

FYS-STK4155
Applied data analysis and machine learning

Analysis of the presidential elections 2012 and 2016

By: Henry Haugsten Hansen and Fred Marcus John Silverberg

14 December 2018

Contents

1	Introduction	2
2	Theory	4
2.1	Data structure	4
2.2	Logistic regression	4
2.3	Neural network	6
2.4	Support Vector Machine	9
3	Code Implementation	12
4	Results	13
4.1	Logistic regression	13
4.2	Linear SVM	14
4.3	Kernel SVM	15
4.4	Neural network	16
4.5	Variable analysis	17
5	Discussion	19
6	Conclusion	20
7	Appendix	21

Abstract

The main purpose of the project was to successfully reproduce the election outcome of 2012 and 2016 in the US, as well as identify important variables. Data for analysing ranged from economics, demographics to ethnicity, among other variables. The data was first constructed into a design matrix and further standardized.

Various machine learning algorithms was deployed for completing the task, where accuracy score was the main metric. The Support Vector Machine with kernel [SVM] produced the best classification results of 89% for the 2012 election and 96% for the 2016. The method of Neural Network surprisingly performed almost as good with 87% for 2012 and 94% for 2016, while the logistic regression came out slightly worse.

The variables that contributed the most for correct classification were the age and population size of the counties. Regarding flipped counties between the elections, the SVM method hinted that the percentage of females and whites may have had a minor impact on the shift from one party to another.

1 Introduction

The idea of creating models that can predict output based on some input could be extremely valuable. Models like these could be as simple as regression to the complexity of deep neural networks. The simple models are easy to understand and easy to implement, but their ability to learn complicated relationships can be to somewhat limited. That's why neural networks have become so popular, because they can get a deeper understanding of the data. Models that are capable of predicting can help humans in decision making, reduce the number of experiments and observations needed and replace humans in doing boring and repetitive tasks which humans are normally not good at.

The model is first trained for all the data and model statistics are calculated. Then the model is validated and tested to assure the quality of the model. The validation technique used in this report to validate the different models is cross validation. Cross validation is widely used when the data set is small or finding the difference between the true value and the prediction for a single or a sample of observations. When the data set is small, the resulting model is more sensitive to an arbitrary splitting of the data into training data and test data. The split could be leaving out important data from the training part. Another common validation technique is bootstrap.

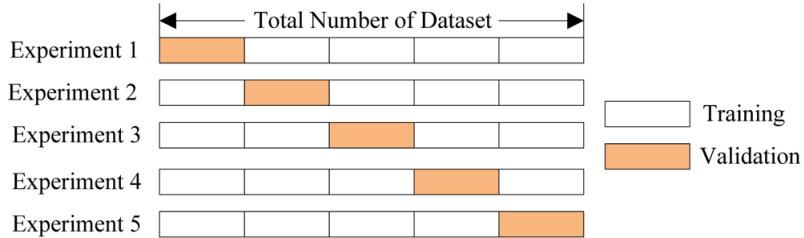


Figure 1: Cross validation explained graphically. The idea of cross validation. A small part of the data is used for training. The other larger part for testing/validation. Then another experiment and the test data changes and thus, so does the training data. In the end all data was test once but only once.

The bias-variance tradeoff is fundamental dilemma in statistical learning theory. Bias is erroneous assumptions in a model, so that the model misses out on important relationships. This is called underfitting. Variance is error from sensitivity to fluctuations in the training data. This is called overfitting.[3]

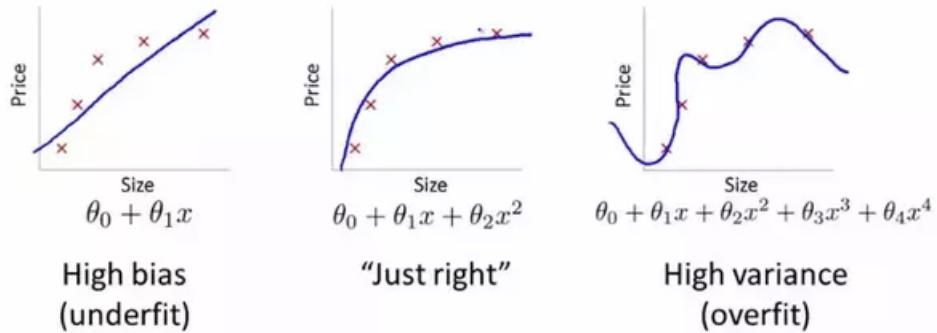


Figure 2: A simple visual example of underfitting and overfitting.

In classification, models are assessed based on the percentage of correct calculated labels.

$$Accuracy = \frac{\sum_{i=1}^n I(t_i = y_i)}{n} \quad (1)$$

2 Theory

2.1 Data structure

The presidential election data used in this project is downloaded from GeoDa[6]. The underlying information in the variables constructing our design matrix are all drawn from years between 2007 and 2014. They represent sociological aspects such as education, wealth, ethnicity, population density among others. As our target we choose to modify the percentage outcome for the democrats and republicans into ones and zeros respectively, allowing us to proceed with the project as a classification case.

The dependent variables together construct the design matrix. The design matrix has the dimensions (3108, 51), 3108 counties and 51 variables. There are two independent variables, namely the political outcome for each election, 2012 and 2016. The independent variables are each a binary vector with the dimensions (3108, 1). After a reshuffle of the rows, or counties, 75% of the rows are used for training and cross validating a model, and 25% are left for testing on unseen data on the 2012 election. Then the model is applied on the same design matrix without splitting and reshuffling to make predictions on the 2016 election.

Since our machine learning algorithms rely on euclidean distances they do not perform well when the data is scaled differently. In our case the data is both large and low in numerical measurements, with the addition of percentage variables. Therefore we transform the data by sklearn's pre-processing function 'StandardScaler'. [5]

2.2 Logistic regression

A popular binary classification method is the logistic regression, especially among statisticians. This is because it's a more statistical approach to a classification problem than support vector machines or decision trees. There's no analytic expression for the coefficients, so these must be calculated numerically. Why that is and how the mathematical problem is dealt with is briefly described below.

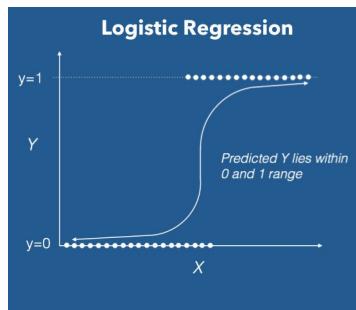


Figure 3: Logistic regression displayed graphically

The logistic function is called the logit or the sigmoid function (see below). It

returns a value or interpreted as a probability between 0 and 1. β is the wanted coefficients and x_i is an observation. The function tries to map the product of these two to either 0 or 1, but some input values will give an output value in-between. This is where the the function sharply rises. At this range of input values, the observations do not belong to a clear category.

$$p(\beta x_i) = \frac{1}{1 + \exp^{-\beta x_i}} \quad (2)$$

Moving on to deriving the cost function. The given probabilities could be either 1 or 0.

$$p(y_i = 1|x_i, \hat{\beta}) = \frac{1}{1 + \exp^{-\hat{\beta} x_i}} \quad (3)$$

$$p(y_i = 0|x_i, \hat{\beta}) = 1 - \frac{1}{1 + \exp^{-\hat{\beta} x_i}} \quad (4)$$

To obtain a cost function, define the total likelihood for all possible outcomes from a data set $D = (y_i, x_i)$, with the binary labels $y_i \in \{0, 1\}$ and where the data points are drawn independently, and then apply the Maximum Likelihood Estimation (MLE), i.e., maximizing the probability of seeing the observed data and approximating the likelihood in terms of the product of the individual probabilities of a specific outcome y_i .

$$P(D|\hat{\beta}) = \prod_{i=1}^n [p(y_i = 1|x_i, \hat{\beta})]^{y_i} [1 - p(y_i = 1|x_i, \hat{\beta})]^{1-y_i} \quad (5)$$

From the MLE, then the logistic regression has the following cost function.

$$C(\hat{\beta}) = \sum_{i=1}^n (y_i \cdot \log \cdot p(y_i = 1|x_i, \hat{\beta}))^{y_i} + (1 - y_i) \cdot \log \cdot [1 - p(y_i = 1|x_i, \hat{\beta})]^{1-y_i} \quad (6)$$

By defining a vector \hat{y} with n elements y_i , an $n \times p$ matrix \hat{X} which contains the x_i values and a vector \hat{p} of fitted probabilities $p(y_i|x_i, \hat{\beta})$, the first and second derivative of the cost function can be rewritten in a more compact way.

$$\frac{\partial C(\hat{\beta})}{\partial \hat{\beta}} = -\hat{X}^T (\hat{y} - \hat{p}) \quad (7)$$

$$\frac{\partial^2 C(\hat{\beta})}{\partial \hat{\beta}^2} = \hat{X}^T \hat{W} \hat{X} \quad (8)$$

\hat{W} is a diagonal matrix with the elements $p(y_i = 1|x_i, \hat{\beta})(1 - p(y_i = 1|x_i, \hat{\beta}))$

To find the minimum of the cost function, several approaches are possible. Since the second derivative can be found, one possibility is the multivariate version of

the Newton-Raphson's method. Newton-Raphson's method tries to find a root of a function $f(x)$.

$$x^{(k+1)} = x^{(k)} - J_f(x^{(k)})^{-1} \nabla f(x^{(k)}) \quad (9)$$

The problem can be redefined as finding where the gradient of the cost function is equal to zero, $\nabla f(x) = 0$. For logistic regression, Newton-Raphson's method will then look the following way.

$$\hat{\beta}^{new} = \hat{\beta}^{old} - \frac{\partial C(\hat{\beta})}{\partial \hat{\beta}}_{\hat{\beta}^{old}} \cdot \frac{\partial^2 C(\hat{\beta})}{\partial^2 \hat{\beta}}_{\hat{\beta}^{old}} \quad (10)$$

or

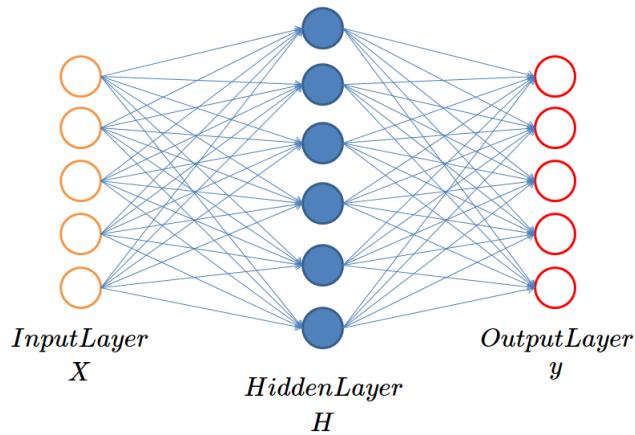
$$\hat{\beta}^{new} = \hat{\beta}^{old} - \hat{X}^T \hat{W} \hat{X} \cdot \hat{X}^T (\hat{y} - \hat{p}) \quad (11)$$

$\hat{\beta}^{old}$ must randomly initialized to start the iteration. \hat{p} is the output from $\text{sigmoid}(\hat{\beta}X)$. The iteration continues until the some criteria is reached, like the change in $\hat{\beta}$ is acceptable small.[2]

Computing the inverse of $\hat{X}^T \hat{W} \hat{X}$ could be computationally expensive. Therefore, in larger data sets a gradient descent method is used instead. In practice, it means the second derivative is replaced by a scalar, also called a learning rate. The function is convex, so Newton's method or any gradient descent method will always converge to the global minimum.[1]

2.3 Neural network

Figure 4: Logistic regression model



Source: <https://chunml.github.io/ChunML.github.io/project/Creating-Text-Generator-Using-Recurrent-Neural-Network/>

A neural network is constructed as shown above. Here the output layer contains more than one output, if wanted. And one hidden layer is added, constructing a net of weights (blue lines) and biases. Additional hidden layers can be included. Each hidden layer can be assigned arbitrary many neurons (the blue circles). The idea is sprung from the logistic regression and extended into a more complex configuration of interconnected elements that processes the information dynamically. Further break down of the concept is as follows, with the encapsulated iteration process in mind:[7]

First, we chose the cross entropy function as our cost function, defined in Eq[11].

Forward phase

Weights and biases are created randomly and normally distributed. The net input value (Z) is then calculated as:

$$Z = X \bullet W + b \quad (12)$$

Noted is that the calculations now take place as nested matrices, where the matrix W represent all weights on both sides of a calculated layer, hence have the size (2,).

Z is then activated by chosen activation function for the interaction between the input layer and the hidden layer, and similarly by the hidden and output layer.

$$a(Z) \quad (13)$$

Back-propagation phase

An error calculation is made by use of the cost function, which is denoted $[\delta_3]$. Next step is to update the weights and biases such that the error between the target and output decreases towards a global minima. This setp is the essence of a neural network and is often noted as the back-propagation algorithm, which evaluate the error and uses partial gradients to pass the necessary adjustments back into the system. For cross entropy as choice of cost function and Sigmoid as activation function, the back-propagation algorithm is implemented as below:[8]

$$\delta_3 = (y_p - y) \quad (14)$$

$$\delta_2 = \delta_3 \bullet W_2^T * (a(Z_1))' \quad (15)$$

$$\frac{\partial C}{\partial W_2} = a(Z)_2^T \bullet \delta_3 \quad (16)$$

$$\frac{\partial C}{\partial b_2} = \text{sum}(\delta_3) \quad (17)$$

$$\frac{\partial C}{\partial W_1} = X^T \bullet \delta_2 \quad (18)$$

$$\frac{\partial C}{\partial b_1} = \text{sum}(\delta_2) \quad (19)$$

Layers are representative by $(1,2,3) = (I, H_1, O)$ such that the recipe can be extended if additional hidden layers are added, as $(1,2,3,4) = (I, H_1, H_2, O)$. Note

that if the cost function or the activation function are changed, one would have to derive δ_3 from scratch.

There must be highlighted that the neural network are prone to some parameters that governs entirely how well the model will preform.[8]

Learning rate

A hyper-parameter that controls how much the weights and biases will adjust due to the gradient of the cost function. A small value could be described as taking small but accurate steps towards the minimum. For a large value, the path towards the minimum would be rough. To large and there is a probability of never reaching the precise minima, while to low may be stuck at a plateau. It also affects the computational time, a large learning rate would decrease the computational time and vice versa.

Regularization term

By applying a regularization term, the weights and biases (or coefficients for the linear case) is basically programmed to be small, with the assumption that a lower variance between them would represent a simpler model, hence it is a way of avoiding over-fitting.

Epochs

Denotes how many times a training set is run through the neural network, both forward and backward. By increasing the number of epochs one would expect the model to obtain optimal weights and biases, only running it once would under-fit the model.

Batch size

Included in the SGD method, the training data is divided into smaller batches which run separately through the neural network. So instead of running the whole training set in one epoch, all batches runs once for one epoch. This allows us to run larger data sets. Optimal batch size is discussed but the size will effect the convergence of the gradient. Generally a small batch size in the range of 32-512 is to prefer.

Layers and Neurons

Increasing the number of neurons or layers also increase the networks ability to learn detailed parts of the data, so for a complex situation more neurons and / or layers would be necessary. This can be adjusted along the way, by doing a grid-search for different numbers and compare the final score. The configuration also have impact on the proper choice of activation function.

Activation function

The choice of activation function will have impact on how the model preform as it transform the values into different domains. They may have steeper gradients and lower computational time, or be able to avoid vanishing gradients allowing deeper nets, or decrease the probability of exploding cost functions. For a network with multiple layers, one could use different functions for each layer.

The above theory is nicely implemented in the package Tensorflow. And con-

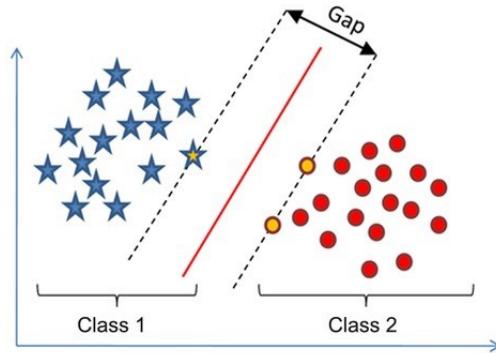
structuring a Neural Network with it can be described with a few lines:

```
# Set up the Neural Network
clf = tensorflow.keras.Sequential()
# Adding two hidden layers
clf.add(tensorflow.keras.layers.Dense(...))
clf.add(tensorflow.keras.layers.Dense(...))
# Compile model
sgd = tensorflow.keras.optimizers.SGD(...)
clf.compile(optimizer=..loss=..metrics=...)
# Fit the model
clf.fit(X,y,epochs=..batch_size..)
```

2.4 Support Vector Machine

A machine learning method that is broadly used in classification problems, noted is that it could as well be used for regression purpose. It relies on supervised data to construct a decision boundary that divides classes in space. The advantage of the algorithm lies partly in the support vectors, marked as dotted lines in figure[4]. The optimal decision boundary (red line) is obtained by maximizing the gap between the support vectors. This feature is highly appreciable when dealing with classes that are difficult to separate. And further advantage is gained by the allowance of analysing data in higher dimensions.[4]

Figure 5: Visualized SVM concept



The decision boundary and support vectors are here described by:

$$y = w^T x + b = 0 \quad , \quad y = w^T x + b = \pm 1 \quad (20)$$

Where $[w]$ is the normal vector to the decision boundary, $[b]$ is the bias. The maximal margin, or gap in figure[4] is defined as:

$$\max_w \frac{2}{\|w\|} \quad \text{or equivalently} \quad \min_w \|w\|^2 \quad (21)$$

Cost function:

Mentioned margin is to be optimized. But first we introduce a slack variable

$[\xi]$, which for $[\xi \geq \|w\|]$ a point is misclassified and for $[0 < \xi \leq \|w\|]$ a point is located within the margin. This is added as a term which operate as a margin softener, simply allowing some mistakes, we have:

$$\min_w \|w\|^2 + C \sum_{i=1}^m \xi_i \quad \text{constrained to} \quad y_i * f(x_i) \geq 1 - \xi_i \quad (22)$$

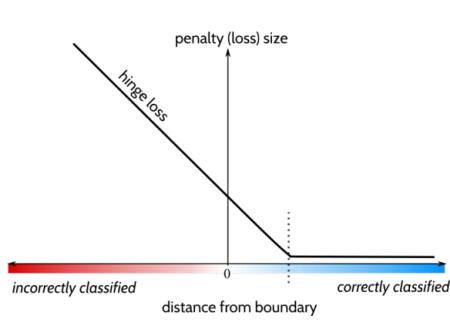
Here C is the regularization parameter that regulates how much we value the mistakes and $[x_i]$ is the support vectors. By including $[\xi_i \geq 0]$ in the constraint, we can write the constraint and optimization function respectively as:

$$[y_i * f(x_i) \geq 1 - \xi_i] \rightarrow [\xi_i = \max(0, 1 - y_i * f(x_i))] \quad (23)$$

$$\min_w \|w\|^2 + C \sum_{i=1}^m \max(0, 1 - y_i * f(x_i)) \quad (24)$$

Where $[f(x_i) = w^T x_i + b]$. The summation term is denoted the hinge loss function, illustrated in figure[5].

Figure 6: The hinge loss function



Since this a non smooth function, it is needed to interpret it as a piece-wise function in order to minimize it. We can denote the term inside the summation as: $[\max(0, 1 - y_i * f(x_i)) = L(x_i, y_i, w)]$, by observing the two possibilities:

$$\frac{\partial(L)}{\partial(w)} = -y_i x_i \quad , \quad \frac{\partial(L)}{\partial(w)} = 0 \quad (25)$$

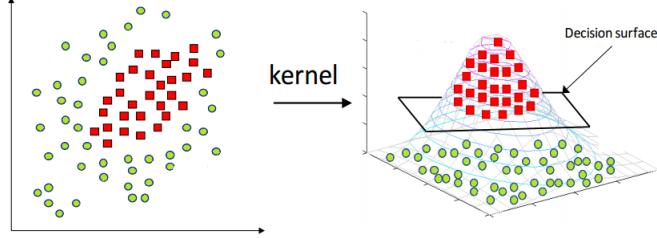
Further, we take use of the Lagrange method and define $\lambda = 2/(mC)$. We can now write the cost function to minimize as:

$$Cost(w) = \frac{1}{m} \sum_{i=1}^m \left(\frac{\lambda}{2} \|w\|^2 + L(x_i, y_i, w) \right) \quad (26)$$

Kernel trick

By transforming the above primal form into dual form, the advantage of using the kernel-trick become available. The trick essentially transform a non-separable space into a separable, allowing us to obtain a decision boundary.

Figure 7: Kernel trick visualized



This is done by observing that the solution $[w]$ can be expressed as a linear combination of the data:

$$w = \sum_{k=1}^m \lambda_k y_k x_k \quad (27)$$

This is then substituted into $f(x_i)$ and the defined cost function, leading to a optimization problem over λ instead of w , with the following decision function:

$$f(x) = \sum_{i=1}^m \lambda_i y_i K(x_i, x) + b \quad (28)$$

And the optimization function follows as:

$$\max_{(\lambda_i \geq 0)} \lambda_i - \frac{1}{2} \sum_{kj} \lambda_k \lambda_j y_k y_j K(x_k, x_j) \text{ for } [0 \geq \lambda \geq C] \text{ and } \sum_{i=1}^m \lambda_i y_i = 0 \quad (29)$$

Finally the kernel trick can be applied by inserting chosen kernel function suitable for the specific classification problem. A few commonly functions are:

Linear

$$K(x_i, x) = x^T x_i \quad (30)$$

Polynomial

$$K(x_i, x) = (x^T x_i + \gamma)^d \quad (31)$$

Radial basis

$$K(x_i, x) = e^{-\gamma ||x - x_i||^2} \quad (32)$$

Parameters:

For being able to obtain the most efficient model of data, one need to tune some parameters. There is no golden rule here, the tuning must be done by consideration for each applied situation. This is usually handeled by doing a cross validation search. In addition to the parameters described below the different kernel functions also govern the final results, hence experimanting with those is also recommended.[5]

C

A parameter that governs the sensitivity of correct classifications. It is essentially a trade-off between the amount of misclassification on training data and the maximization of the margin between the support vectors. For larger values

of C only smaller margins would be allowed and for smaller values the margins would be larger and hence the decision boundary would be simpler, at the expense of accuracy. For $C = \infty$ the margin is denoted hard and do now allow any misclassification's. C is simply the regularization factor for SVM.

γ

This parameter is part of the kernel function and vary the dependence on whether the algorithm will construct a decision boundary with high consideration of points in its neighbourhood, or not. A low value would loosely fit the data, hence under-fitting it with a less complex boundary, and vice versa for a high gamma value.

The above theory is easily implemented by a two liner in Scikit learn:

```
# Set up Support Vector Machine
clf = SVC(C=...,kernel=...,gamma=...)
# Fit the model
clf.fit(X,y)
```

3 Code Implementation

The python programming language is used for implementation of the methods. An the Python packages Numpy, Scikit learn and Tensorflow are used extensively because of their rich machine learning functionality. In terms of computational time, the Tensorflow package is several order of magnitude faster compared to our own codes developed by the theory described above, we will therefore only focus on using this package when running the relevant calculations.

Overall code structure:

```
Runnable classes:
* SVM and Logistic regression.py
    Used for calculating SVM and regression results.
* Binary neural network.py
    Used for calculating NN results.

Runnable files:
* variable_analysis.py
    Used for analysing the variables.

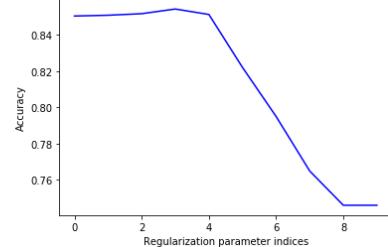
Imported Files:
* methods.py
    Holds various functions needed for the project.
* data.py
    Loads the dataset properly.
* Flipped counties.py
    Only used for drawing a map of flipped counties.
* k_fold_CV.py
    Used for cross validation.
* to_binary.py
    Used for changing the target and output into binary.
```

4 Results

4.1 Logistic regression

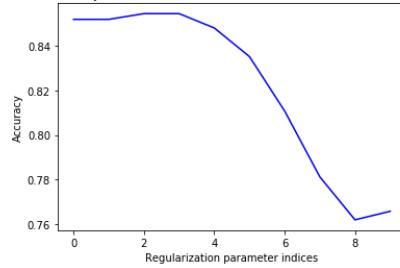
The logistic regression can up to a certain regularization value classify a satisfactory amount of counties. The calculation are made with the built in logistic regression from Scikit learn.

Logistic regression: Accuracy train as function of different lambdas indices. Training data. 2012 election.



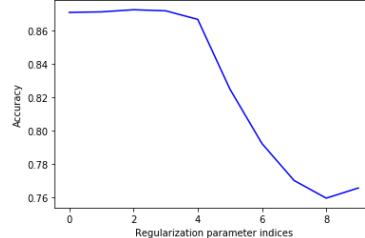
(a) Train data 2012

Logistic regression: Accuracy test as function of different lambdas indices. Test data. 2012 election.



(b) Test data 2012

Logistic regression: Accuracy new data as function of different lambdas indices. Testing on the 2016 election.



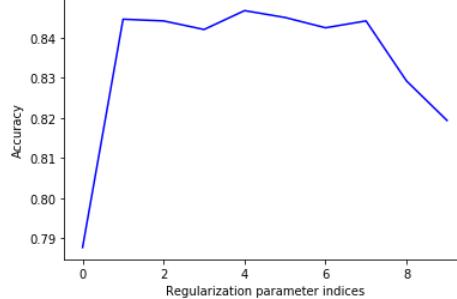
(c) Test data 2016

Figure 8: Accuracy graph

4.2 Linear SVM

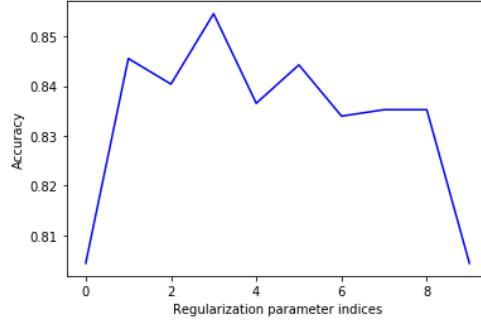
The scikit learn is also used to produce the SVM results below. The C's applied are values in the interval from 0.001 to 30. The accuracy stabilizes and peaks at C=3, index 1. The results are similar to logistic regression. This is not surprising, since they are both linear classifiers.

Linear SVM: Accuracy as a function of different C indices. Training data. 2012 election.



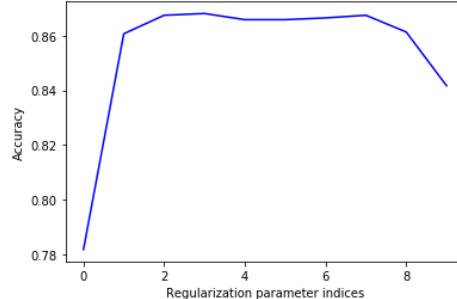
(a) Train data 2012

Linear SVM: Accuracy as a function of different C indices. Test data 2012. election



(b) Test data 2012

Linear SVM: Accuracy as a function of different C indices. Testing on the 2016 election.



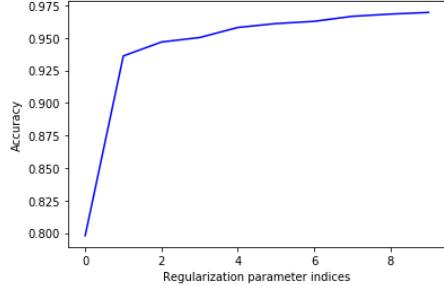
(c) Test data 2016

Figure 9: Accuracy graph

4.3 Kernel SVM

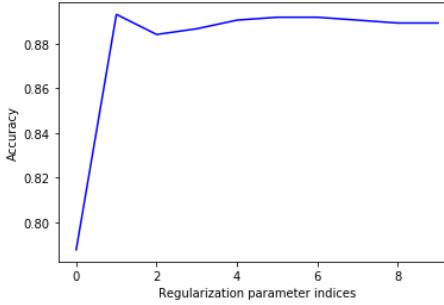
The C's values applied are the same as in linear SVM. The accuracy stabilizes and peaks at C=3, index 1. These results suggest that higher accuracy can be achieved by using the kernel trick.

Kernel SVM: Accuracy train as a function of different C indices. Training data. 2012 election.



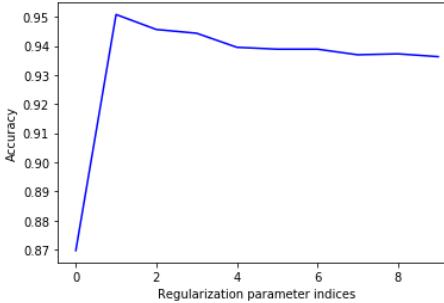
(a) Train data 2012

Kernel SVM: Accuracy test as a function of different C indices. Test data. 2012 election.



(b) Test data 2012

Kernel SVM: Accuracy as a function of different C indices. Testing on the 2016 election.



(c) Test data 2016

Figure 10: Accuracy graph

4.4 Neural network

We found that optimal results were given by a layer set-up of [H,H,H,H,O] Were the activation functions for the hidden layers [H] were the Relu function, while the output layer [O] had sigmoid as activation function. The hidden layers were assigned 1000 neurons each. The optimization technique used is the stochastic gradient descent, SGD, with a batch size of 32 and 10 epochs.

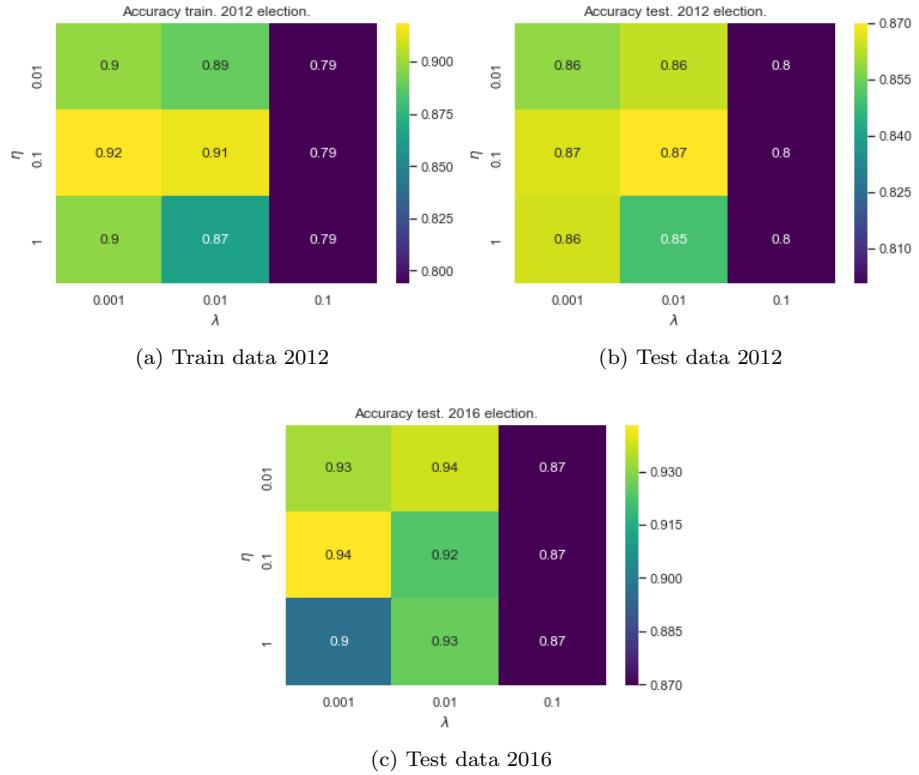


Figure 11: Accuracy heatmap

4.5 Variable analysis

Below we present details regarding the variables impact on the results and on the flipped states between 2012 and 2016. For translation of the variables, see footnote.¹

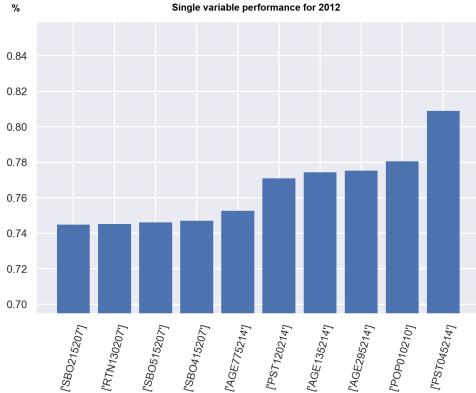


Figure 12: Most explanatory variables with accuracy score

Graphics from test data against 2012. Calculated with SVM for gamma=0.00778, c=2 and plain guess would be 78%. Highlights the importance of the population and age variables as well as the one for firms. Further data can be found in appendix [2]

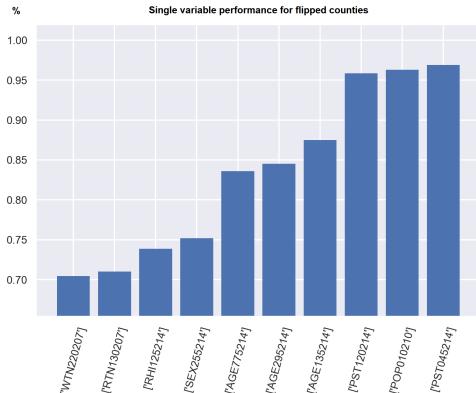


Figure 13: Most explanatory variables for flipped states with accuracy score

Graphics from test data against 2016. Calculated with SVM for gamma=0.00334, c=2 and plain guess would be 95%.

¹https://geodacenter.github.io/data-and-lab//county_election_2012_2016-variables/

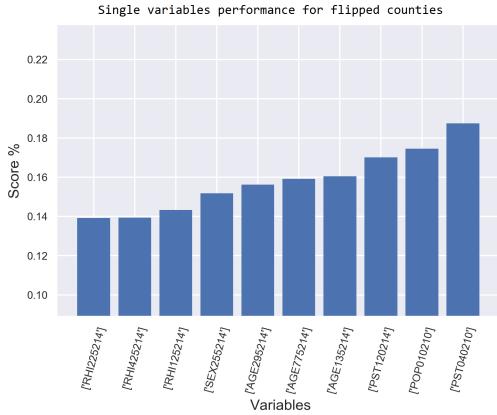


Figure 14: Most explanatory variables for flipped states with F1 score

Graphics from test data against 2016. Calculated with SVM for gamma=0.00778, c=2 and plain guess would be 95%. For the flipped case age and population still have the most impact, but small signs of female percentage and white percentage are showing up as well.

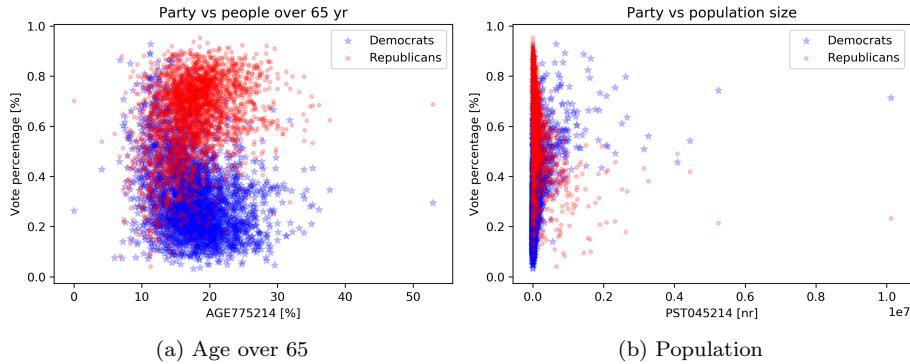
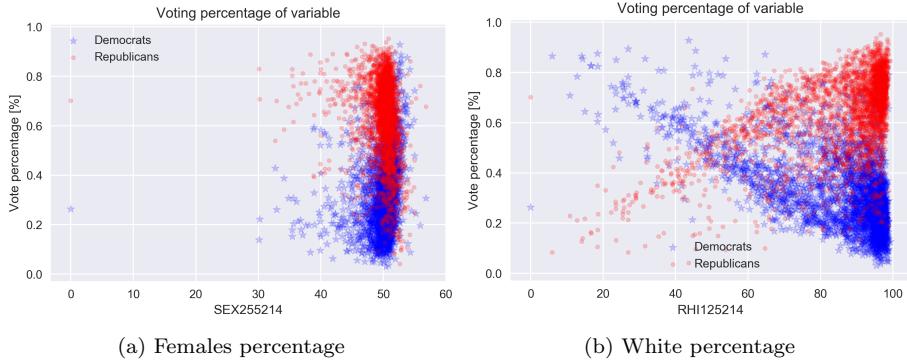


Figure 15: Scatterplot of important variables

Example that highlight the difference in political stand for the outstanding variables age and population. One can see that for a county with 30% percent old people, democrats are few. And with growing population, there is a decline in republicans.



(a) Females percentage (b) White percentage

Figure 16: Scatterplot of important variables for flipped states

For the flipped counties the variables of white people percentage and female percentage stood out as slightly more important than the other variables, not considering the age and population. It is very clear how different these two groups vote.

5 Discussion

The methods of logistic regression and linear SVM produces similar results, the logistic regression seems to be more sensitive to the regularization parameter C . This could most likely be explained by the difference in loss functions, where the logistic 'cross entropy function' diverges faster than the 'hinge loss function' used for linear SVM.

A quick look at the Kernel SVM results imply that the transformation of data by the kernel trick, yields substantially better results. Especially the training results are better, which compared to the test results imply that the model is over-fitting to a larger extent. The trade-off in accuracy for test data is regardless favourable in this case. As for the neural network performance, we did not expect the results to be greater than the SVM, since our data set is not the largest. However, the method produced satisfying results in line with kernel SVM. The drawback in comparison is the computational time. We calculated our results with the libraries Tensorflow and Sklearn. What SVM was able to produce in seconds, NN produced in several minutes, performing the same computations without the libraries would most likely scale this differences even further.

An interesting aspect of the results is the difference between the 2012 and 2016 election. Here one would expect 2012 training data to fit 2012 test data better, which is not the case. An explanation could be the accuracy paradox, where the target is even more unbalanced for the 2016 election, making the overall classification prone to higher accuracy score.

Based on the variable performance we can observe that all variables included in this dataset contributes with valuable information. A variable selection test

was made, and the result confirmed the statement above as it yielded worse results for selected variables, see appendix [1]. A quick eye on the variables in figure [16], imply that the size and age of the population matters the most. This corresponds well with the reality that the most dense cities generally coincide with the democrats, as can be seen in figure [19].

As for the flipped case with 257 counties, we preferred the SVM algorithm since it is better suited for small data sets compared to Neural Networks. The set for the flipped case was also heavily unbalanced, as the republicans had 95% of the flips. The combination of a small data set and unbalanced data was handled to some extent by cross validation and the balance function included in Sklearn. However, the accuracy score varied with the percentage of democrats included in the test data, so the results in figure[17] should not be interpreted with high confidence. We tried another metric approach, where we used the F1 score metric. The results yielded were very bad, as can be seen in figure[18]. This imply that the SVM method could not classify this small data set satisfactory enough.

6 Conclusion

The use of machine learning for reproducing the election outcome gave valuable results that predicts the outcome with high certainty. All methods produced test results in the range of 85% to 89% for the 2012 election and in the range of 86% to 95% for the 2016 election. These results should be incorporated with the unbalanced target percentages of (79/21)% and (87/13)%, respectively. We can conclude that the SVM kernel method were slightly better suited than the Neural Network, and is an obvious choice if computational time were to be considered. The addition of the kernel trick raised the results significantly when considering the unbalances of the data. It is also apparent that the Neural Network is more sensitive for tuning parameters, as the results varies more than SVM.

Population size and age of the inhabitants are the two most important features for the classification of a democrat or republican. As for the flipped case we had the desire to sort out a fancy silver bullet that could explain the party shift in general. However, the disappointed results were to weak for any robust conclusion, only vague implications that the percentage of white people and the percentage of females may have had some extra impact.

We can also conclude that the accuracy score metric used throughout this project, may not be the most suitable for this dataset. The testing of the F1 score metric gave clear indications that further analysis should explore different metrics to tackle this dataset.

7 Appendix

[1] Accuracy score for selection of variables:

Table 1: Accuracy score for selection [SVM:Gamma=Auto]

Original variables	Selected variables	C value
0.87	0.85	1
0.88	0.85	3
0.88	0.85	5
0.88	0.84	8

This was produced by the function "ExtraTreesClassifier" by sklearn.

[2] Extended results for variable analysis:

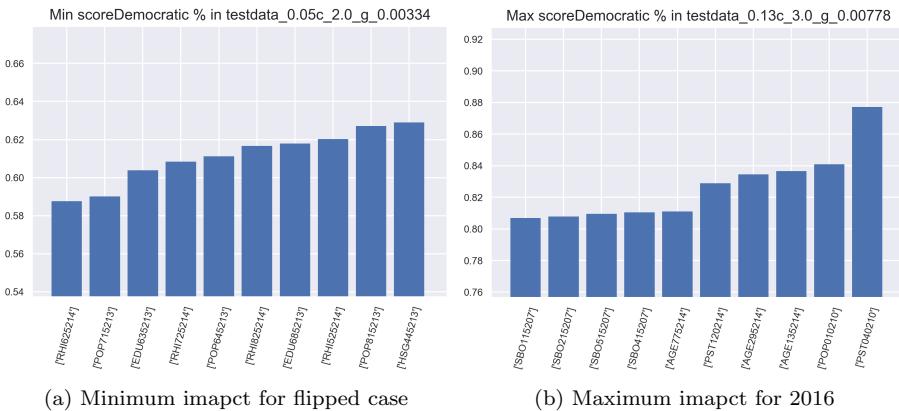


Figure 17: Single variables

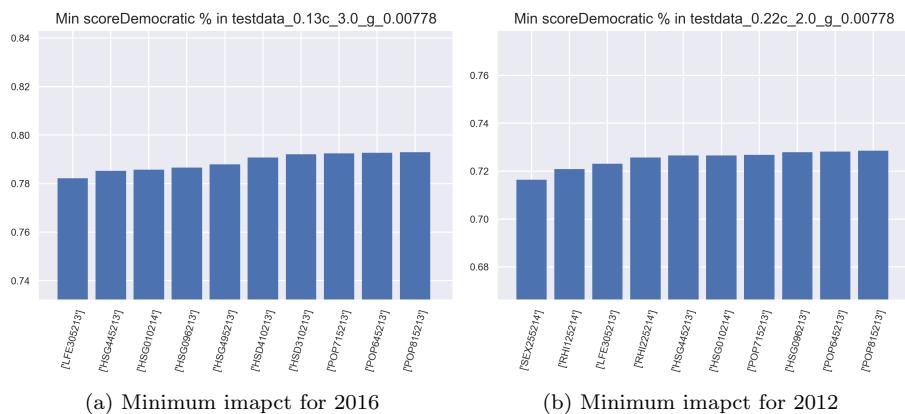


Figure 18: Single variables

[3] List of variables indices:

PST = Population
POP = Population
AGE = Age data
SEX = Gender data
RHI = Ethnicity data
EDU = Education data
VET = Veterans data
LFE = Travel time for work
HSG = Housing data
INC = Income data
PVY = Poverty data
BZA = Private non-farm employment
NES = Non-employer establishments
SBO = Firms owned by different ethnics
MAN = Manufactures shipping
WTN = Merchant wholesaler sales
RTN = Retail sales
AFN = Accommodation and food services sales
BSP = Building permits
LND = Land area

All variables can in full be found at:

https://geodacenter.github.io/data-and-lab//county_election_2012_2016-variables/

[4] List of previous research:

<http://www.ryan-peach.com/school-projects/2017/5/22/describing-the-2016-election-with-machine-learning>

By: Ryan Peach

[5] Map of miss classifications:

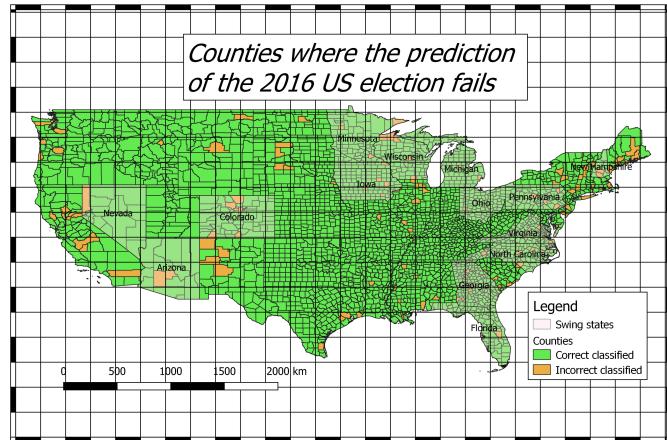


Figure 19: Counties that the ??? could not classify

[6] Map of flipped counties:

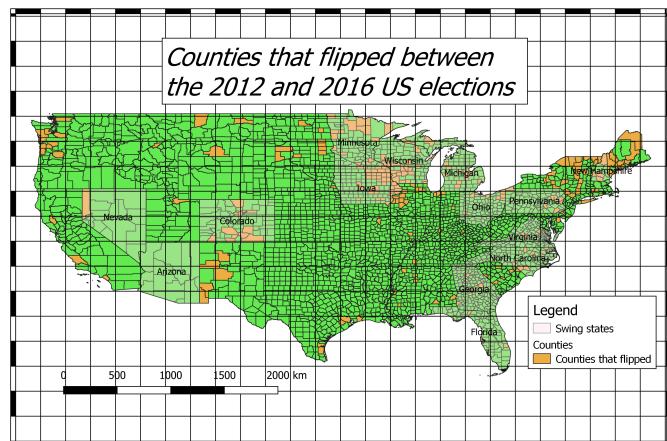


Figure 20: Map of flipped counties between 2012 and 2016

References

- [1] A. C. Faul. A concise Introduction to Numerical Analysis. Newton's method.
Page 116-119.

- [2] Morten Hjorth-Jensen. 2018. Data Analysis and Machine Learning: Logistic Regression. [ONLINE] Available at: <https://compphysics.github.io/MachineLearning/doc/pub/LogReg/html/LogReg.html>. [Accessed December 2. 2018].
- [3] An Introduction to Statistical Learning with Applications in R, (Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani). Page 33.
- [4] Ritchie Ng. 2018. Support Vector Machines [ONLINE] Available at: <https://www.ritchieng.com/machine-learning-svms-support-vector-machines/> [Accessed 14 December 2018].
- [5] Scikit – learn. 2018. RBF SVM parameters. [ONLINE] Available at: https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html [Accessed 14 December 2018].
- [6] GeoDa. 2017. 2012 and 2016 Presidential Election. [ONLINE] Available at: <https://s3.amazonaws.com/geoda/data/election.zip>. [Accessed 15 December 2018].
- [7] Morten Hjorth-Jensen. 2018. Data Analysis and Machine Learning: Neural Network. [ONLINE] Available at: <https://compphysics.github.io/MachineLearning/doc/pub/NeuralNet/pdf/NeuralNet-minted.pdf>. [Accessed December 2. 2018].
- [8] Michael Nielsen. 2018. Neural Networks and Deep Learning. [ONLINE] Available at: <http://neuralnetworksanddeeplearning.com/chap3.html>.

Access to all material can be found at: <https://github.com/silverberg89/FYS-STK4155/tree/master/Project3>