Indian Parliamentary Election Result Predictor

DSE-302, Data Science and Machine Learning
Course Instructor: Dr. Kushal Kumar Shah

Anirudh Kalla
IISER Bhopal

Contents

1 Introduction

India is one of the most diverse and culturally polylithic countries in the world. There are deep rooted division in the country along many ascribed/merited lines. From Linguistic divisions, to educational disparity, India is a melting pot of people from all walks of life. To predict any trend in this broad-spectrum country, is oft nothing more than a wild goose chase. The only way one can perform this gargantuan task of mapping voting trends of India, is by careful consideration of the available features presented to us in the form of raw data.

2 Objective

In this project, we aim to understand and subsequently predict Indian Parliamentary election trends using (modded) Election Commission data from the 2019 General Election. Instead of going for a brute force approach of using all features, we create new features from existing data to enhance the model learning. We will use a large array of ML algorithms to see which ones work best in this real life test dataset. Our dataset only has 2263 data points with 18 base data-labels, so one might expect traditional ML techniques to work well, along with some shallow neural networks. These algorithms and architectures are discussed in great detail below.

3 Dataset Description

We are using official Election Commission data along with inputs from data mined off myneta.info for the project. Our dataset has 18 data labels and 2263 data points. These data labels include, but are not limited to, information on assets/liabilities, educational qualification, votes polled, and criminal cases. These, among others, are quite instrumental in deciding the fate of the candidate as the electors slowly start to look beyond old caste equations that have traditionally governed the winners in a significant portion of the country.

We have also used these raw data labels to generate new features which are more useful for the ML Models we deploy. This process is analogues to a

change of basis in a Vector Space. While the change of basis doesn't actually have an effect on the eigenvalues of an object, say, the Hamiltonian, using orthonormal bases can significantly reduce our work and increase "efficiency". The same information, encoded as a different feature map can significantly boost model performance by enhancing the way the optimization algorithm performs "learning".

4 Machine Learning Methods

We will subdivide this header into the individual models we used to study them in further detail:-
Note: All figures pertaining to performance metrics and scalability are added right at the end to ensure proper formatting and readability.

4.1 Traditional Machine Learning

4.1.1 Linear Classifier

Since Linear Regression is not actually a classification method, we need to use a concept called "thresholding". Since the sum of the probability of a Lok Sabha Seat being won by one of the many candidates is equal to one, we assign a threshold probability above which the datapoint is may be considered as belonging to a certain class. Since our data is scaled from 0-1, we also bypass the issue of the Ridge Classifier running out of the imposed probabilistic bound. We must however remember that Logistic Regression is far more suitable for Multi-Class Classification as our outcome label is discrete, not continuous. The Linear Ridge Classification algorithm that we have used, employs the linear least squares loss with L-2 regularization. The Objective Function for this method is:-

$$||Y - X * \theta||_2^2 + \alpha * ||w||_2^2 \tag{1}$$

4.1.2 Logistic Regression (Classifier)

This method is quite a misnomer since it's actually a classifier. This is the method of choice for Multi-Class Classification problems where we require some inference from the results as it is fully capable of giving us the probabilities too. One of the major advantages of Logistic Regression is that

it allows the evaluation of multiple explanatory variables by extension of the basic principles. Moreover, It makes no assumptions about distributions of classes in feature space making it a very efficient method for multi-class problems which have linearly separable boundaries. The ease of interpretation in Logistic Regression models make it particularly useful for tasks like ours where we aim to derive intelligent inferences from the model. It gives us a good overview of how our feature space is minimally correlated with low inter-feature dependency. One can easily infer that features like age and gender have minimum impact on each other and can thereby be used for classification w/o error.

$$P = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... \beta_n X_n)}} = \frac{1}{1 + e^{-(\beta_0 + \sum \theta_i X_i)}} \tag{2}$$

4.1.3 K Nearest Neighbors

As we can see from the performance graph, this is one of the fastest methods for classification out there. For us, this method is particularly useful as we can add/remove data points seamlessly without losing accuracy. When working with election data, dynamically adding new information (i.e contesting candidates) is quite imperative. Moreover, since KNNs are examples of Instance Based Learning models, they do not have a conventional training period. They make faster, real-time predictions. Another reason why this is such a light and effective model is that it is really easy to implement, requiring only two parameters, K and euclidean distance. Since our data is also scaled in the pre-processing step, KNNs work even better, as scaling prevents accuracy reduction due to extreme outliers/abnormal values among other things. The only major disadvantage that KNNs bring to the table is that they tend to slow down quite rapidly with increase in dimensions.

4.1.4 Support Vector Machines (Classifier)

Support Vector Machines are extremely helpful when we are handling election data. There are a ton of places where the data we encounter has massive outliers. These abnormal data points generate a lot of inaccuracy in other methods like logistic regression etc. As one can clearly see, our data has a lot of outliers. For example, when we look at the asset-vs-liability plot, we see a lot of "extreme" points on the graph. These points can be subdued, so to say, using SVM Classifiers. There are, however, a few downsides to using SVM

Classifiers. SVMs are traditionally binary classifiers with no inherent support for multi-class classification. In a similar vein, SVMs traditionally classify using linear decision boundaries (we can use non-linear kernels though). They are also quite hard to interpret and analyse, especially in comparison with other methods like logistic regression, which give us the probability values as well. Finally, they too are more suited for low dimensional data, so if we have a larger feature space, we can expect some performance degradation.

4.1.5 Random Forest Classifier

This is arguably one of the best models for the problem we are tackling. It provides great support for non-linear decision boundaries (our data has piecewise non-linearities) and is very well suited for high dimensional data. It is the method of choice for multi-class classification problems as it follows a rule based approach that can easily support these problems. Also, this model helps in managing outliers and noise with great ease as these abnormal data points have no bearing on the "decision boundary" due to the rule based approach preventing the decision boundary from getting skewed. Using Random Forest is further more useful as it helps in working around the main disadvantage of decision trees, i.e overfitting. Using a large number of trees helps in overcoming the high variance/low bias that one particular tree may have. We may, however, face a few issues due to the slow and computationally heavy nature of this algorithm. It is almost impossible to interpret and as mentioned before, quite slow. But as we can see from the performance and accuracy charts, it has phenomenal performance for our data and is very well suited for this project.

4.1.6 AdaBoost Classifier

This partiular algorithm was not directly covered in the course but seemed like a good fit for the problem at hand. It is a type of ensemble learning method, namely boosting. Here, a large number of weak classifiers are overlain **sequentially** to make a strong classifier. This is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first model. Models are added until the training set is predicted perfectly or a maximum number of models are added. The most suited and therefore most common algorithm used with AdaBoost are decision trees with one level. Because these trees are so short and only

contain one decision for classification, they are often called decision stumps. Predictions are made by calculating the weighted average of the weak classifiers. The ingenuity of this method really appealed to me and I couldn't help but add this model even though it wasn't covered in course material. As we can see, the gamble of using this played off well. This method gave 100% accuracy with fairly quick execution times.

4.2 Neural Networks

4.2.1 Multi-Layer Perceptron

This problem is quite well placed to be solved using Neural Networks. While the dataset isn't quite as large as one would like, it comfortably makes the cut for using Neural Networks. Neural Networks have an ability to learn complex decision boundaries, making them useful for us, as we have piece-wise non-linearity in some attributes of our data. Along with this, Neural Networks don't place any restrictions on the distribution of our input variables making them adaptable and versatile. They, like KNNs, are quite robust with reference to addition/removal of data points ensuring performance metrics don't suffer. A prospective downside with neural networks is that in our pursuit to tune them most appropriately, we may end up causing it to overfit the training data. We, however, use multifold cross-validation to root out this issue and ensure good testing accuracy. One thing we must be cautious of, is that using the optimal number of hidden layers (and neurons per layer) is imperative. Model performance can drastically go down if we use an arbitrarily large/small number of hidden layers/neurons as this is a prime cause of overfitting/underfitting. So hyperparamters must be tuned properly. As for our case, since our data isn't wholly/vastly non-linear, and neither do we have an over-abundance of data points, using a lot of hidden layers affects the performance (testing) accuracy negatively. Lastly, the real functioning of Neural Networks and how/what they learn is quite a mystery. Being what one would call a "Black Box" algorithm, they produce hard to interpret results, making them slightly less lucrative for problems where large inference/insight is required from the predicted labels.

4.2.2 Stochastic Gradient Descent

In our quest to find the best of all methods, it was only natural that we try out Gradient Descent, or more specifically, *Stochastic* Gradient Descent. As we can however see, the experiment did not yield very good results. This is most likely due to the following reasons. The direction in which the gradient was computed veered off into the wrong direction as the model parameters are updated after every step. Since we have 16 features (after pre-processing and augmenting the original data labels), our gradient is computed in a 16 dimensional vector space. So even if one partial derivative starts to move in the wrong direction (along one feature axis, i.e), the convergence may happen at the wrong minima. Also, vector implementation doesn't really help us here as we update model parameters after every step (epoch) anyway making the convergence very "noisy".

4.3 Unsupervised Learning Methods

4.3.1 K-Means Clustering

In this part of the project, it became quite evident that Unsupervised Learning was not very well suited for our problem. The primary issue, it seems, is the high dimensionality of the data. As the number of dimensions increases, a distance-based similarity measure converges to a constant value between any given data-points. The only way to bypass this, is to use Principal Component Analysis. By using PCA, we can project the high dimensional data space into an orthogonal lower dimensional space. Even this, however, didn't suffice, since our data anyway has low correlation between features. This is something we mentioned quite early on in our analysis as well. This is also a reason why some of our supervised learning methods performed very well. This Curse of *Irreducible* Dimensionality has come back to haunt us. Another reason why our clustering algorithm didn't work very well was that our data has data of varying sizes and density as the K-Means algorithm has problems classifying such data. We still obtained a decent Silhouette Score on our data, proving that there is scope for improvement and our roadblock (high dimensionality and varying data density) isn't a permanent, unsolvable issue.

4.3.2 Gaussian Mixture Method

Using this method, we obtained slightly better results, as compared to K-Means Clustering. This improvement can be attributed to the following reasons. GMM is a lot more flexible in terms of cluster covariance. So having high variational data density does not affect our model as adversely as it does in the case of K-Means Clustering. The second major benefit of GMM is that it accommodates mixed membership.In GMM a point belongs to each cluster to a different degree. The degree to which a point belongs to a given cluster is based on the probability of the point being generated from each cluster's (multivariate) normal distribution, with cluster center as the distribution's mean and cluster covariance as its covariance. Having inherent support for overlapping clusters, makes this model significantly more useful for us since our data has independent, yet overlapping attributes. Explaining with an example, we have, say, a candidate that is a part of a cluster that (loosely) correlates to age. This candidate may well also be equally likely to be part of a cluster which represents a specific educational qualification. Allowing mixed membership will help us in retaining as much information, about this candidate, as possible. Whereas in K-Means Clustering, we would necessarily have to choose one of the clusters and thereby (in some sense) lose a bit of information about this data point (candidate).

5 Comparative Analysis

After having gone through all these methods in great detail and analysing their usefulness for our project, we can finally conclude the analysis based off a few important pointers.

1. If we have consistent and mostly complete data, using ensemble learning can really improve performance as it weighs out the cons of each (singular) classifier and ensures our model doesn't overfit. This explains why boosting algorithms like AdaBoost perform well and are a good match for our problem.

2. If we have clean data, without many outliers, logistic regression is still a method of choice as it is the most insightful and easy to interpret of them all. This is majorly due to how it can give us probability values

too. The only downside is that if we have strong non-linear tendencies to our data, Logistic Regression may suffer.

3. If we have enough time and abundant resources, Random Forest Classifiers do not disappoint. They capture non-linearities very well and can very well handle high-dimensional data. They are, however hard to interpret and may have a slight overfitting tendency (This can be rooted out by using more trees, though).

4. Neural Networks (Multi Layer Perceptron, in our case) seem to be a magic bullet for problems like ours. They have inherent support for non-linear data and can tackle outliers too. We can tweak the hyperparameters to obtain great results for most problems in the ML space. The only thing one must be cautious of is the fact that these models usually only work well with a large number of data points and suffer from a lack of interpretability, since they are blackbox algorithms.

5. If speed (with reasonably good accuracy) is a criterion for selecting models, SVMs should definitely make the cut as they are among the quickest models out there. SVM Classifiers, however, really suffer when we have high dimensional data with a multi-class classification problem.

The overall verdict would be to use a neural network based model since they provide the right balance of all that we require, from scalability to reliable performance time and time again. This, coupled with the fact that our data will only increase with time exemplifies the reason why NN models are apt for this problem. More (consistent) data is always a blessing and mostly so for data-hungry Neural Networks.

6 Implications/Future Work Potential

These findings can be used for tackling other real-life problems too, like social dynamics analysis, public perception modeling for PR firms etc
I plan to extend this project in the following ways:-

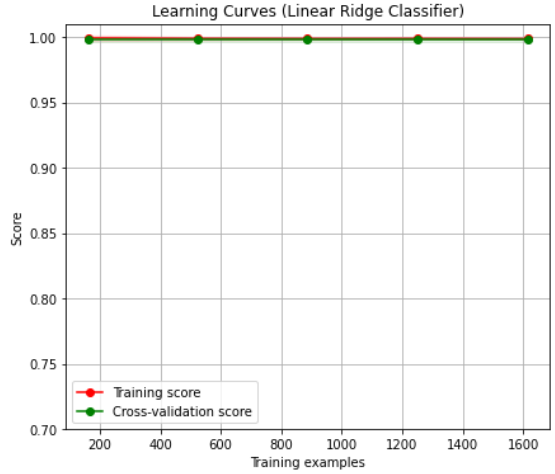
1. Increase the use-case of the model. We can slowly extend this project to encompass assembly elections and do district by district modeling of the public mood. This includes anti/pro incumbency sentiments,

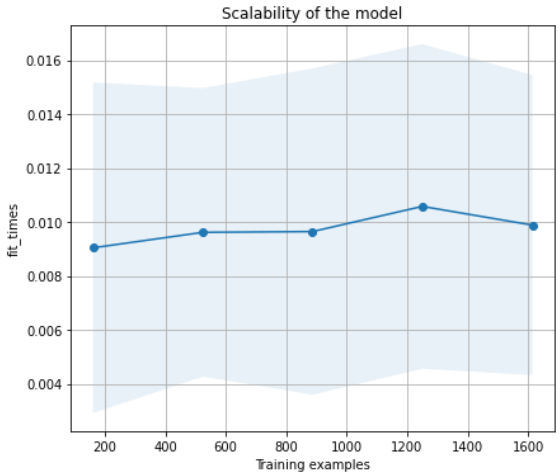
death of leaders (sympathy wave phenomenon), polarizing figures and speeches in the run-up to elections (leading to votebank consolidation).

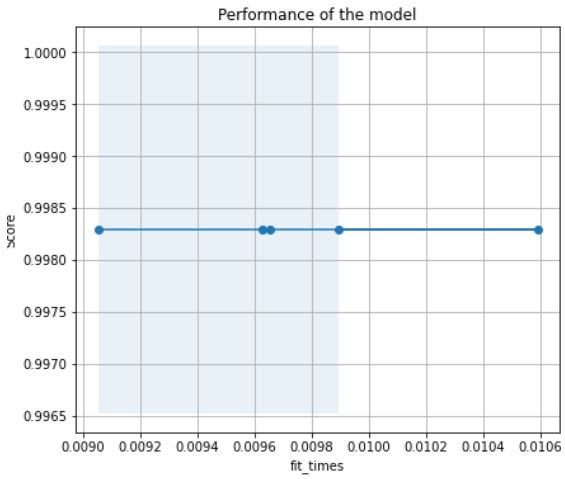
2. Include region specific data, like caste equations for western UP, popular culture relevance in Tamil Nadu (MGR, Jayalalitha and so on) etc. This will help us in capturing the "Hawaa" as Mr Ramdas Athawale beautifully put it.

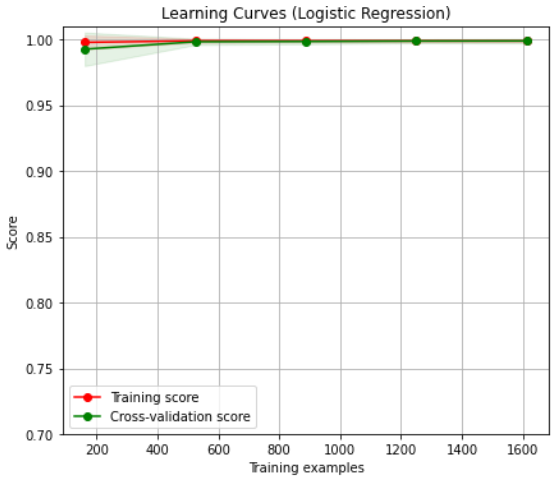
In the end, I'd just like to conclude by saying that this is a rapidly evolving field where there are infinite possibilities to create endless wonders. Thank you for joining me in this lovely adventure. 'Twas an absolute delight.

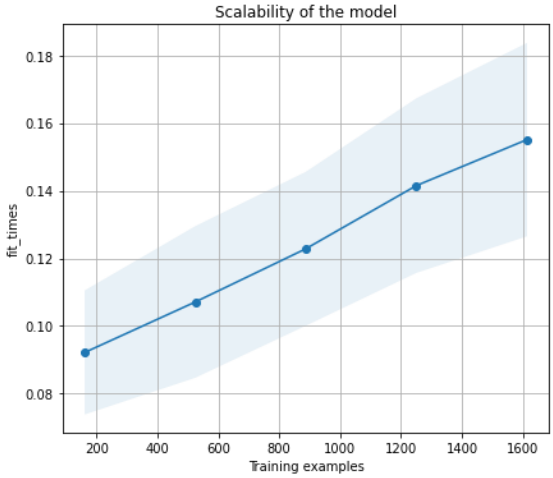
Appendix I

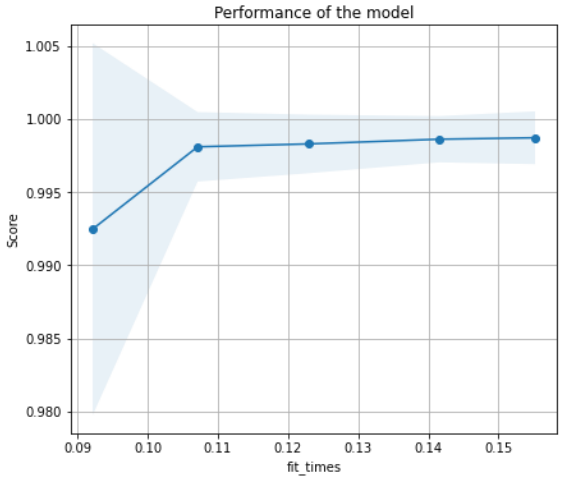

Learning Curves (Linear Ridge Classifier)
Score
Training examples
Training score
Cross-validation score

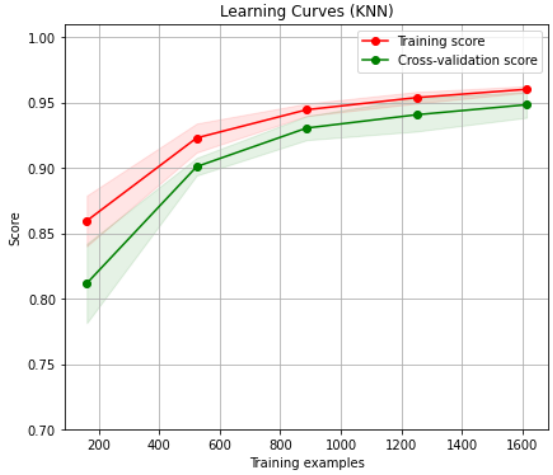

Scalability of the model
fit_times
Training examples

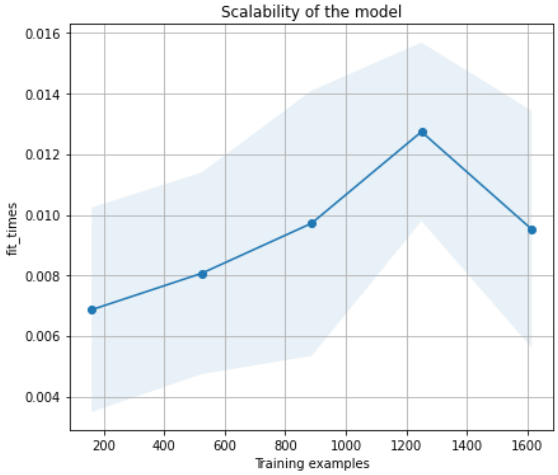

Performance of the model
score
fit_times

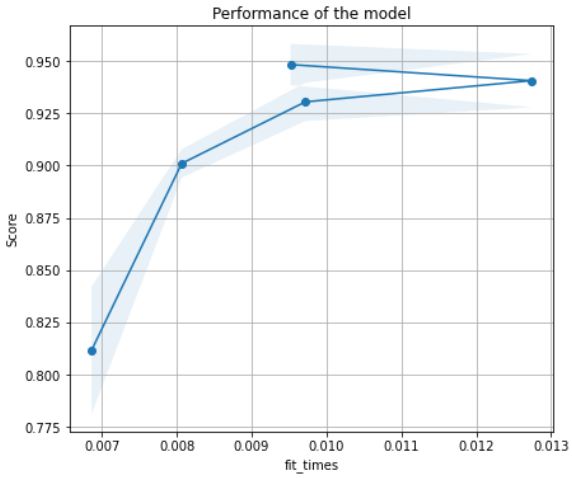

Learning Curves (Logistic Regression)
Score
Training examples
Training score
Cross-validation score

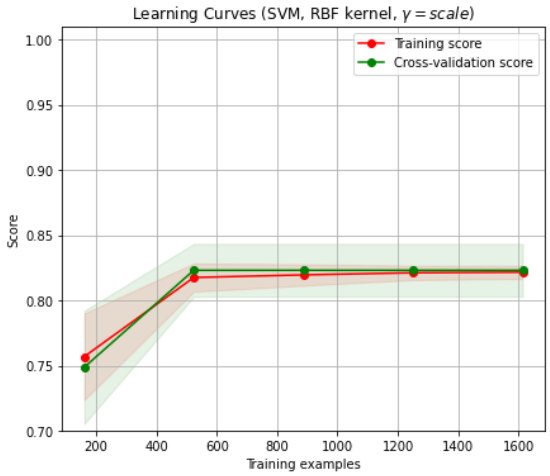

Scalability of the model
fit_times
Training examples

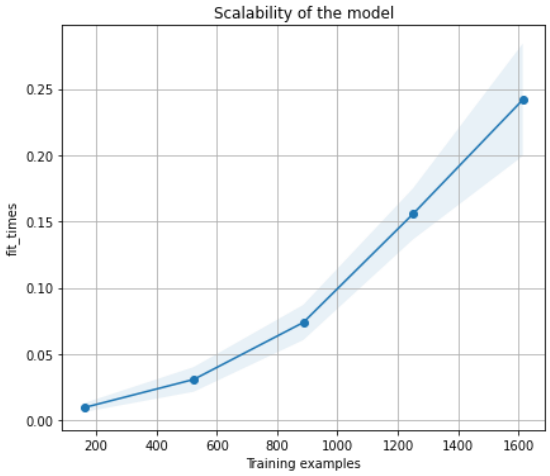

Performance of the model
Score
fit_times

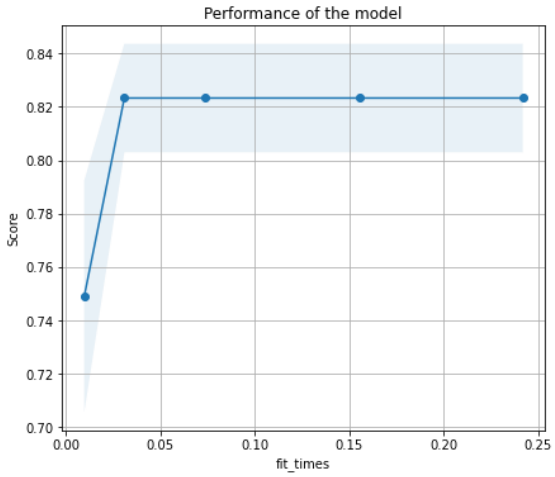

Learning Curves (KNN)
Training score
Cross-validation score
Score
Training examples
1.00
0.95
0.90
0.85
0.80
0.75
0.70
200
400
600
800
1000
1200
1400
1600

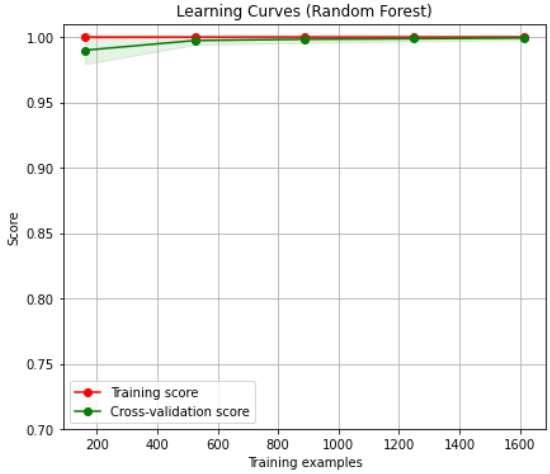

Scalability of the model
fit_times
Training examples
0.016
0.014
0.012
0.010
0.008
0.006
0.004
200
400
600
800
1000
1200
1400
1600

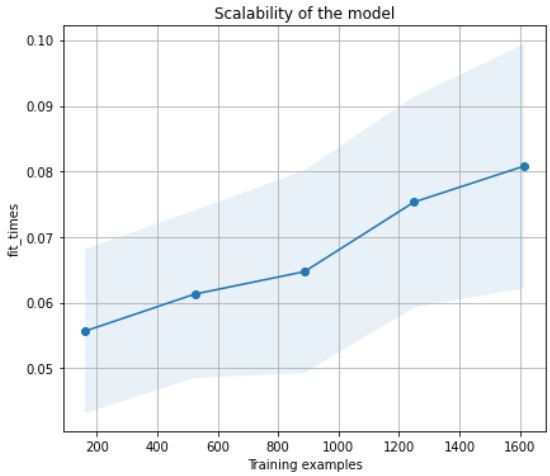

Performance of the model
Score
fit_times
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.007
0.008
0.009
0.010
0.011
0.012
0.013

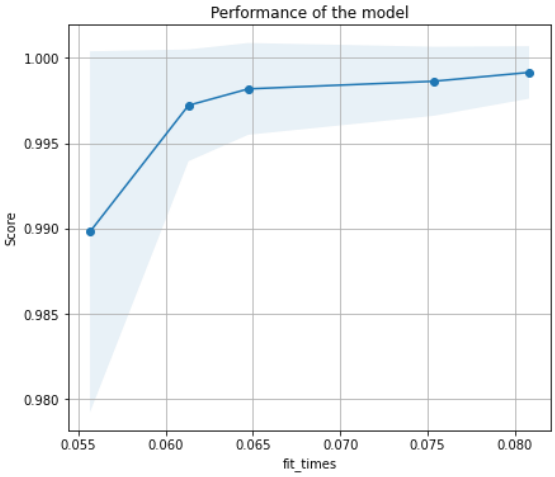

Learning Curves (SVM, RBF kernel, $\gamma = scale$)
Training score
Cross-validation score
Score
Training examples

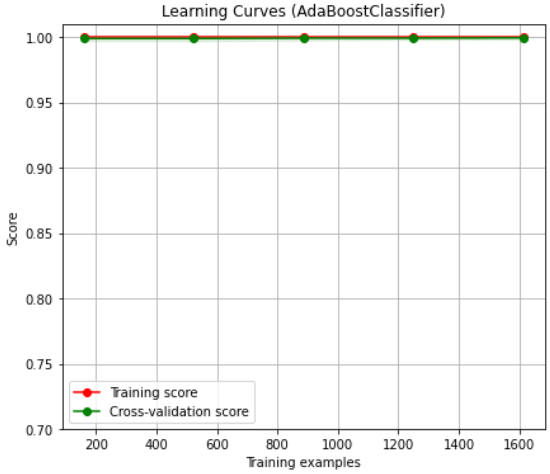

Scalability of the model
fit_times
Training examples

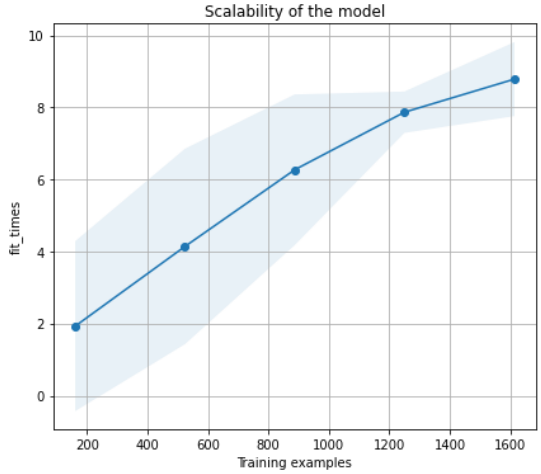

Performance of the model
Score
fit_times

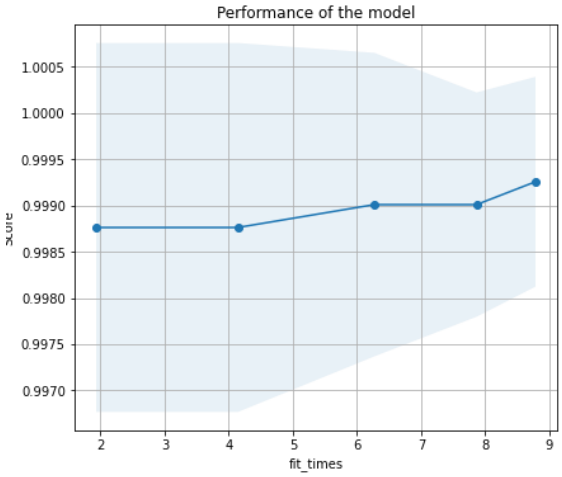

Learning Curves (Random Forest)
Score
Training examples
Training score
Cross-validation score

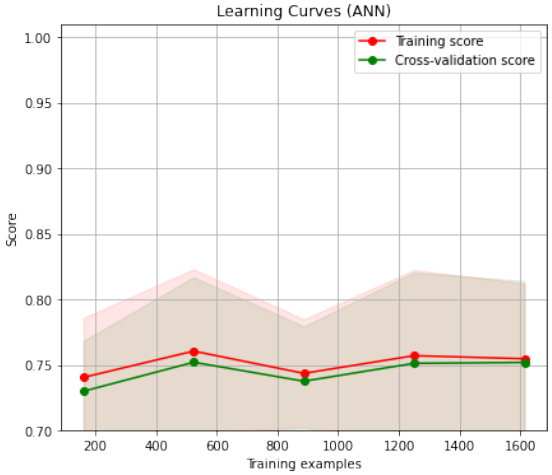

Scalability of the model
fit_times
Training examples

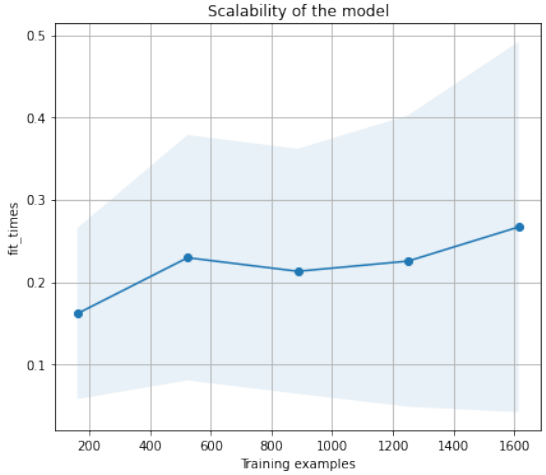

Performance of the model
Score
fit_times

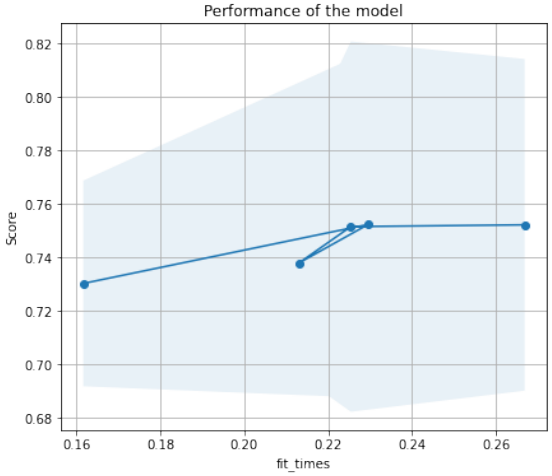

Learning Curves (AdaBoostClassifier)
Score
Training examples
Training score
Cross-validation score

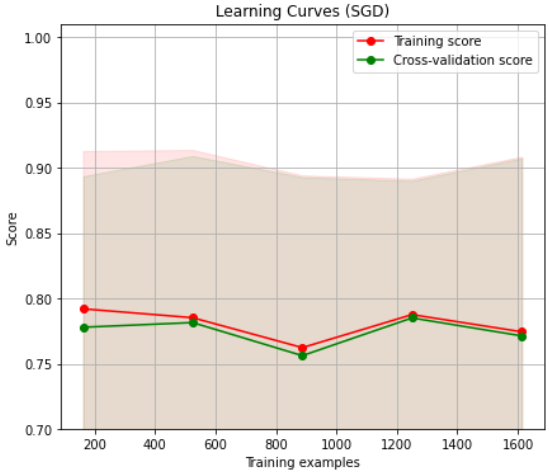

Scalability of the model
fit_times
Training examples

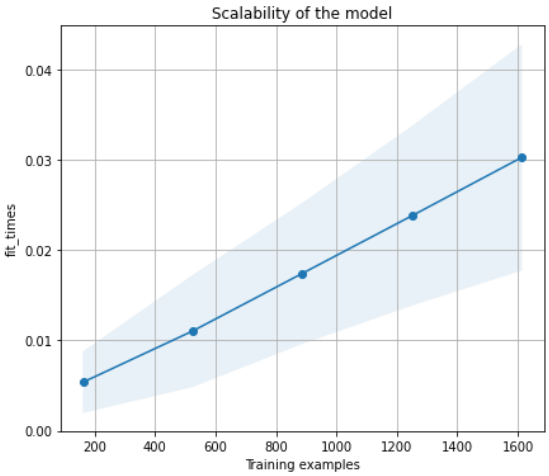

Performance of the model
score
fit_times

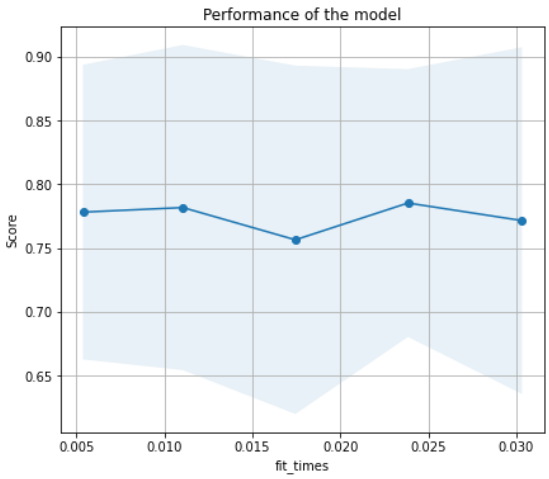

Learning Curves (ANN)
Training score
Cross-validation score
Score
Training examples
1.00
0.95
0.90
0.85
0.80
0.75
0.70
200
400
600
800
1000
1200
1400
1600

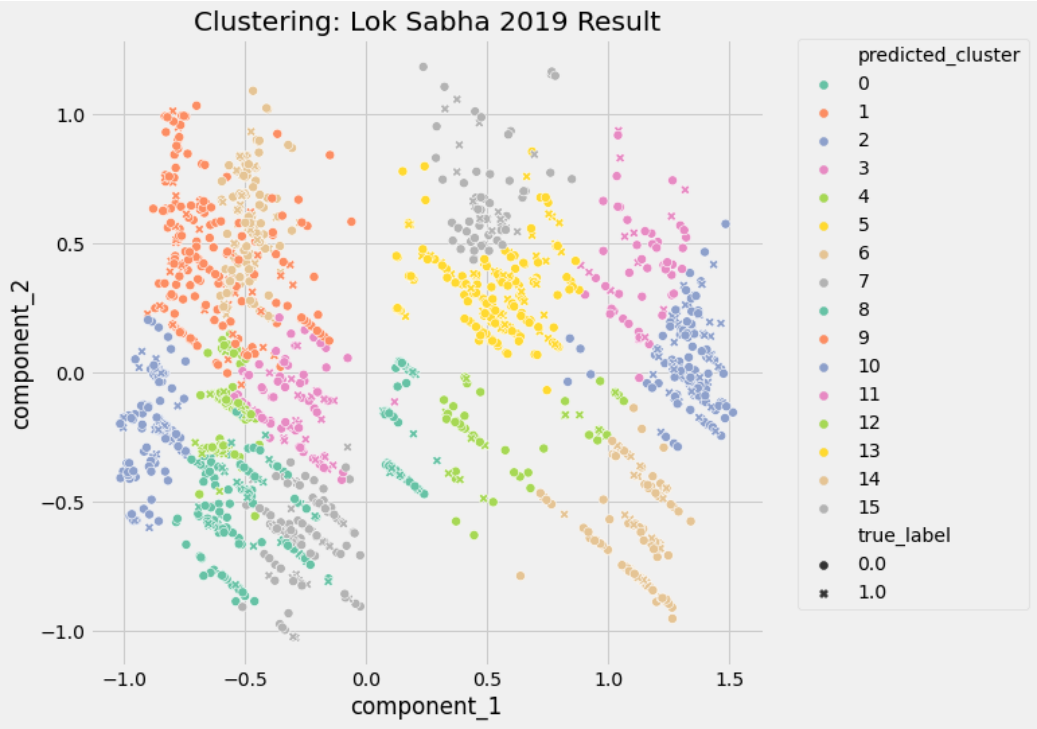

Scalability of the model
fit_times
Training examples
0.5
0.4
0.3
0.2
0.1
200
400
600
800
1000
1200
1400
1600

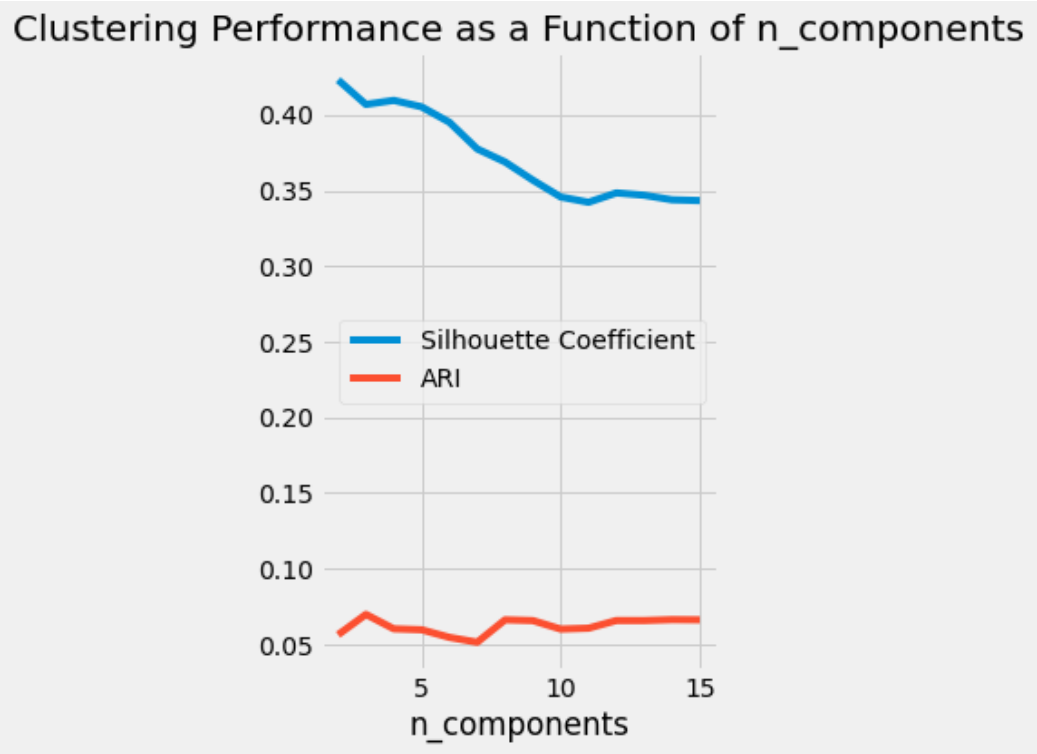

Performance of the model
Score
fit_times
0.82
0.80
0.78
0.76
0.74
0.72
0.70
0.68
0.16
0.18
0.20
0.22
0.24
0.26


Learning Curves (SGD)
Training score
Cross-validation score
Score
Training examples
1.00
0.95
0.90
0.85
0.80
0.75
0.70
200
400
600
800
1000
1200
1400
1600


Scalability of the model
fit_times
Training examples
0.04
0.03
0.02
0.01
0.00
200
400
600
800
1000
1200
1400
1600


Performance of the model
Score
fit_times
0.90
0.85
0.80
0.75
0.70
0.65
0.005
0.010
0.015
0.020
0.025
0.030


Clustering: Lok Sabha 2019 Result
component_2
component_1
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
predicted_cluster
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
true_label
0.0
1.0


Clustering Performance as a Function of n_components
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
Silhouette Coefficient
ARI
5
10
15
n_components