Our goal is:

- To make the most accurate prediction of the likelihood of winning a US presidential election based on the words used in speeches;
- To find some of the words that might be associated with winning speeches (and maybe to see if they collectively suggest any strategies for good speeches);
- To see if deceptive language has any role in winning US presidential elections.

My works are mainly answering the first question to make the most accurate prediction. In 4.4, I'm answering which single words and which group of words are associated with winning speeches' prediction by feature selection method and discussing different results with different techniques.

Also, in 1-6, I'm focusing on mostfreq1000docword.csv. Then, I will add deceptiondocword.csv to the model in part 7 to see if deceptive languages have any role in improving the optimal model.

All important code for python is in the section 7.5, the outputs are shown in each session.

1. Data overview

1.1 data description

To make the most accurate prediction of the likelihood of winning a US presidential election based on the words used in speeches, I am using mostfreq1000docword.csv, winners.csv, speeches.csv, mostfreq1000word.csv and combine them into a new dataset.

	the_AT	and_CC	to_TO	of_IN	a_AT
1992clinton920416.txt	0.047838	0.037313	0.021814	0.020475	0.020283
1992clinton920422.txt	0.040797	0.031784	0.023482	0.016603	0.022770
1992clinton920716.txt	0.037022	0.033041	0.016919	0.018113	0.019904
1992clinton920814.txt	0.053289	0.037926	0.018243	0.024964	0.015362
1992clinton921002.txt	0.051535	0.029605	0.012061	0.026316	0.025219

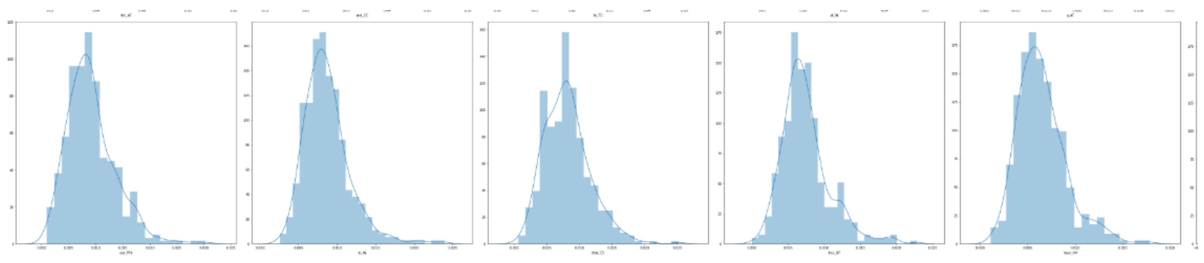
The shape of the word is (431,1000). The row name is who and when for each speech. (year + name + date + 'txt'), column name is name of the words in Stanford tags (word + '_' + type of the word). Each entry is the rate at which a word was used in a speech. The target is from winners.csv, describe who win the speech. It gives me an idea to group columns by type of words.

1.2 Statistics

I am using basic statistics and correlation. Since it has 1000 columns, so see the target first. For the target value, it has following properties:

	0
count	431.000000
mean	0.670534
std	0.470566
min	0.000000
25%	0.000000
50%	1.000000
75%	1.000000
max	1.000000

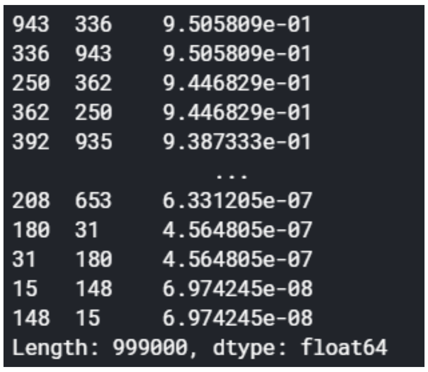
The target value is a group of 0s and 1s. It has mean 0.67, which means the distribution of target column is about 67% '1's, and 33% '0's, about 67% of speeches won. It also has a standard deviation 0.47, high standard deviation means values spread out over a wide range.



(Full graphs see **7.2**)

By the histogram of variables (other plots are in appendix), some variables are negatively skewed. So, from this investigation, a standard scaler helps us.

Pairs of correlation:



```
943  336    9.505809e-01
336  943    9.505809e-01
250  362    9.446829e-01
362  250    9.446829e-01
392  935    9.387333e-01
                ...
208  653    6.331205e-07
180  31     4.564805e-07
31   180    4.564805e-07
15   148    6.974245e-08
148  15     6.974245e-08
Length: 999000, dtype: float64
```

Here is the data for pairs of correlation, with descending sequence and ignore diagonal matrix of the correlation matrix. For correlation, I find the pairs with high correlation and remove one of them (>95%) to avoid multicollinearity. We need to handle multicollinearity because it let the assumption of independent variables to dependent. Multicollinearity also reduces the precision of the estimated coefficient, weaken the statistical power of our model. By python code, in the 943rd and 336th column, the correlation is higher than 95%. I choose to remove the 943rd column.

1.2.1 Discussion with PCA

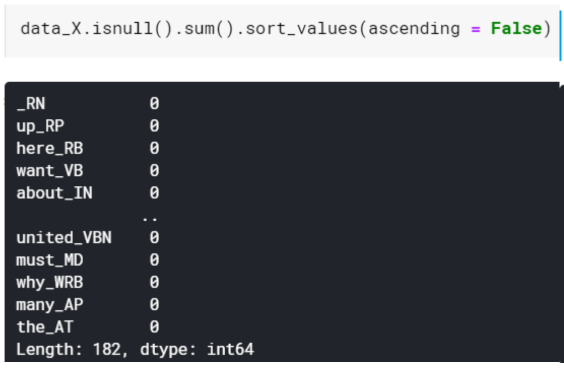
Also, I'm finding if most pairs of variables have a high correlation. High correlation means the relationship is strong between variables. There are 20.4% pairs of variables has correlation above 0.2, also 50.2% pairs of data have correlation above 0.1. To avoid 'irrelevant' 1, we can choose to remove features that is not correlated with target. But most of the features might has some correlation with target, and for non-linear classifiers, remove features is not good since the features might have correlation with other features. A dimensional reduction method might help with this condition. So, it seems like Principle component analysis might help our prediction. PCA is useful to dimension reduction when data is highly correlated with each other by removes correlated features. In this dataset, the correlation is not low, so we can consider use it in some models. By getting rid of correlated variables, PCA speeds up my machine learning algorithms, reduces overfitting when there are too many variables, and improves visualization in clustering techniques. But the trade-off is information loss.

2. Data issues

2.1 missing columns

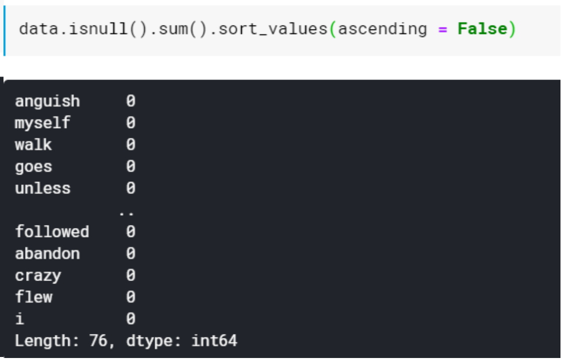
The dataset might have wrong data, missing values. If we directly open the csv file in the first dataset, it only has 182 column names in mostfreq1000word.csv, but it has 1000 columns in mostfreq1000docword.csv. I tried to change the csv file to the txt file, add a double quote to close the quotation before wrong data column names. Wrong data columns are such as '_RN,' and ' -_RBR'. After that, there is still an error when I import the csv file in python for the encoding method 'utf-8'. Then I change the encoding method to 'cp1252', a single-byte character encoding of the Latin alphabet. After that, I got 1000 column names with no error.

2.2 missing values



```
data_X.isnull().sum().sort_values(ascending = False)
```

```
_RN           0
up_RP         0
here_RB       0
want_VB       0
about_IN      0
             ..
united_VBN    0
must_MD       0
why_WRB       0
many_AP       0
the_AT        0
Length: 182, dtype: int64
```



```
data.isnull().sum().sort_values(ascending = False)
```

```
anguish     0
myself      0
walk        0
goes        0
unless      0
           ..
followed    0
abandon     0
crazy       0
flew        0
i           0
Length: 76, dtype: int64
```

By python isnull() function, these datasets do not have null entries.

2.3 normalization, binning, transformation

In the dataset, since the entries are percentages between 0 and 1, there is no need to normalize. A standard scaler arranges the data in a standard normal distribution. This rescaling is useful for optimization algorithms and distance measurement, such as gradient descent, neural network, K nearest neighbours. Also, the column name in the first dataset needs transformation of the information. I can choose to extract the type of words and make them as new columns. Count the sum of possibilities of each type of the words as new entries and remove the original entries. The new columns help us report the solutions for the performance of each type of words and reduce the dimension from 1000 words. But the trade-off is information loss. But does this binning good?

From *[1]*, A good binning algorithm is:

1. Missing values binning separately,

2. Each bin contains at least 5% of observations

3. No bins contains 0 accounts of good or bad.

In out dataset, there is no missing values. Also, from the scatter plot, it satisfies condition 3.

(Full plots see **7.3**)

431 rows × 65 columns

Since there are 65 bins, each bin contains less than 5% observations and does not satisfies 2. So, I will not directly use it, compare with other feature selection methods in **part 4**.

3. Clustering

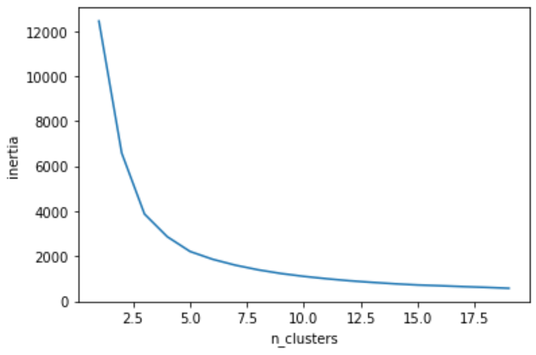
Clustering is an unsupervised machine learning method of identifying and grouping data points without labels. So, this method lets us classify data structures, more easily understand the pattern of data. I use PCA for dimensional reduction to 2 columns, so it is easier to visualize clustering.

3.1 K-means Clustering

K-means clustering is a distance-based, iterative algorithm that partition datasets into K pre-defined non-overlapping groups. It assigns data points to a cluster by minimizing the sum of the squared distance between the data points and the cluster's centroid..

Inertia

First, I run the K-means by different number of clusters, and use inertia in python, which measures the sum of squared distance of samples to their closest cluster center. K-means clustering algorithm aims to minimize inertia. Inertia makes assumption that clusters are convex and isotropic. The lower value of inertia is, the more we accept its assumption. That is one way to measure the performance of clustering.



When number of clusters is below 5, there is a high rate to reduce inertia. Inertia helps us testing model assumptions, but we cannot use this blindly. We can always make it small by adding more clusters. So, I will compare other methods of measuring performance.

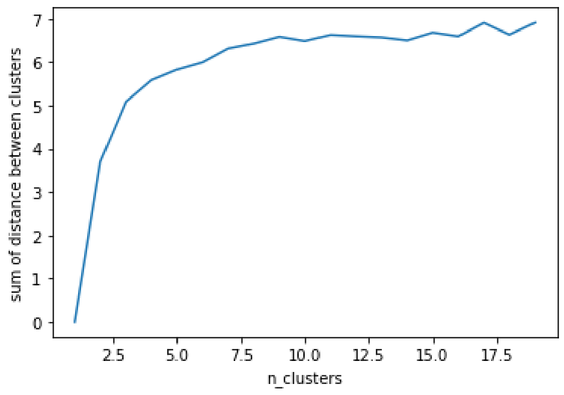
Inter cluster distance

Inter cluster distance is the sum of the square distance between each cluster centroid. If a clustering algorithm makes clusters so that the inter cluster distance between different clusters is big, we can tell that it is a good clustering algorithm.

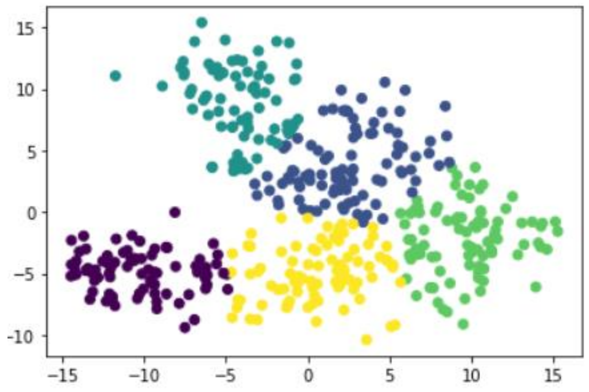
Intra cluster distance (for each cluster)

Intra cluster distance is the sum of the square distance from the items of each cluster to its centroid. If a clustering algorithm makes clusters so that the intra cluster distance between different clusters is low, we can tell that it is a good clustering algorithm.

I am calculating its average intra cluster distance.



When number of clusters is below 5, there is a higher rate to increase intra cluster distance. So, by all above, choose number of clusters equals 5.

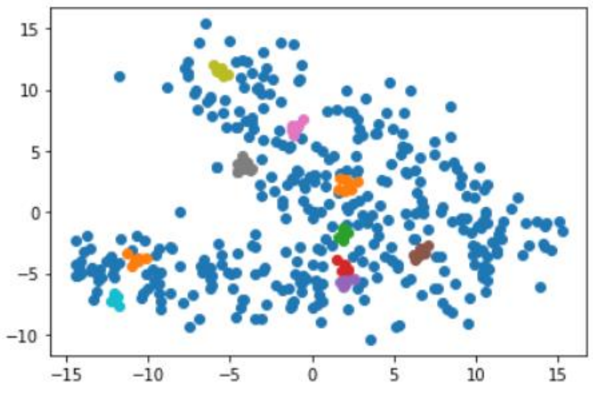


Summary

The cluster has the same sizes and spherical. After I choose the best parameter of the number of clusters, each cluster is coherent or tight themselves. So, it is a good clustering. We can apply this result to the dataset by adding a new column and then taking the average of each row that belongs to the same group.

3.2 DBSCAN

DBSCAN is a clustering method used in machine learning to separate high-density clusters from clusters of low density. DBSCAN can handle outliers within the dataset, but it does not work well when dealing with similar density clusters. I set up the maximum distance between 2 samples (eps) as 0.7 and minimum 6 objects in each core object's neighbourhood.



By the image, it might not be useful to explain the clustering. So, it's necessary to use its silhouette score for a numerical measurement result. Silhouette score is calculated by mean intra cluster distance between points and its mean nearest-cluster distance. When the cluster with high density and far from other clusters will have a strong silhouette score. The silhouette score ranges from -1 to 1. If close to 1, the score is high. But my result for this dataset is a score of -0.39, which is not a good performance in this scoring method.

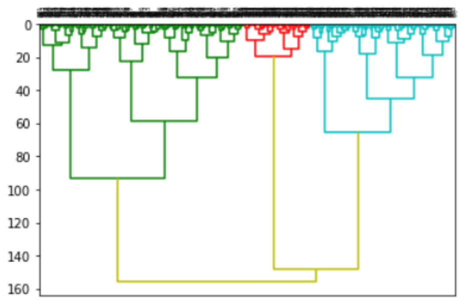
3.3 Hierarchical clustering

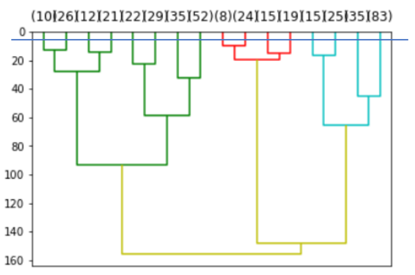
Hierarchical clustering deals with data in the form of trees. The process use measure such as distance to neighbours to decide which objects might be in the same cluster.

I'm using agglomerative hierarchical clustering. In this algorithm, each object is initially considered as a single leaf. At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (nodes). This step is iterated until all points are in this cluster.

I'm using the bottom-up approach, since it's simpler and faster. The bottom-up method assumes each object is its cluster, then iteratively joins the two most similar clusters. Also, I choose Ward's distance measure, because it's usually the best measurement, but the most expensive one. Ward's distance measure is the sum of within-cluster sum of squares for both clusters.

Graph for hierarchical clustering:





Y-axis is Ward's distance measurement, and X-axis is the column name in the index. The order of joining from this graph is from up to down. The bigger the distance between two links, the more difference of chosen features. For our graphs, there are four colours, and each colour represents a group of data. We can add a line to define clusters when the height equals 10 (the graph on the right) so that below this line, we can find clusters with all groups of data.

4 Simple prediction

4.1 Discussion

Since there are lots of attributes but 436 records, this is a medium size of the dataset. We can try neural networks but may need to be concerned about its high computational cost and consider principle analysis. Also, for K nearest neighbour, Bayesian predictor, their computational cost is lower than the neural network. For SVM and random forest, they are the best fit choices by comparing computational cost. Other than computational costs, here are the reasons that valuable to try each model:

4.1.1 Neural network system

The neural network system is: first, analyze the input data and detects the properties of the inputs, layer by layer, and gives the output. Then the network system adjusts its internal weightings to the answers. The adjustment helps in improving its performance.

4.1.2 K nearest neighbour

K nearest neighbour assumed similar things are close to each other. KNN is a lazy learner, which makes it fast, and new data can be added seamlessly, which will not impact the algorithm's accuracy.

4.1.3 Bayesian predictor

Bayesian predictor depends on Bayes theorem. It requires a small amount of training data to reduce its training period. Also, when the attributes are conditionally independents for its predictors, the Bayesian predictor performs the best.

4.1.4 Support vector machine

Support vector machine is used to sort two data groups by hyperplanes, to separate the groups according to patterns. SVM is effective in high dimensional spaces, and it’s memory efficient.

4.1.5 Random forest

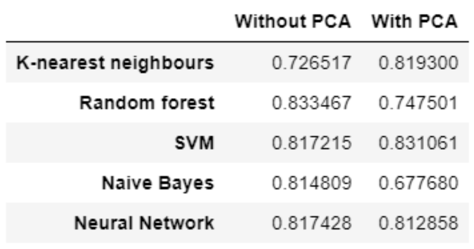
Random forest is based on the bagging algorithm and uses an ensemble learning technique. It grows the decision tree to full size and combines the output of all the trees. So, it can reduce the overfitting problem and minimize the variance.

4.1.6 Overfitting

Overfitting happens when the model does not generalize well from training data. To avoid over-fitting, I’m using cross-validation score, regularization (in part 2). Cross-validation is a technique minimizes overfitting. I’m setting the number of folds as 5. It iteratively trains the algorithm on 4 of the 5 folds and remain one as test set. The folds are made by Stratified folds since it preserves the percentages of samples for each class. Each fold is representative of all strata of data.

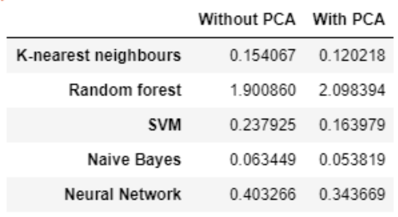
4.2 Classification by default settings

In this part, I’m doing classifications by the default setting of each model and then testing their performance. Set the random state statements to make sure the model is the same. Here is the accuracy table for basic classifications by cross-validation with 5 folders:



	Without PCA	With PCA
K-nearest neighbours	0.726517	0.819300
Random forest	0.833467	0.747501
SVM	0.817215	0.831061
Naive Bayes	0.814809	0.677680
Neural Network	0.817428	0.812858

Since by the above discussion, PCA is valuable to consider for using. As the prediction result with or without PCA, it is valuable to add PCA in SVM and K-nearest neighbours, and it's considerable to add PCA in Neural Network. But PCA does not work well in Random forest and Naïve Bayes. PCA does not work well in Random forest because I reduce the number of features and get rid of collinear features. In Naïve Bayes, the assumption is data are conditionally independent. PCA calculates the coordinate rotation based on statistics of the entire set of samples. So, it might not help with the conditionally independent in the Naïve Bayes theorem. Another disadvantage for PCA is principal components are not as readable and interpretable as original features; there is information loss. This disadvantage is also a reason why PCA does not work well in some of the algorithms.

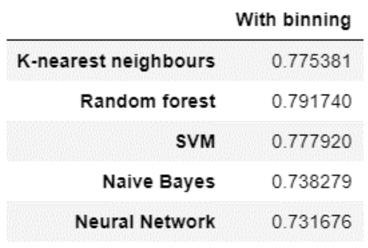


	Without PCA	With PCA
K-nearest neighbours	0.154067	0.120218
Random forest	1.900860	2.098394
SVM	0.237925	0.163979
Naive Bayes	0.063449	0.053819
Neural Network	0.403266	0.343669

This picture measures the execution time (in seconds) for each model, including cross-validation, and it does not calculate the timing for PCA preparation. For our dataset, Random forest costs the highest computational cost, and Naïve Bayes costs the lowest. Since the dataset is not large enough, Neutral Network does not cost too much computational cost.

4.3 Binning

To find the kind of words that is associated with winning speeches, binning method in **2.3** helps us to know which kind of words are associated with the winning speeches.



	With binning
K-nearest neighbours	0.775381
Random forest	0.791740
SVM	0.777920
Naive Bayes	0.738279
Neural Network	0.731676

Since the prediction accuracy is about 70+%, binning methods, the binning techniques include some information. But the binning process makes the most of the predictions worse a lot, except for K-nearest neighbours. Binning methods have a disadvantage of information loss, but it makes more similar things closer to each other, making advantage of K-nearest neighbours. In **4.4,** I will use feature selection techniques for binning to see which column is the best.

4.4 Applying attribute selection techniques

Attribute selections help us reduce irrelevant attributes, reduce type 1 and 2 errors. Also, it can help me reduces overfitting, improves accuracy, and reduces training time. I will use random forest feature selection, forward feature selection, and backward feature elimination.

4.4.1 Random forest

The random forest consists of hundreds of decision trees, each built over a random extraction of the observations from the dataset. When we train a tree, it is possible to compute how much each feature decreases the impurity, such as Gini. The more decrease the impurity, the more important the variable is. That is our strategy to select features.

4.4.2 Forward feature selection

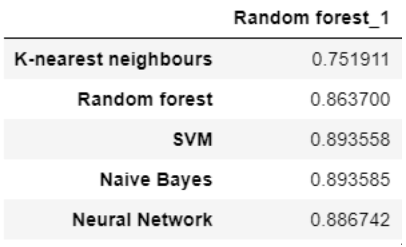
Forward feature selection is an iterative method where we start with the best performing variable against the target. Then, at each stage, the estimator adds the feature that improves the model based on an estimator's cross-validation score until no further improvement.

4.4.3 Backward feature selection

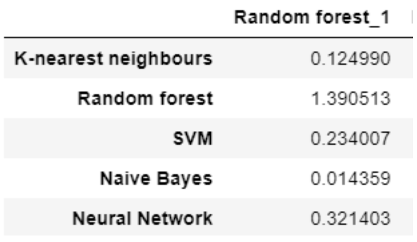
Backward feature elimination works exactly the opposite as forward feature selection. At first, start with all the features, then remove the least significant feature at each stage until no further improvement.

Since forward and backward feature selection needs a high computational cost, I'm applying both methods when the number of columns is lower than 100.

4.4.4 Discover for Random forest:



	Random forest_1
K-nearest neighbours	0.751911
Random forest	0.863700
SVM	0.893558
Naive Bayes	0.893585
Neural Network	0.886742



	Random forest_1
K-nearest neighbours	0.124990
Random forest	1.390513
SVM	0.234007
Naive Bayes	0.014359
Neural Network	0.321403

Random forest's time execution for selecting attribute is 0.463s.

From random forest feature selection, it selects 265 features. I'm mostly using this method since it has the highest accuracy and not requires a high computational cost

4.4.5 Compare with PCA:

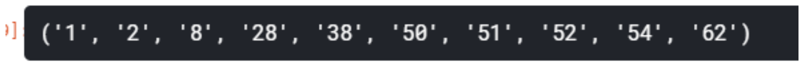
Also, PCA tells us 279 columns explains 95% of the total variance, 53 columns explain 50% of the total variance. If we choose to use fewer columns, it might cause a loss of information for us to predict. By comparing PCA's accuracy in **4.2** with random forest, a random forest is better in accuracy and execution time.

4.4.6 Grouping and forward & backward feature selection:

Since choosing random forest feature selection, I selected the top methods when we group the data.

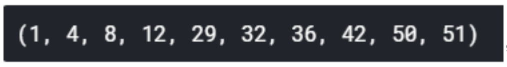
By forward and backward feature selection with SVM, it shows the first 10 groups are the best features.

By forward feature selection with neural network,



```
('1', '2', '8', '28', '38', '50', '51', '52', '54', '62')
```

's column is chosen the best.

For XGB boosting, 

```
(1, 4, 8, 12, 29, 32, 36, 42, 50, 51)
```

's column s chosen the best

4.4.7 Summary

The random forest is far beyond other methods for its accuracy and timing.

For each feature selection technique, it gives us different words that are associated with winner speeches. Random forest selects 265 features, PCA (95%) selects 279 features.

For each prediction algorithm with forward & backward feature selection, the words associated with winning speeches are different. All of them choose column number 1 and 8, which is 'CC' and 'DT'. So, for all these three classifiers (Bayes, Neural network, SVM), the words with the type of 'CC' and 'DT' are the most related to the prediction of winning speeches.

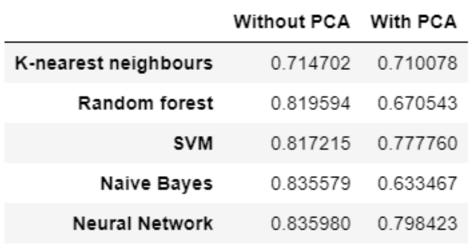
In summary, the words associated with winner speeches depends on which feature selection technique we choose. By choosing one of the selection techniques and get the best words, machine learning models can collectively suggest any strategies for good speeches.

4.5 bagging and boosting

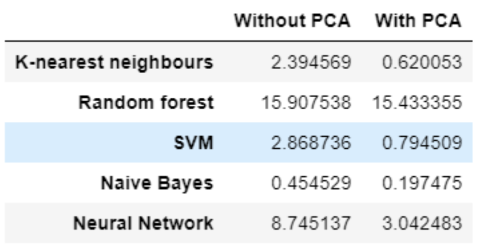
Bagging is a way to decrease the variance in the prediction by generating additional data for training from the dataset using combinations with repetitions to produce multi-sets of the original data. Boosting is an iterative technique that adjusts the weight of an observation based on the last classification. If an observation is classified incorrectly, it tries to increase the weight of this observation. So, bagging and boosting help us deal with variances.

Bagging:

Bagging when setting each bag built on 70% of the samples and 70% of features:
(left side is accuracy, right side is time execution)

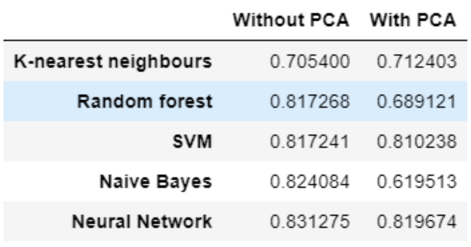


	Without PCA	With PCA
K-nearest neighbours	0.714702	0.710078
Random forest	0.819594	0.670543
SVM	0.817215	0.777760
Naive Bayes	0.835579	0.633467
Neural Network	0.835980	0.798423

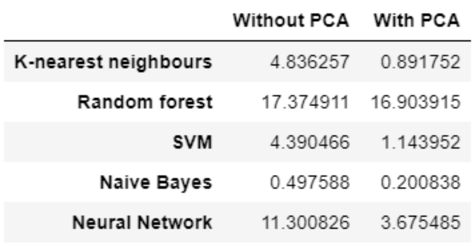


	Without PCA	With PCA
K-nearest neighbours	2.394569	0.620053
Random forest	15.907538	15.433355
SVM	2.868736	0.794509
Naive Bayes	0.454529	0.197475
Neural Network	8.745137	3.042483

Bagging when setting each bag built on 90% of the samples and 90% of features:

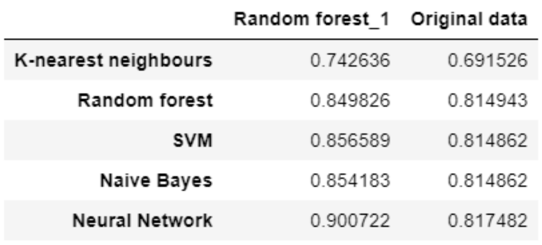


	Without PCA	With PCA
K-nearest neighbours	0.705400	0.712403
Random forest	0.817268	0.689121
SVM	0.817241	0.810238
Naive Bayes	0.824084	0.619513
Neural Network	0.831275	0.819674

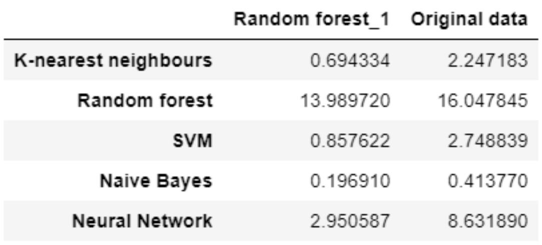


	Without PCA	With PCA
K-nearest neighbours	4.836257	0.891752
Random forest	17.374911	16.903915
SVM	4.390466	1.143952
Naive Bayes	0.497588	0.200838
Neural Network	11.300826	3.675485

Random forest feature selection + Bagging (70% of the samples and 70% of features):

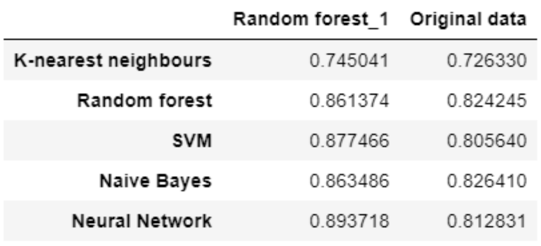


	Random forest_1	Original data
K-nearest neighbours	0.742636	0.691526
Random forest	0.849826	0.814943
SVM	0.856589	0.814862
Naive Bayes	0.854183	0.814862
Neural Network	0.900722	0.817482

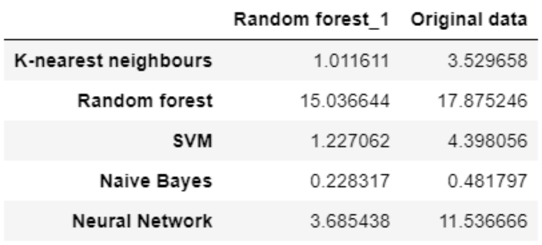


	Random forest_1	Original data
K-nearest neighbours	0.694334	2.247183
Random forest	13.989720	16.047845
SVM	0.857622	2.748839
Naive Bayes	0.196910	0.413770
Neural Network	2.950587	8.631890

Random forest feature selection + Bagging (90% of the samples and 90% of features):



	Random forest_1	Original data
K-nearest neighbours	0.745041	0.726330
Random forest	0.861374	0.824245
SVM	0.877466	0.805640
Naive Bayes	0.863486	0.826410
Neural Network	0.893718	0.812831



	Random forest_1	Original data
K-nearest neighbours	1.011611	3.529658
Random forest	15.036644	17.875246
SVM	1.227062	4.398056
Naive Bayes	0.228317	0.481797
Neural Network	3.685438	11.536666

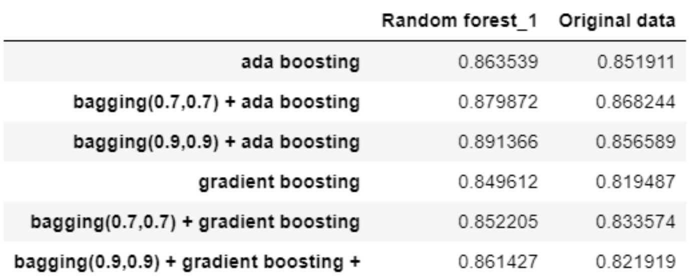
Discussion:

By the result above, Bagging with feature selection is better than bagging on the original dataset. Also, bagging with 90% of the samples and 90% of features is worse than both 70% for each model's performance on the original dataset, but better when using feature selection, except for neural network. Bagging costs a long time on the Neural network but less time on other classifiers.

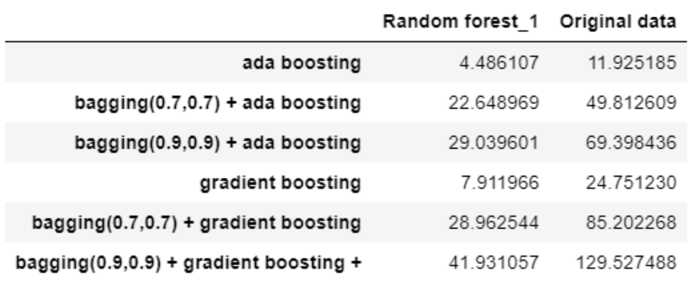
Boosting

I'm choosing adaptive boosting, Gradient boosting and Extreme Gradient Boost classifiers. Adaptive boosting works well with a decision tree, boosting model's learning from previous mistakes by increasing the weight of misclassified data points. Gradient boosting uses the average model and decision tree as estimators, and it is derived from the weights optimized by gradient descent to minimize the cost function. Extreme Gradient Boost is faster than other Gradient boosting a lot, includes a variety of regulation which reduces overfitting and improve model performance.

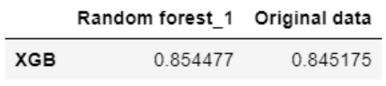
Boosting Results



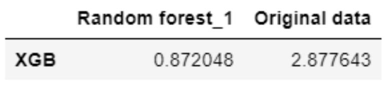
	Random forest_1	Original data
ada boosting	0.863539	0.851911
bagging(0.7,0.7) + ada boosting	0.879872	0.868244
bagging(0.9,0.9) + ada boosting	0.891366	0.856589
gradient boosting	0.849612	0.819487
bagging(0.7,0.7) + gradient boosting	0.852205	0.833574
bagging(0.9,0.9) + gradient boosting +	0.861427	0.821919



	Random forest_1	Original data
ada boosting	4.486107	11.925185
bagging(0.7,0.7) + ada boosting	22.648969	49.812609
bagging(0.9,0.9) + ada boosting	29.039601	69.398436
gradient boosting	7.911966	24.751230
bagging(0.7,0.7) + gradient boosting	28.962544	85.202268
bagging(0.9,0.9) + gradient boosting +	41.931057	129.527488



	Random forest_1	Original data
XGB	0.854477	0.845175



	Random forest_1	Original data
XGB	0.872048	2.877643

Discussion:

From the result above, by random forest selection, ada boosting with bagging (0.9,0.9), we get the best accuracy. Ada boosting works the best among these boosting methods, But XGB boosting has an extremely low time execution.

4.6 Summary

By all above, we need a model with high accuracy and relatively low time consumption. I'm choosing the following:

1. Neural Network: Random forest feature selection + Bagging (70% of the samples and 70% of features)
2. SVM: Random forest feature selection + Bagging (90% of the samples and 90% of features)
3. XGB: Random forest feature selection

These first two models have the best accuracy, and the third has the minimized computational cost.

5 Predictor choice

5.1 Hyperparameter pruning: SVM, Neural Network

In this part, I will choose the best predictors for our selected models.

For SVM, we are deciding which parameter is the best. The penalty parameter trades off correct classification of training examples against maximization of the decision function margin, so more points contains in the boundary. The gamma parameter defines how far the influence of a single training example reaches. The kernel functions are some functions that correspond to an inner product in some expanded feature space. So, I will use python's gridsearchCV method, a hyperparameter pruning method, choose different penalty, gamma, and kernel function, to get an optimal choice of parameters. I will try these parameters:

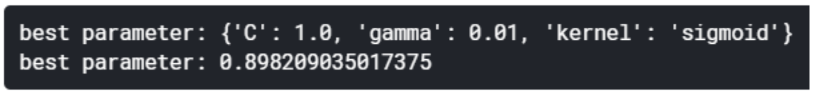
Penalty parameter **C**: e^2, e^1, ···, e^10

Gamma: e^-9, e^1, ···, e^3

Kernel: linear, polynomial, rbf(exp(-gamma ||x-x'||)**2) , sigmoid (tanh(gamma * xt * x))

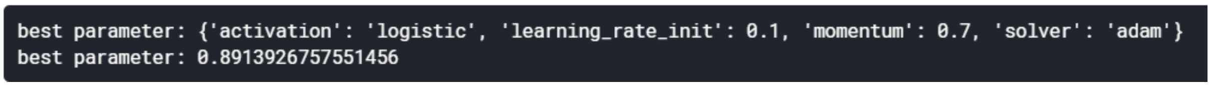
For neural network, I will consider activation function for hidden layers, solver for weight optimization, learning rate and momentum. The activation function is a function that is attached to each neuron in the network. The functions are mathematical equations and have the outputs. The solver is a gradient descent-based optimizer for optimizing the parameters in the computational graph. The learning rate alters the step size in the error space. Momentum affects the tendency to keep moving in the same direction. I will try these parameters:
Activation function: identity function, f(x) = x; logistic function, f(x) = 1/ (1+exp(-x)); tanh function, f(x) = tanh(x); rectified linear unit : f(x) = max(0,x)
Solver: 'lbfgs': optimizer by quasi-Newton methods; 'sgd': stochastic gradient descent; 'adam': stochastic gradient descent optimizer.
Learning rate: 0.001, 0.002, 0.005, 0.01, 0.05, 0.1, 0.5
Momentum: from 0.7 – 0.95, step is 0.05

The result for SVM is:



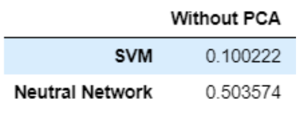
```
best parameter: {'C': 1.0, 'gamma': 0.01, 'kernel': 'sigmoid'}
best parameter: 0.898209035017375
```

The result for neural network is:



```
best parameter: {'activation': 'logistic', 'learning_rate_init': 0.1, 'momentum': 0.7, 'solver': 'adam'}
best parameter: 0.8913926757551456
```

Timing for the optimizing parameters (with cross-validation of 5 folders):



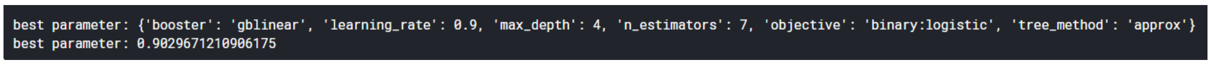
	Without PCA
SVM	0.100222
Neutral Network	0.503574

5.2 Hyperparameter pruning: XGB boosting

For XGB boosting, I'm choosing the following parameters to get the optimal choice: booster, learning rate, max depth, number of estimators, objective and tree method.

Booster is specifying which booster to use, gbtree or gblinear. Gblinear uses linear regression with l1&l2 shrinkage, and gbtree uses a regression tree as a week learner.

Learning rate is boosting the learning rate. Max depth is the maximum tree depth for base learners. The tree method is the tree construction algorithm used in XGBoost. Like auto method, it uses a heuristic to choose the fastest method. For a large dataset, 'approx' is a good parameter for chosen. Approx parameter approximate greedy algorithm using quantile sketch and gradient histogram.

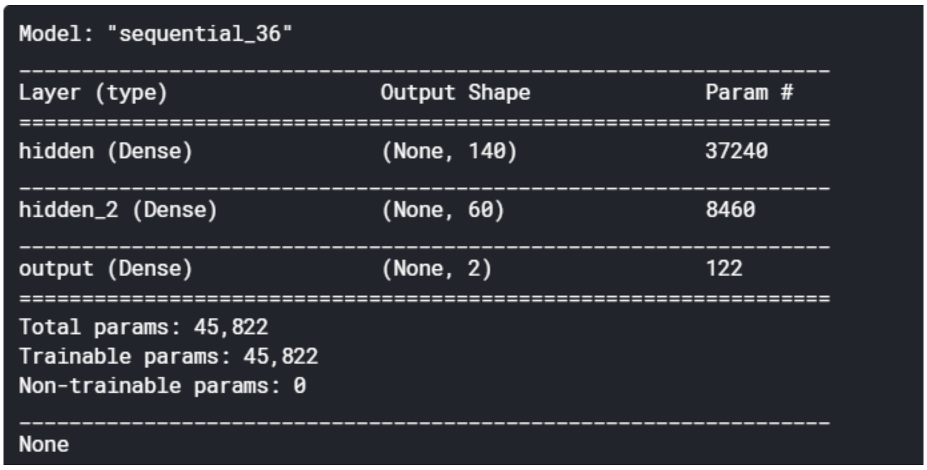


```
best parameter: {'booster': 'gblinear', 'learning_rate': 0.9, 'max_depth': 4, 'n_estimators': 7, 'objective': 'binary:logistic', 'tree_method': 'approx'}
best parameter: 0.9029671210906175
```

Execution time (without random forest feature selection) : `0.07597164999992856`

5.3 Self-setting Neural network - Python Keras

Instead of calling neural network classifier API in python, I want to try keras to self setting the neural network.



```
Model: "sequential_36"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
hidden (Dense)               (None, 140)               37240
_________________________________________________________________
hidden_2 (Dense)             (None, 60)                8460
_________________________________________________________________
output (Dense)               (None, 2)                 122
=================================================================
Total params: 45,822
Trainable params: 45,822
Non-trainable params: 0
_________________________________________________________________
None
```

It's the structure for my self-setting neural network. For minimize the computational cost, I'm using random forest feature selection and has the input dimension of 265 columns. Then I set the dense layers by different activation functions. The activation function is nonlinear function that allows the network to learn complex relationship between the predictors and target variables (0,1). I choose to use activation function 'relu' and 'softma''. Relu is an activation function defined as f(x) = max(x,0). Softma is a function that turns a vector of x real values into a vector of x real values that sum to 1. After adding dense layers, I compile the model by specify the optimizer and loss function. I'm using adam as my optimizer, it's a stochastic gradient descent method; also set the loss function as 'categorical_crossentropy', The loss function is:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

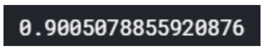
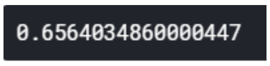
It's a good measure of how distinguishable two discrete probability distribution are from each other. Finally, training the model and setting the size of samples and determine how the gradient is calculated for each sample.

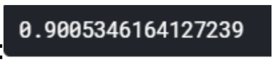
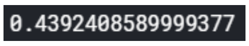
The **score** with self-setting cross-validation by StratifiedKFold: (since there is no direct API for Keras with cross validation score): `0.8750334143638611`

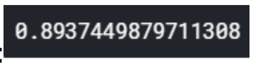
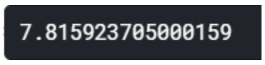
Total **timing**: `138.18322120799894`seconds

Compare with other models, the accuracy is not very high, and requires high computational cost. Since training with original dataset might provides a better result, but the high computational cost is the barrier for me to use this. Also, no hyperparameter setting in Keras is another disadvantage that requires more coding works to find the best parameter.

5.4 Add bagging

XGB: Accuracy: 
```
0.9005078855920876
```
time: 
```
0.6564034860000447
```

SVM: Accuracy: 
```
0.9005346164127239
```
time: 
```
0.4392408589999377
```

Neural network: Accuracy: 
```
0.8937449879711308
```
time 
```
7.815923705000159
```

5.5 Payoff

By the result of prediction, the winner model is XGB + Random forest feature selection, which has an accuracy about 90% and total timing of 0.533(0.07+0.463) seconds. Other good models are: Neural network + Random forest feature selection + bagging, and SVM + Random forest feature selection + bagging. They all have a cross-validation score of more than 90% but a higher computational cost.

For our question, there are large numbers of attributes. Random forest feature selection helps us reduce the dimension and get highly accurate and interpretable data. XGB boosting give us a high performance by regulation, effective tree pruning. XGB uses L1 regulation and L2 regulation, which prevents the overfitting. Also, tree pruning is a kind of greedy algorithm, which makes splits until the max steps we wrote for, or there is a negative loss. So that might be the reason why XGB boosting works the best. Also, I told about XGB is faster than other boosting algorithms before, and compare with other models with 90% accuracy, our winner model does not have bagging. So that is the reason why it's fast.

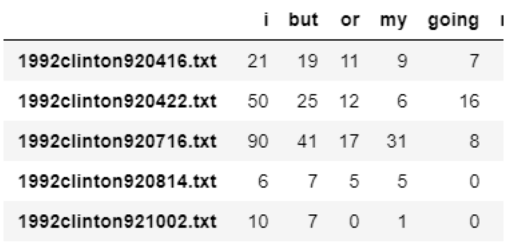
For the other two good models, bagging does not help a lot when we already reduce the dimension of data, but it can still increase performance. The payoff for using bagging is the complexity of the model.

If we only consider the accuracy, the forward feature selection method and backward feature selection elimination are beneficial for improving accuracy by the greedy algorithm, but the computational cost is high. My computer cannot afford the high computational cost when doing these two algorithms with our dataset. Previously I tried this method with the first 182 columns of data, and it gives me the best result for that data frame.

6. deception language?

Now I'm adding this dataset into our original word dataset to see if it can improve our result. The reasons for each step I already wrote before, so quickly repeat the steps.

6.1 Properties of dataset



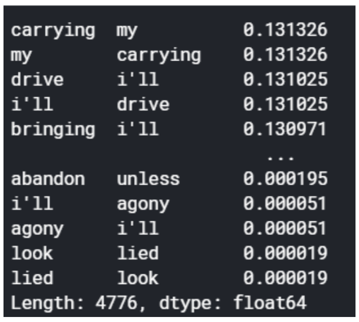
	i	but	or	my	going
1992clinton920416.txt	21	19	11	9	7
1992clinton920422.txt	50	25	12	6	16
1992clinton920716.txt	90	41	17	31	8
1992clinton920814.txt	6	7	5	5	0
1992clinton921002.txt	10	7	0	1	0

The shape of the data is (431,76). The row name is the same as the previous one, and the column name is deceptive words. Each entry is a word occurrence. This dataset describes the occurrence of each word in each speech. To apply it in the original one, we need to change it into percentages in each speech. But there is no total number of words in each speech. A better way is using the normalizer, but it may still have some bias.

6.2 null data

From **2.2**, there is no null data. Also, there is no null and wrong columns.

6.3 correlation



```
carrying  my          0.131326
my        carrying    0.131326
drive     i'll        0.131025
i'll      drive       0.131025
bringing  i'll        0.130971
                        ...
abandon   unless      0.000195
i'll      agony       0.000051
agony     i'll        0.000051
look      lied        0.000019
lied      look        0.000019
Length: 4776, dtype: float64
```

This data frame has a very low correlation among each attribute. So, from discussion with PCA in **1.2.1**, when data is not highly correlated, PCA not works very well. Also, we don't need to consider multicollinearity (discussed in **1.2**).

6.4 normalizer

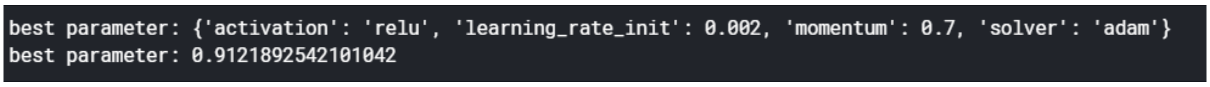
First, I find the biggest value in original data frame. According to this value, I use MinmaxScaler in python to scale the deception data frame between 0 and the biggest value in original data frame. Then, use standard scaler to make the mean as 0, standard deviation as 1.

6.5 feature selection and prediction

By discussion in 4 and 5, feature selection with SVM, neural network, naïve bayes makes the best solution. I'm using the same method to improve the result.

By feature selection with random forest, I select 277 columns, which is lower than previous (279).

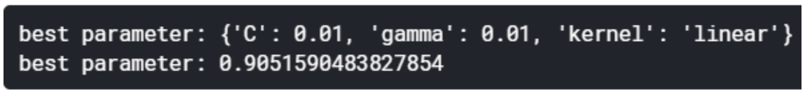
Neural Network:



```
best parameter: {'activation': 'relu', 'learning_rate_init': 0.002, 'momentum': 0.7, 'solver': 'adam'}
best parameter: 0.9121892542101042
```

Time: `0.9566910789999383`

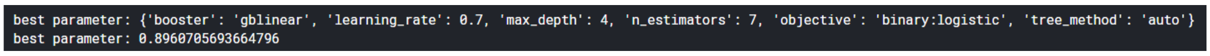
SVM:



```
best parameter: {'C': 0.01, 'gamma': 0.01, 'kernel': 'linear'}
best parameter: 0.9051590483827854
```

Time: `0.1241147999999157`

XGB Boost:



```
best parameter: {'booster': 'gblinear', 'learning_rate': 0.7, 'max_depth': 4, 'n_estimators': 7, 'objective': 'binary:logistic', 'tree_method': 'auto'}
best parameter: 0.8960705693664796
```

Time: `0.0818694360000336`

After Bagging:

Neural Network: Accuracy: `0.8936380646885859`, time: `8.585778667999989`

XGB Boosting: Accuracy: `0.9098904036353916`, time: `0.7292257670000026`

SVM: Accuracy: `0.9074846297781342`, time: `0.4914454050000358 6`

6.6 Payoff

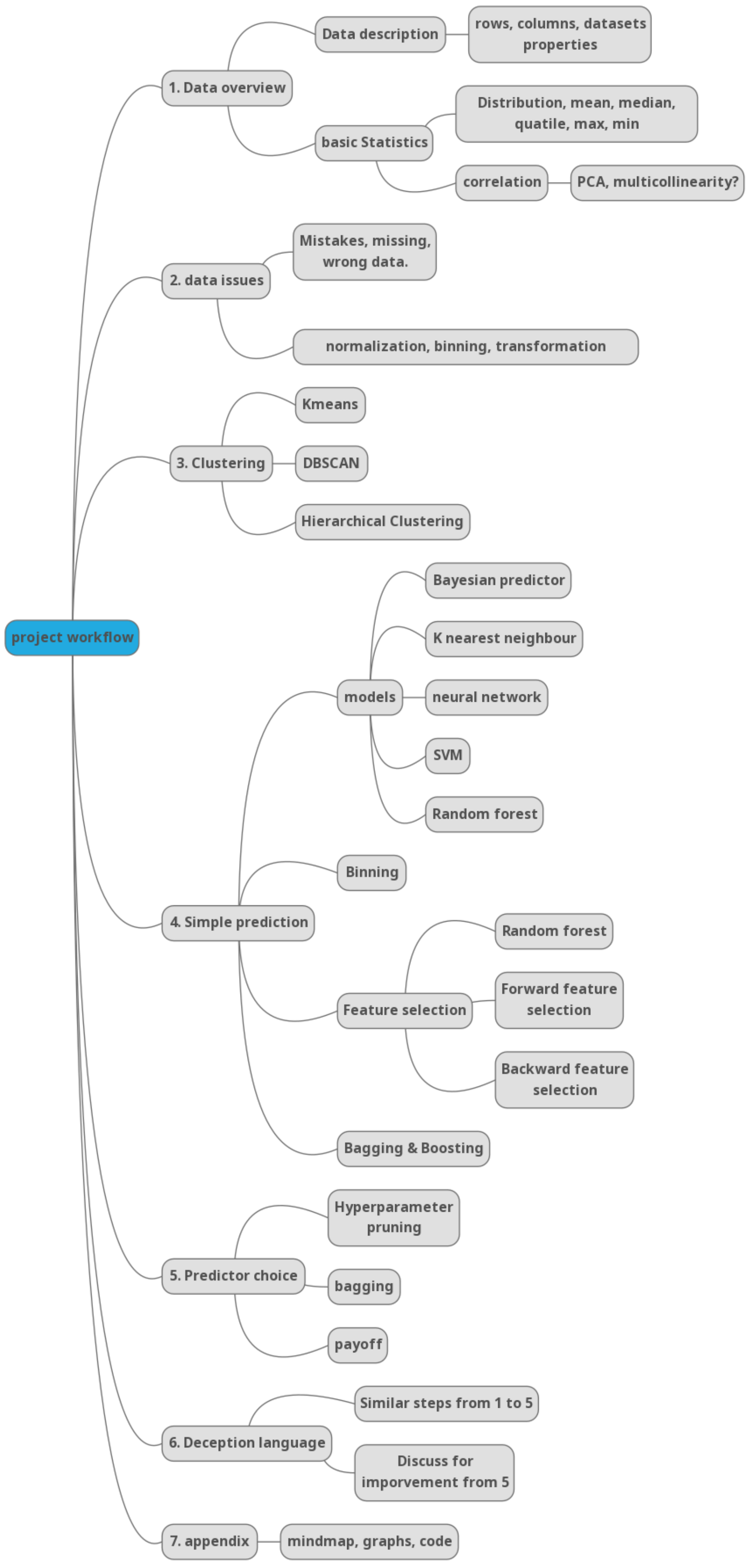
By comparing these results, the neural network with random forest feature selection works the best, which has accuracy 91.2%. SVM also has a great result and less computational cost than the neural network. For XGB, it has the lowest computational cost but the worst accuracy among these three models. The bagging method improves accuracy in XGB Boosting and SVM, but its computational cost is a lot higher.

Compared with part 5, with more data by deception language, the results improve about 1%. The best result increases from 90% to 91.3% accuracy. The deception helps our prediction from random forest feature selection, and we select some rows from deception language instead of the previous dataset. More data always gives us better results since sample size dictates the amount of information we have.

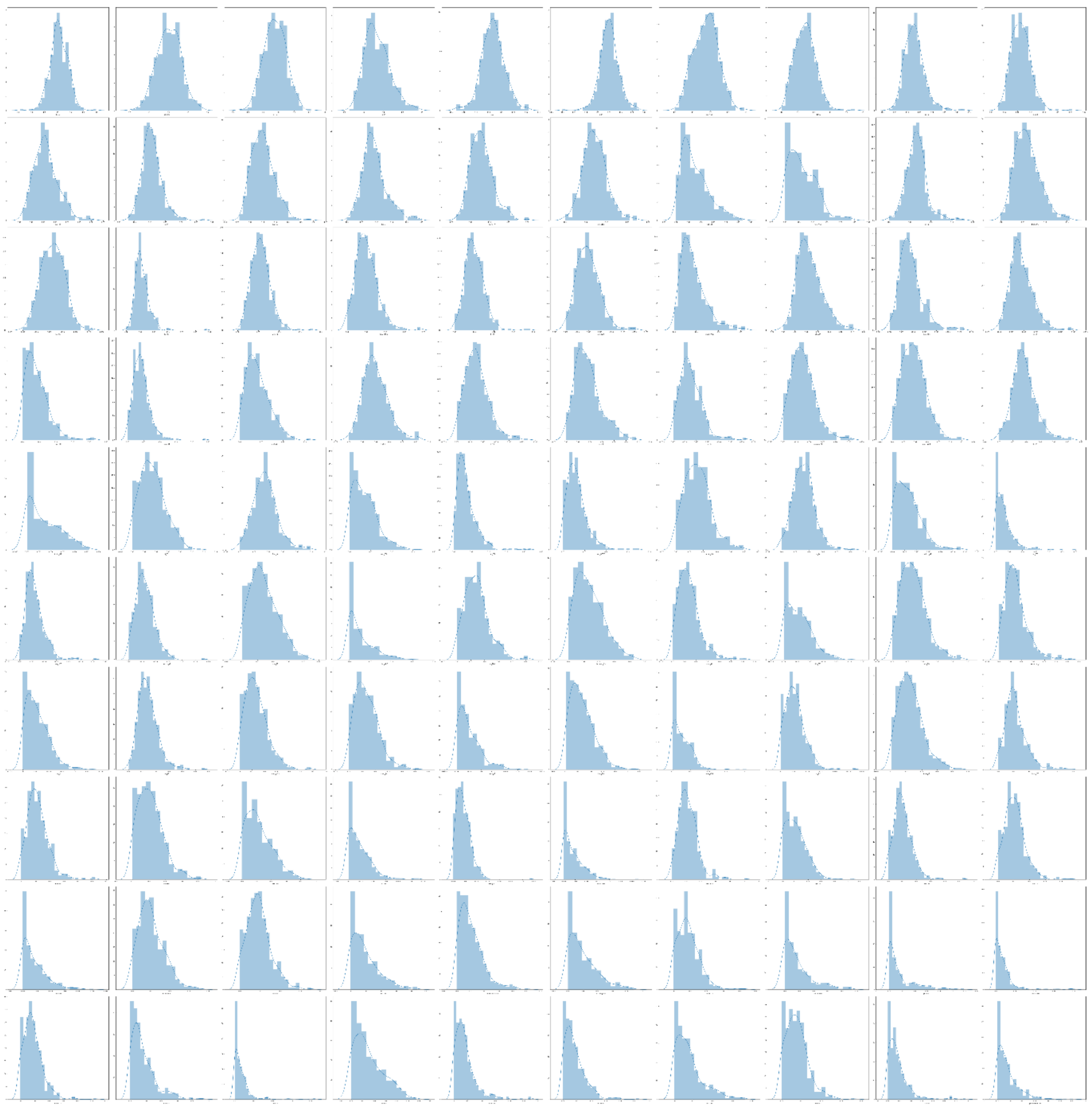
For improve the model, the scaling of data is important. From my method of scaling, I'm assuming in deception words, the best occurrence percentage among all words in each speech is same as the original data. In original dataset, the best entry is 0.08, so I set the minmax scaler from 0 to 0.8. It might cause an error since we don’t know how many words actually in each speech, so we need require data for how many words actually in each speech for improve the minmax scaler.

7. Appendix

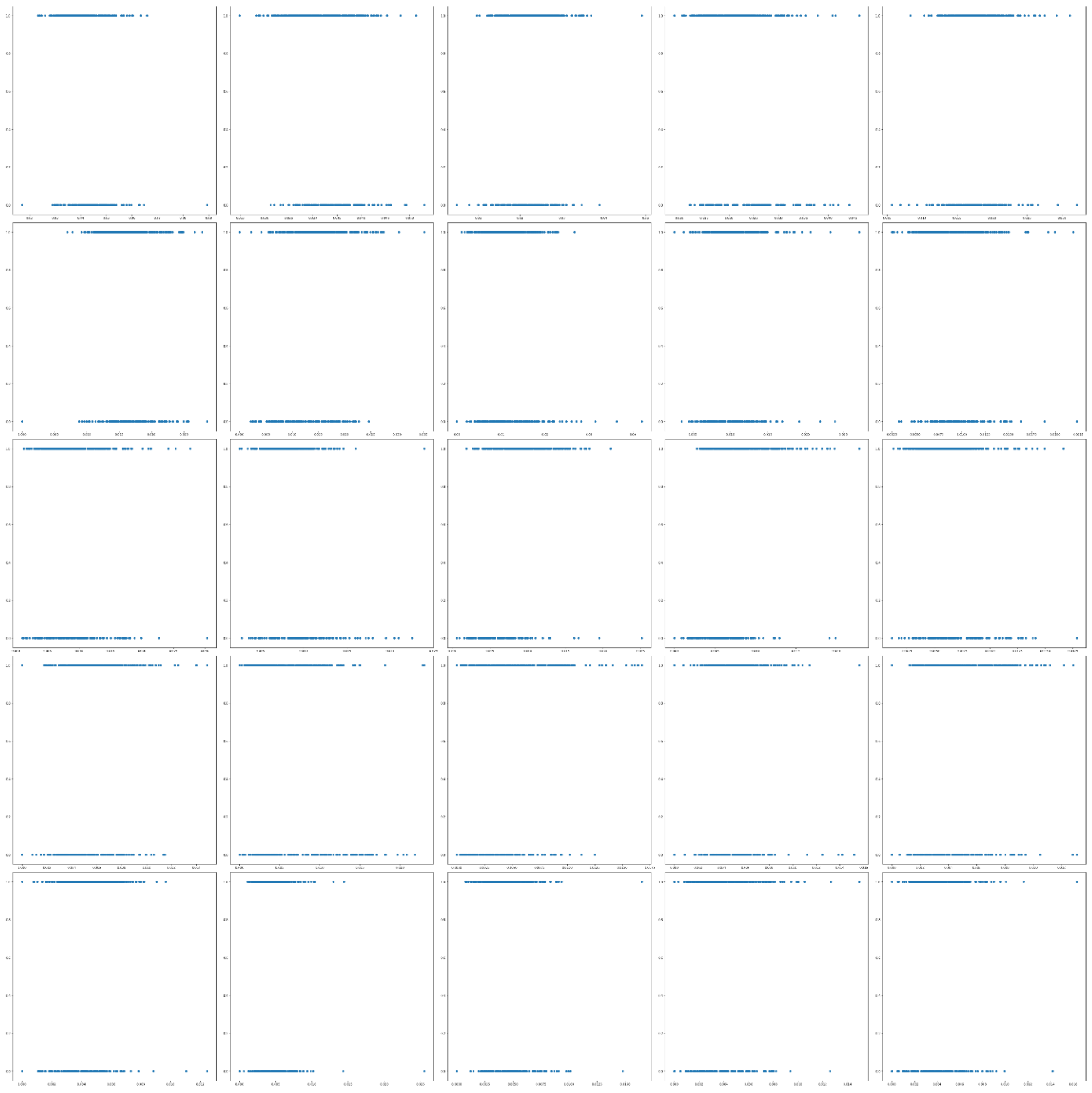
7.1 Overall workflow



7.2 **Histogram** (first 100 graphs) in **1.2**



7.3 **Scatter plot** for **2.3** (first 25 attributes)



7.4 Reference

[1] Good, I. J. (1985). Weight of Evidence: A Brief Survey. BAYESWSTATISTICS 2, 249-270.

7.5 Coding in python

Since I use python in this project, the following is some parts of coding:

Scatter plot

```
from sklearn.preprocessing import StandardScaler
from matplotlib.pyplot import scatter
transformer = StandardScaler().fit(data_X)
data_X_normalized = pd.DataFrame(transformer.transform(data_X))
data_X_normalized.columns = data_X.columns
fig = plt.figure(figsize=(50,50))
for index, column in enumerate(data_X.columns[0:25]):
    plt.subplot(5,5,index+1)
    scatter(x = data_X[column], y = data_y)
    fig.tight_layout(pad=1.0)
```

Missing value

```
data_X.isnull().sum().sort_values(ascending = False)
```

Data binning

```
data_binning = data_X.copy()
column_list = []
for name in data_binning.columns:
    type_word = re.split(r'\_', name)[1]
    if type_word not in column_list:
        data_binning[type_word] = 0
        column_list.append(type_word)
    data_binning[type_word] += data_binning[name]
```

```
data_binning = data_binning.iloc[:,999:]
data_binning
```

(after delete one column)

DBSCAN

```
#DBSCAN
from numpy import unique
from numpy import where
from sklearn.cluster import DBSCAN
from matplotlib import pyplot
#model = DBSCAN(eps=0.7, min_samples=6)
model = DBSCAN(eps=0.8, min_samples=7)
yhat = model.fit_predict(data_X2)
clusters = unique(yhat)
for cluster in clusters:
    row_ix = where(yhat == cluster)
    pyplot.scatter(data_X2[row_ix, 0], data_X2[row_ix, 1])
pyplot.show()
```

PCA

```
pca = PCA(n_components=0.95)
pca.fit(data_X)
print(pca.explained_variance_ratio_)
```

KNN

```
start = timer()
neigh2 = KNeighborsClassifier()
scores = cross_val_score(neigh2, data_X1, np.ravel(data_y), cv=5)
end = timer()
score_list.append(sum(scores)/5)
sum(scores)/5
time_list.append(end - start)
```

Random forest

```
start = timer()
clf = RandomForestClassifier(random_state = 40)
scores = cross_val_score(clf, data_X1, np.ravel(data_y), cv=5)
end = timer()
score_list.append(sum(scores)/5)
sum(scores)/5
time_list.append(end - start)
```

SVM

```
start = timer()
clf_svc = svm.SVC()
scores = cross_val_score(clf_svc, data_X1, np.ravel(data_y), cv=5)
end = timer()
score_list.append(sum(scores)/5)
sum(scores)/5
time_list.append(end - start)
```

Naive Bayes

```
start = timer()
gnb = GaussianNB()
scores = cross_val_score(gnb, data_X1, np.ravel(data_y), cv=5)
end = timer()
score_list.append(sum(scores)/5)
sum(scores)/5
time_list.append(end - start)
```

Neural Network

```
start = timer()
clf_neuralnetwork = MLPClassifier(solver='lbfgs', random_state = 40)
scores = cross_val_score(clf_neuralnetwork, data_X1, np.ravel(data_y), cv=5)
end = timer()
score_list.append(sum(scores)/5)
sum(scores)/5
time_list.append(end - start)
```

GridSearchCV (hyperparameter pruning)

```
#parameters
gamma_range = np.logspace(-2,10,13)
C_range = np.logspace(-9,3,13)
kernel_choice = ['linear', 'poly','rbf','sigmoid']
param_grid = dict(gamma = gamma_range, C = C_range, kernel = kernel_choice)
grid = GridSearchCV(svm.SVC(), param_grid = param_grid, cv = 5)
grid.fit(data_X, np.ravel(data_y))
print('best parameter:', grid.best_params_)
print('best parameter:', grid.best_score_)
```

Forward feature selection

```
score_list = []
time_list = []
def func_timing_score(clf_importance, X, y = data_y):
    sel = SequentialFeatureSelector(clf_importance, k_features = 10)
    sel.fit(X, y)
    start = timer()
    X_important = sel.transform(X)
    scores = cross_val_score(clf_importance, X_important, np.ravel(data_y), cv=5)
    end = timer()
    score_list.append(sum(scores)/5)
    time_list.append(end - start)
```

Backward feature selection

```
def func_timing_score(clf_importance, X, y = data_y):
    sel = SequentialFeatureSelector(clf_importance, k_features = 10, forward = False)
    sel.fit(X, y)
    start = timer()
    X_important = sel.transform(X)
    best_parameter_name_id.append(X_important.k_feature_idx_)
    scores = cross_val_score(clf_importance, X_important, np.ravel(data_y), cv=5)
    end = timer()
    score_list.append(sum(scores)/5)
    time_list.append(end - start)
```

Random forest attribute selection

```
from sklearn.feature_selection import SelectFromModel
start = timer()
sel = SelectFromModel(RandomForestClassifier(n_estimators = 100))
sel.fit(data_X, data_y)
data_X1 = pd.DataFrame(data_X)
X_important_RF = sel.transform(data_X)
end = timer()
selected_feat= pd.DataFrame(data_X).columns[(sel.get_support())]
end- start
```

Self-setting neural network:

```
score_list = []
time_list = []
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
for train_index, test_index in skf.split(X_important_RF, data_y):
    y = to_categorical(data_y)
    X_train, X_test = X_important_RF[train_index], X_important_RF[test_index]
    y_train, y_test = y[train_index], y[test_index]
    #print("TRAIN:", train_index, "TEST:", test_index)
    start = timer()
    #model
    model = Sequential()
    model.add(Dense(140, activation='relu', input_dim=265, name = 'hidden'))
    model.add(Dense(60, activation='relu', name = 'hidden_2'))
    model.add(Dense(2, activation='softmax', name = 'output'))
    # Compile the model
    model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=100, batch_size = 1, validation_split=0.2, verbose = 0)
    scores = model.evaluate(X_test, y_test, verbose=0)
    end = timer()
    score_list.append(scores)
    time_list.append(end - start)
```

Advanced neural network (still trying)

```
model = Sequential(
    [
        keras.Input(shape=(data_X.shape[0],data_X.shape[1], 1)),
        layers.Conv2D(32, kernel_size=(2, 2), activation="relu"),
        layers.Conv2D(32, kernel_size=(2, 2), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Dropout(0.125),
        layers.Conv2D(32, kernel_size=(2, 2), activation="relu"),
        layers.Conv2D(32, kernel_size=(2, 2), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Dropout(0.125),
        layers.Flatten(),
        layers.Dropout(0.25),
        layers.Dense(256, activation = "relu"),
        layers.Dropout(0.5),
        layers.Dense(10, activation="softmax"),
        layers.Dense(1, activation="softmax")

    ]
)

model.summary()
```