.1007/s10994-018-5714-4. URL `http://link.springer.com/10.1007/s10994-018-5714-4`.

Sudhir Varma and Richard Simon. Bias in error estimation when using cross-validation for model selection. *BMC Bioinformatics*, 7(1), 2006. ISSN 1471-2105. doi: 10.1186/1471-2105-7-91. URL `https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-7-91`.

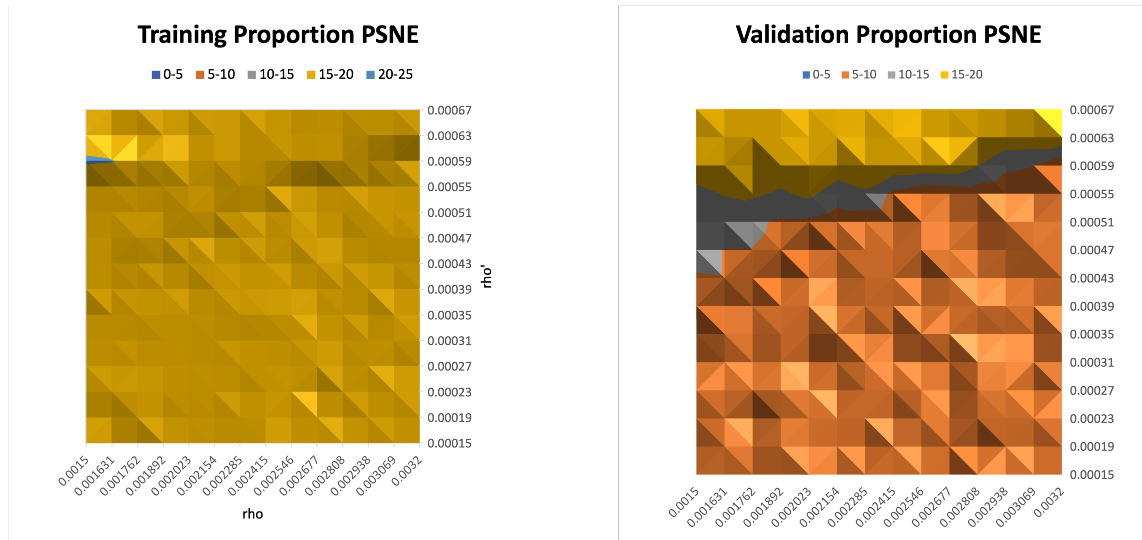
A Appendix

Figure 6: 10-fold CVT results across a wider range of hyperparameter values. Charts for the proportion of learned ideal points and polarities which equal zero are included here, whereas on the runs Nested CVT and BBC-CV, the narrower range of hyperparameter values meant that these proportions were essentially zero across the entire range.



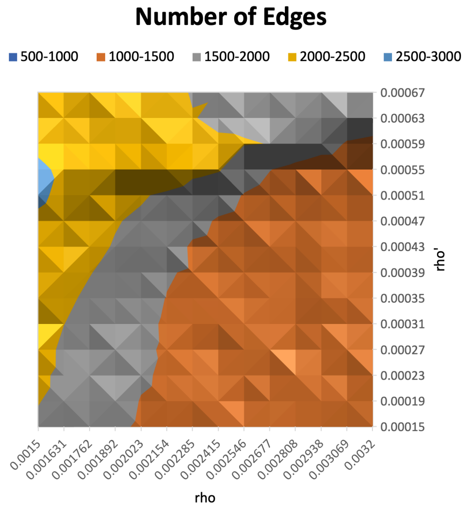
(a) Training error for CVT with $0.0015 < \rho < 0.0032$, $0.00015 < \rho' < 0.00067$.

(b) Validation Error for CVT with $0.0015 < \rho < 0.0032$, $0.00015 < \rho' < 0.00067$.

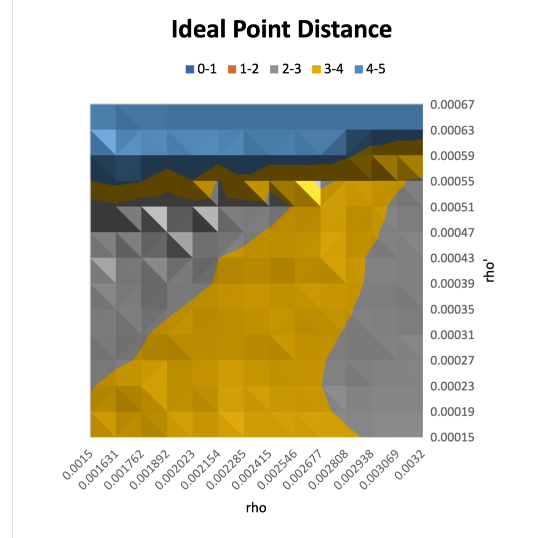


(c) Training PSNE for CVT with $0.0015 < \rho < 0.0032$, $0.00015 < \rho' < 0.00067$.

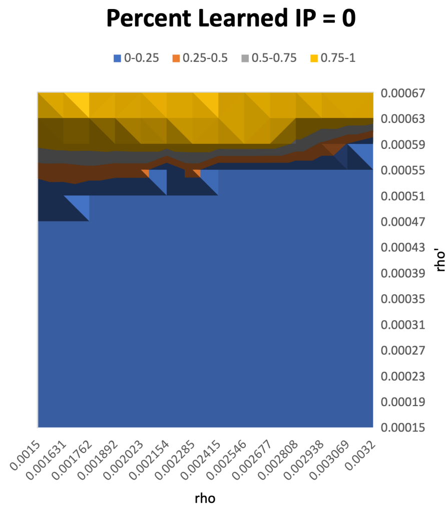
(d) Validation PSNE for CVT with $0.0015 < \rho < 0.0032$, $0.00015 < \rho' < 0.00067$.



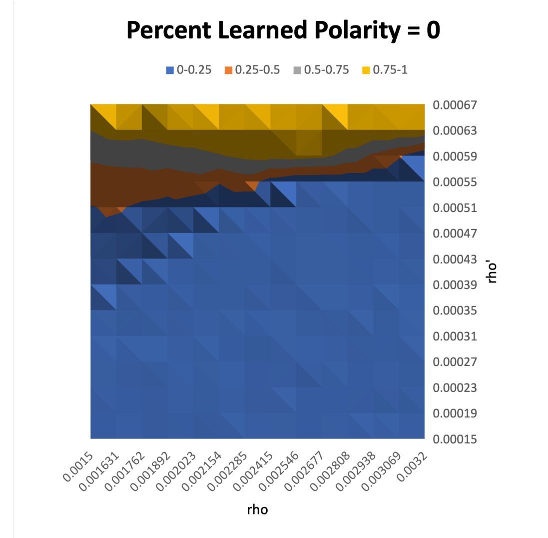
(e) Number of edges for CVT with $0.0015 < \rho < 0.0032$, $0.00015 < \rho' < 0.00067$.



(f) Ideal point distance to a set of known reference values for CVT with $0.0015 < \rho < 0.0032$, $0.00015 < \rho' < 0.00067$.



(g) Percent learned ideal points = 0 for CVT with $0.0015 < \rho < 0.0032$, $0.00015 < \rho' < 0.00067$.



(h) Percent learned bill polarities = 0 for CVT with $0.0015 < \rho < 0.0032$, $0.00015 < \rho' < 0.00067$.

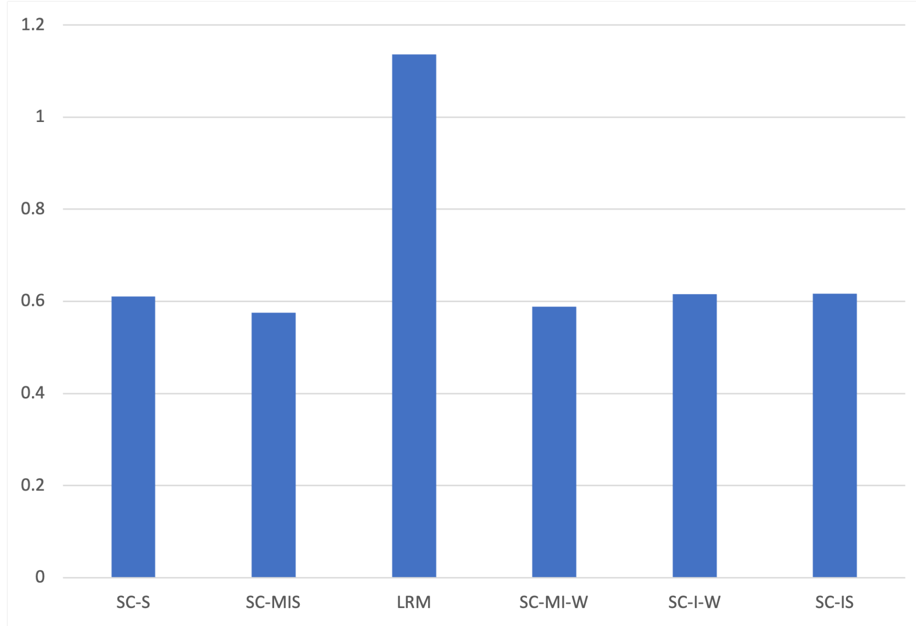


Figure 7: Mean polarity estimation distance from learned values. Polarities were learned with hyperparameter values $\rho = 0.002, \rho' = 0.0002$, and estimations calculated with $\delta = 0.5$.

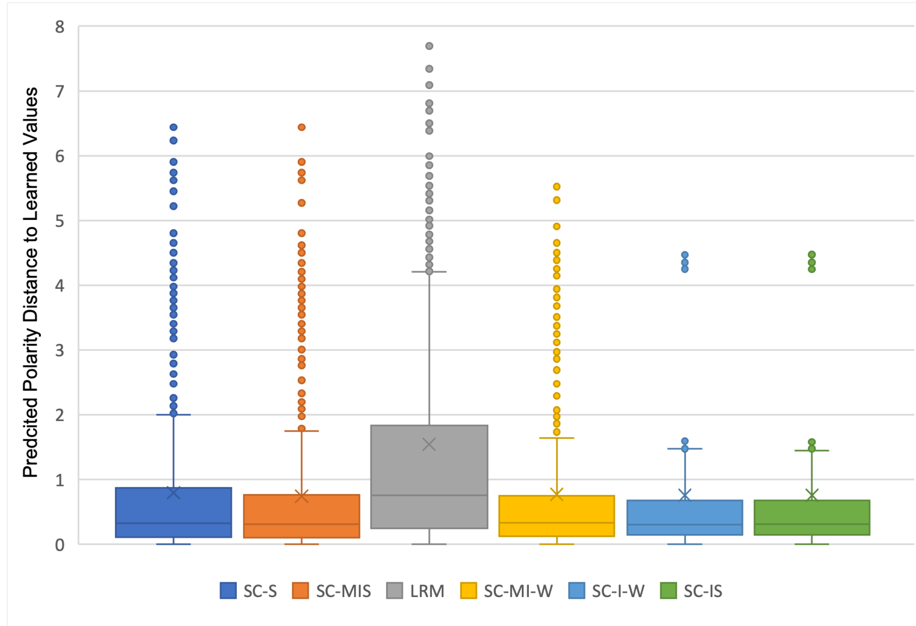


Figure 8: Box-and-Whisker plot of polarity estimation distance from learned values. Polarities were learned with hyperparameter values $\rho = 0.002, \rho' = 0.00013$, and estimations calculated with $\delta = 0.5$.

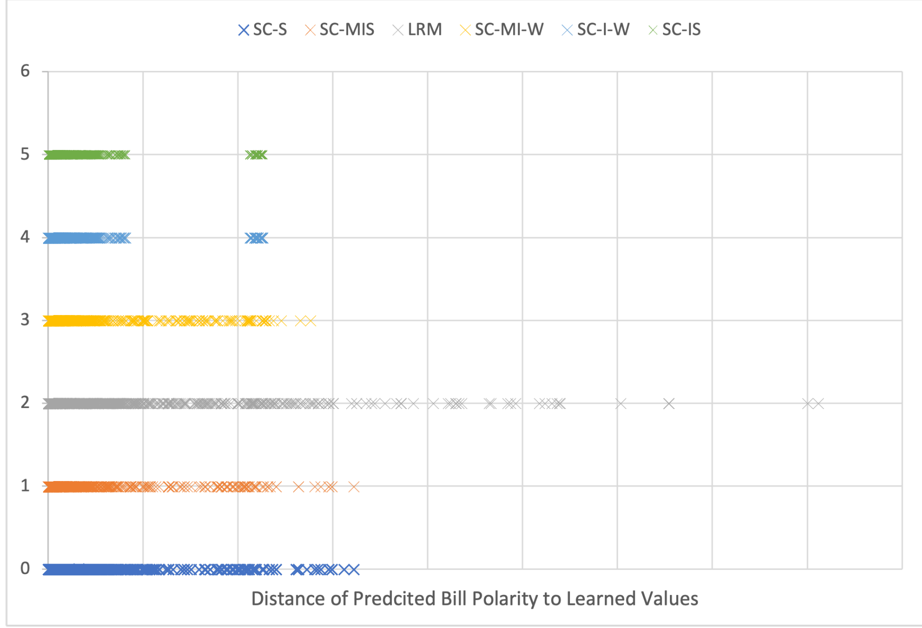


Figure 9: Distribution of polarity estimation distance from learned values. Polarities were learned with hyperparameter values $\rho = 0.002, \rho' = 0.00013$, and estimations calculated with $\delta = 0.5$.

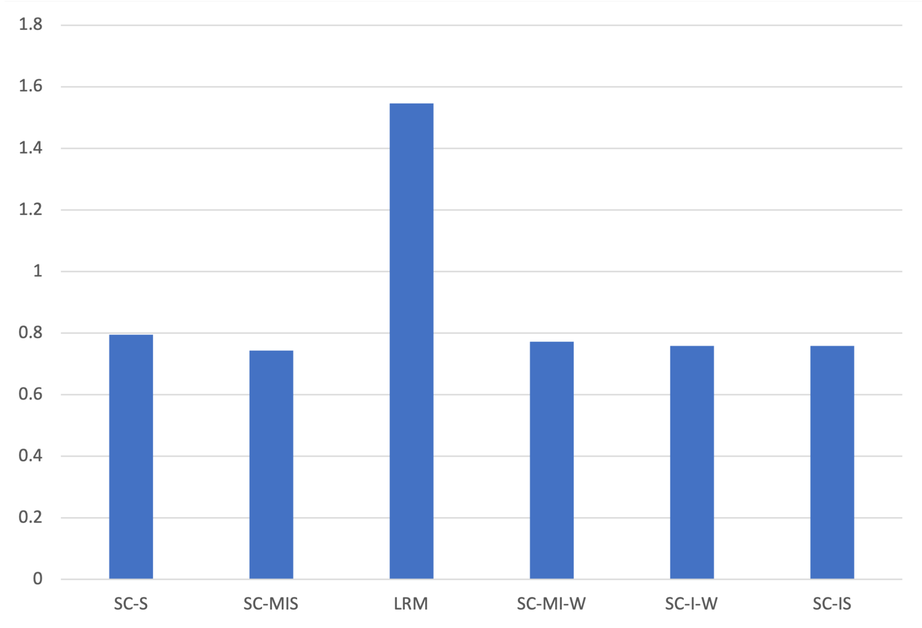


Figure 10: Mean polarity estimation distance from learned values. Polarities were learned with hyperparameter values $\rho = 0.002, \rho' = 0.00013$, and estimations calculated with $\delta = 0.5$.

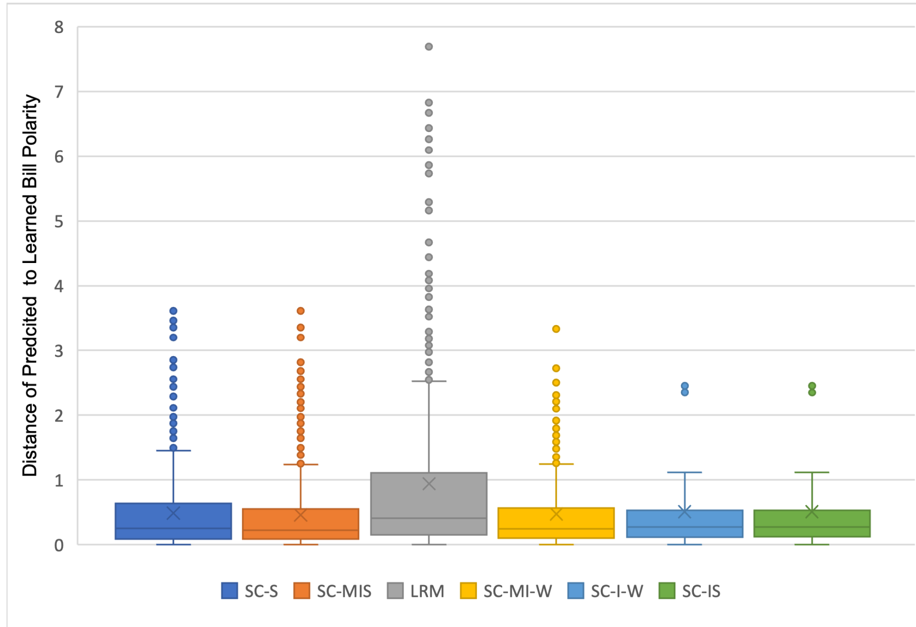


Figure 11: Box-and-Whisker plot of polarity estimation distance from learned values. Polarities were learned with hyperparameter values $\rho = 0.002, \rho' = 0.0003$, and estimations calculated with $\delta = 0.5$.

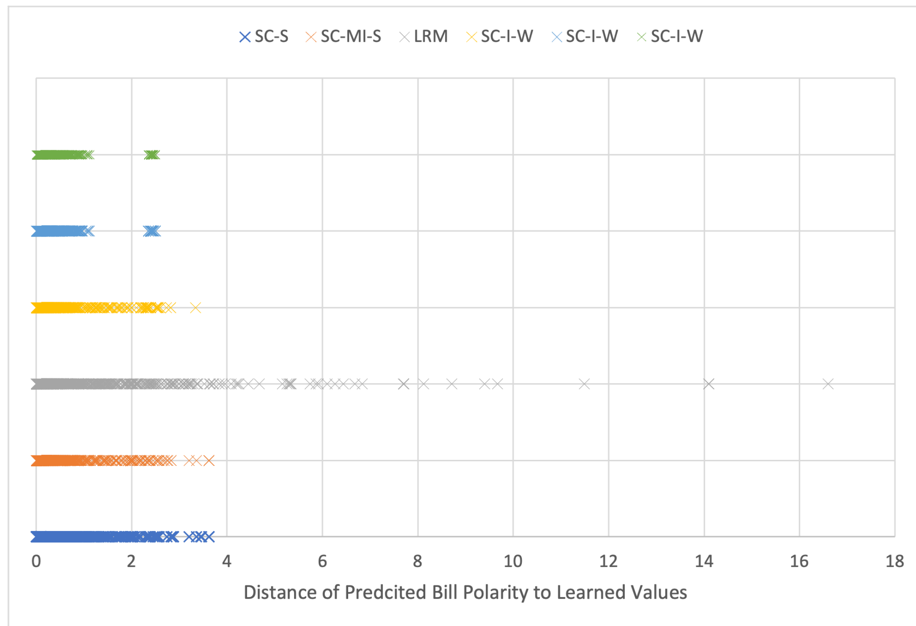


Figure 12: Distribution of polarity estimation distance from learned values. Polarities were learned with hyperparameter values $\rho = 0.002, \rho' = 0.0003$, and estimations calculated with $\delta = 0.5$.

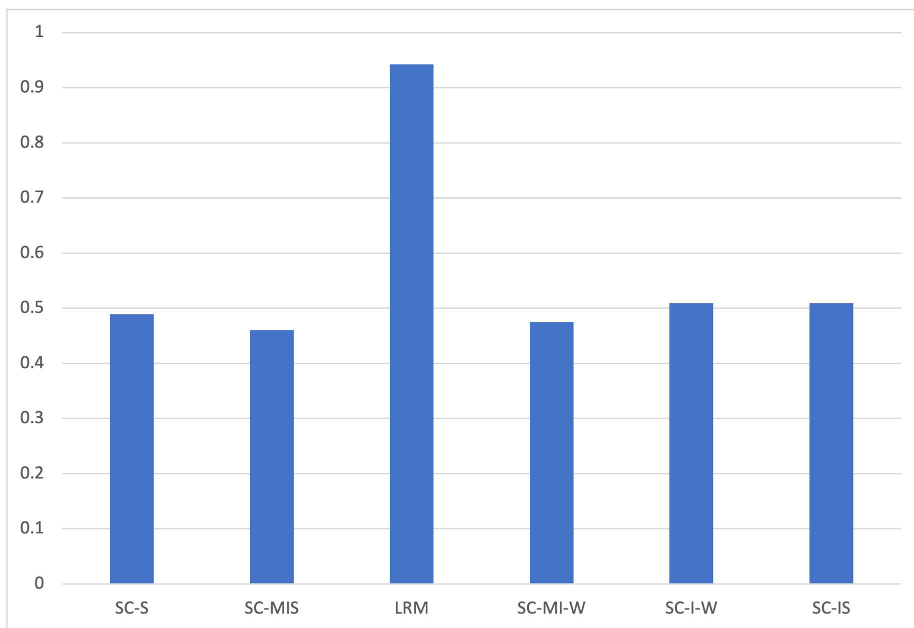


Figure 13: Mean polarity estimation distance from learned values. Polarities were learned with hyperparameter values $\rho = 0.002$, $\rho' = 0.0003$, and estimations calculated with $\delta = 0.5$.