Preliminary notes

<u>The goals of the overall analytic task:</u>

The primary question our analytics attempts to answer is this: What kind of pattern in the language of a speech predicts whether a candidate will win or lose an election? There are three specific goals we address while answering this question. Our goals are:

1. To make the most accurate prediction of the likelihood of winning a US presidential election based on the words used in speeches.
    a. Although the wording of this goal suggests a regression problem, this is a classification problem. Each row represents a campaign speech, and each column shows the frequency of a particular word in such speech. Thus, our model tells whether the speech it is given is associated with a winning campaign or a losing one. We rigorously assess the performances of various predictive models, and subsequently define a final model of choice.
2. To find some of the words that might be associated with winning speeches, and to see if they collectively suggest any strategies for good speeches.
    a. Once we build a strong model that knows how to separate a winning speech from a losing one, we rank the attributes in terms of their predictive power, as this lets us see which words might be predictive of winning. We then discuss whether such words show any possible strategies for good speeches.
3. To see if deceptive language has any role in winning US presidential elections.

<u>The process our analytic task takes is as follows. We are to:</u>

1. Investigate the dataset statistically and learn more about its properties. This involves looking at the distribution of the values for attributes of interest. This also means looking at averages, medians, quartile ranges, and maximums and minimums across the records. We also show useful visualizations.
2. Look for data issues. Missing or wrong data can inadvertently impact how strongly models learn and predict. This would be the first sub-step of data pre-processing.
3. Carry out any data transformations. This would be the second sub-step of data pre-processing. This step involves asking the following question: are there attributes whose values need to be manipulated in some way to allow the model to perform particularly well? Supposing we discretize the values, some models would not be able to handle categorical attributes. Other models cannot train on numerical attributes as they are, and would thus need to be normalized. We apply different transformations to the dataset depending on the model we are attempting.
    a. One separate topic in data pre-processing is attribute selection. Some models, such as neural networks, Bayesian predictors, and K-nearest neighbour, only perform well on smaller datasets, which means that it may be useful to have fewer records and attributes. It may be useful to manipulate the dataset in terms of size for certain models.
4. Build clustering models. With clustering, our goal is to discover the underlying structure of the data. We are attempting to understand the patterns that exist within the data. In sum, it allows us to understand the relationships amongst records based on similarity. It also gives us a sense of how difficult the prediction problem might be. We can decide on a final clustering of choice for future records.
5. Build prediction models and assess each model's results. The model that provides the strongest performance would be considered the final predictive model of choice. This would be the suggested model for future records.
6. Consider the overall payoff of steps one-five. This is a simple reflection on our process.

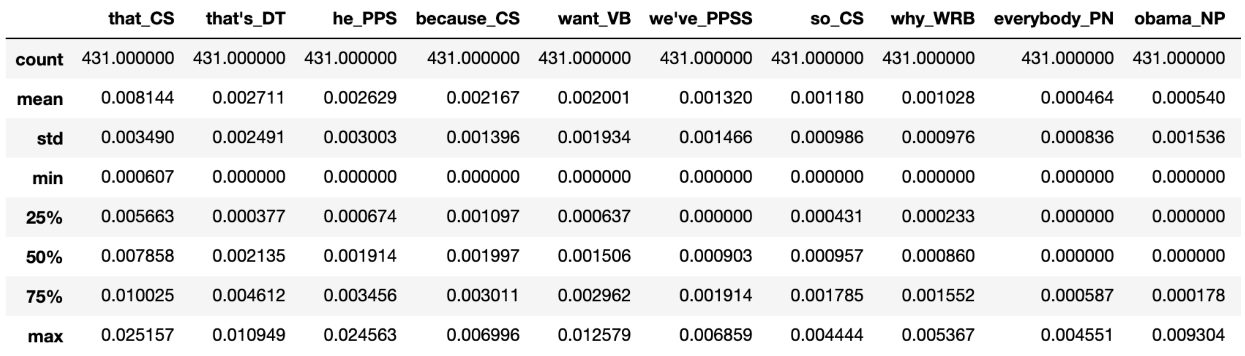
<u>Other notes:</u>
A majority of the data analytic work is conducted using Python. When using KNIME, the expectation is to post screenshots of the workflows. Given that I am using Python, this report includes code snippets instead. These code snippets are the equivalent of workflow screenshots or visuals and would not be considered "text" for this report. The size of these code snippiets is minimized as much as possible. Any "housekeeping" operations such as loading CSV files, storing the data in objects, and producing the scatterplot is not included in these snippets. All of these operations are done mostly using the `numpy` and `pandas` libraries. Thus, please note that every code snippet in this report includes the following imports. These imports are not explicitly stated. Listed on the following page.

```python
import numpy as np
import pandas as pd
```

Report

Investigative statistics:

Our discussion on page 10 of this document shows the 10 most predictive attributes. Some statistics on these attributes are shown here.



	that_CS	that's_DT	he_PPS	because_CS	want_VB	we've_PPSS	so_CS	why_WRB	everybody_PN	obama_NP
count	431.000000	431.000000	431.000000	431.000000	431.000000	431.000000	431.000000	431.000000	431.000000	431.000000
mean	0.008144	0.002711	0.002629	0.002167	0.002001	0.001320	0.001180	0.001028	0.000464	0.000540
std	0.003490	0.002491	0.003003	0.001396	0.001934	0.001466	0.000986	0.000976	0.000836	0.001536
min	0.000607	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.005663	0.000377	0.000674	0.001097	0.000637	0.000000	0.000431	0.000233	0.000000	0.000000
50%	0.007858	0.002135	0.001914	0.001997	0.001506	0.000903	0.000957	0.000860	0.000000	0.000000
75%	0.010025	0.004612	0.003456	0.003011	0.002962	0.001914	0.001785	0.001552	0.000587	0.000178
max	0.025157	0.010949	0.024563	0.006996	0.012579	0.006859	0.004444	0.005367	0.004551	0.009304

Figure 1: Statistics on some records associated with the top 10 attributes.

It would not be particularly useful to show screenshots of statistical values for every attribute.

Data issues:

A modified dataset was developed from the mostfreq1000docword.csv file. However, the original dataset was not discarded because the modified version consists of a nearly 20% reduction in the number of attributes: 817 attributes as opposed to the original 1000. Such a reduction in data is not trivial. Thus, both the original dataset and modified datasets are assessed in our analytic process. Please assume that, unless otherwise noted, the results reported on this document is associated with the modified dataset. The modified dataset consists of the changes outlined below.

Some words were incomplete or accented. These were removed because they can skew the model results in a direction without reason. Since there are only three instances of this in the dataset, this change is not significant.

Stop words were also removed. These are high-frequency words that do not significantly contribute to the meaning of a speech — that do not influence an impression of competence or lack thereof in the minds of voters. Examples of such words include "the", "of", "a", "an", "to" etc, and simple nouns such as "we", "him" etc. Removal of such words allows machine learning algorithms to better focus on the words that genuinely impact the meaning of a given speech. It resolves any ambiguity in the pattern in language of the speech. The result of this careful selection in attirbutes is better performance by models. Neural networks, for example, are designed to capture the subtle interactions among the words that *truly* determine the overall essence of the speech. Including stop words — words with little impact on the record or observation — do not allow a neural network to determine such subtle interactions. Note that the list of stop words used is from the `NLTK` library in Python.

A common approach is to standardize the data (i.e. a z-score normalization). This scales the inputs to have a mean of 0 and a variance of 1. In some cases, however, it may be preferable to do a min-max normalization. In our case, all of the data for all the models are standardized. The module used in Python were `StandardScaler` of `sklearn`.

Building clustering models:

Generally, our goal in clustering is to try to understand the patterns exhibited within certain sets of records in the data, to find the sub-populations that exist. The following associated questions are of interest:

- How many sub-populations are there?
- What are their sizes?
- Do elements in a sub-population have any common properties?
- Are the sub-populations cohesive? Can they be further split up?
- Are there outliers?

In our particular analytic problem, we want to find the most natural division that emerges from a specification of two sub-populations. Ideally, one sub-population would consist of records associated with a winning campaign, while the other sub-population would consist of records associated with a losing one. Note that there *are* clustering algorithms in which we do not need to specify the number of sub-populations, but the ones that *do* require specification are of interest to us in our analytic task.

To have a sense of which clustering algorithms might provide strong insights, it would be useful to visualize the data on a scatterplot. This would provide an understandable representation of how the records are similar to one another. Thus, for the purpose of dimensionality reduction, principal component analysis (PCA) was conducted. This reduced the number of dimensions (i.e. attributes) from 817 to 2.

PCA is a robust method for dimensionality reduction. It uses feature extraction, a technique that extracts information from the existing feature set to build a new feature subspace. PCA finds the axis, a "principal component", which captures the maximum variance or spread in the high-dimensional data. Every subsequent axis the algorithm finds runs in a direction perpendicular to the previous axis *and* accounts for the next highest variance. After the axes are derived, the data points from the high-dimensional set are projected onto the new subspace through a specific linear mapping.

The `PCA` module of Python's `sklearn` library was used. The `numpy` array `features` contains all of the data from our dataset except for the target attribute: winners. Thus, its dimensions are 431 rows x 817 columns. PCA was implemented as follows.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

#To show use of StandardScaler.
features = StandardScaler().fit_transform(features)

pca = PCA(n_components = 2)
principalComponents = pca.fit_transform(features)
principalDf = pd.DataFrame(data = principalComponents, columns = ["PC1",
"PC2"])
```

`features` was standardized. `principalDf` is a `DataFrame` object which consists of the data after dimensionality reduction. The first column corresponds with the first principal component, PC1, and the second principal component, PC2.

On the following page is a scatterplot representing the data points on the principal component axes — that is, the points listed on `principalDf`. Each of the points is color coded by the output label with which it is associated.

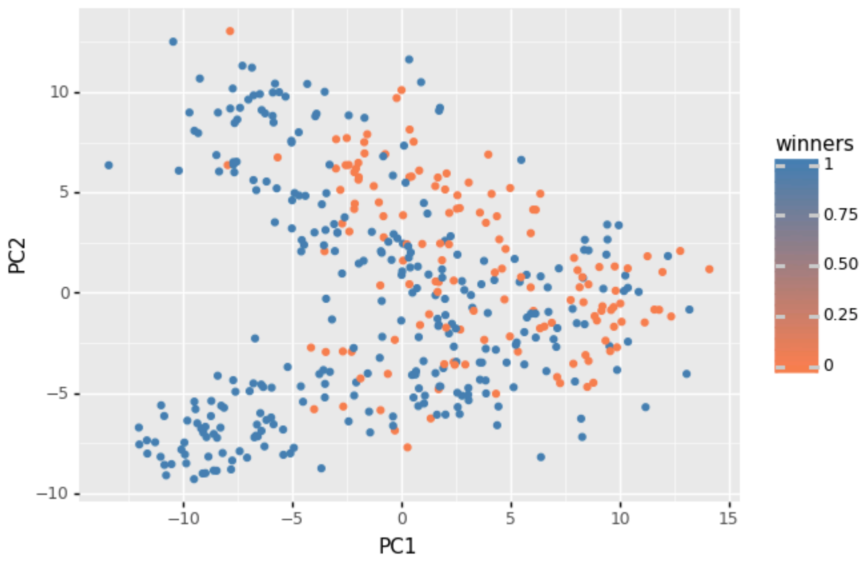


Figure 2: Scatter plot of speeches after PCA.

It is important to note that while the legend indicates a color gradient, the data point colors are absolutely discrete: either blue or orange, or 1 or 0, respectively. Blue represents a winning campaign and orange represents a losing campaign. PC1 represents the first principal component and PC2 represents the second principal component.

If we consider all winning speeches to be in a single cohesive group or cluster and all losing speeches to be in another group, we notice a major overlap between the groups. The losing speeches overlap with the winning ones by a higher proportion than the winning speeches do with the losing ones, which means that losing speeches are generally more similar to winning ones than vice-versa. The winning records in the upper-left and bottom-left corners of the plot are markedly different from the losing ones: there are few losing speeches in these regions. These speeches might be what separate a winning campaign from a losing one. A second observation is that these two left regions, in addition to being separate from the losing speeches, are separate from *each other*. This is indicative of a significant sub-division within the set of winning speeches. This suggests that there are two substantially different kinds of speeches for a winning campaign.

Clustering is understood as a representation of records in tightly-bound, well-separated groups. Given that clusters generally do not overlap to the extent shown on the previous page, any formation of clusters in our data would not be by their class labels; in other words, we cannot define two clusters such that one would entirely be associated with a winning campaign and the other would completely be associated with a losing one. This inability to form clusters based on class labels gives us an idea of how truly difficult the prediction problem might be. Even though there is no clear natural separation (i.e. no good clustering that matches this plot can be formed) between winning speeches and losing ones, it is still useful to perform clustering. It gives us a general sense of the underlying structure within the data, and might be indicative of *some* separation of winning speeches from losing ones. Our goal is to best form clusters based on a division between winning speeches and losing ones. The clustering methods available to us are hierarchical clustering, expectation maximization (EM) with Gaussian Mixture Models (GMM), k-means clustering, density-based clustering, and single value decomposition (SVD).

Hierarchical and k-means clustering would not be useful for our goal. Hierarchical clustering is a hierarchical decomposition of the data based on group similarities. This method allows us to vary the degree of dissimilarity among the clusters: the lower we are on the dendrogram, the smaller the dissimiarity among the corresponding clusters, and the greater the number of clusters. Hierarchical clustering clearly does not fit our goal, as we know the number of clusters we would like to find. k-means clustering would also not be useful to our goal. Although this method appears as an obvious choice at first glance, k-means fails when some points are closer to the center of another cluster than to the center of their own. This happens when clusters are of different sizes, or the clusters are not spherical. Below is a visual example from the course slides that helps explain this. The red line represents the middle distance between the two centroids.

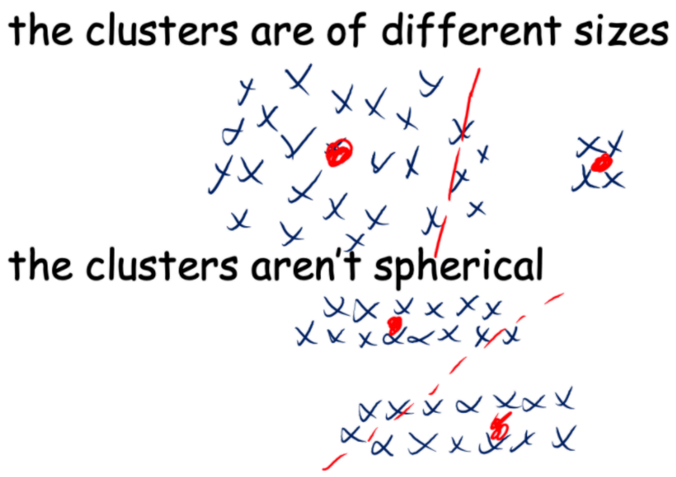

the clusters are of different sizes
the clusters aren't spherical

Figure 3: Example showing the primary flaw of k-means.

We notice that some points are incorrectly being considered in a the cluster to which it is not "joined".

Given our scatterplot on the previous page, since our hope is to form clusters divided on the basis of winning and losing speeches, we cannot reasonably expect them to be spherical or of similar sizes. The ones that do appear to fit our problem are EM with GMM (which we will refer to as "EM" for short), density-based clustering, and single value decomposition.

EM is suited for this data because one, points within a cluster are based on a Gaussian distribution, and two, points do not have hard assignments to clusters. For the first argument, EM says that while points exist within a cluster for a specific reason — a reason which can be represented by the mean of the distribution — there is some kind of random variation we expect to come along with such reason. Given that our goal is to form clusters based on a division between winning speeches and losing ones, the relevance of this algorithm for our data is intuitive. Not all speeches *associated* with a winning campaign can exemplify such a campaign. Not all of them account for the success of that campaign. Similarly, not all speeches linked to a losing campaign exemplify the speech for a losing candidate. Speeches closer to the centroid or within one standard deviation of the mean exemplify its cluster. For the second argument, soft assignments are important simply because they give us more information about the data. If the clusters indicate some level of separation that corresponds with class labels, they can help us see how likely a point is to fall in one cluster over another. This is good to know.

I conducted EM using the `GaussianMixture` module of the `sklearn` library on Python. `features` is our dataset for all attributes except for the target. Its dimensions is thus 431 rows x 817 columns. EM was implemented as follows.

```
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler

#Read CSV file as DataFrame object df. df.shape = (431, 818).

features = df.iloc[:, :(df.shape[1] - 1)]

features = StandardScaler().fit_transform(features)
EM = GaussianMixture(n_components = 2)
EM.fit(features)
clusters = EM.predict_proba(features)

#for i in clusters:
  #set clusters[i] to the max(clusters[i])
```

`EM.predict_proba` returns an array of probabilities for each cluster. The highest probability value is assigned to `clusters[i]`. Lastly, `n_components` represents the number of clusters we would like, which is 2.
The results were interesting. Firstly, the clustering with each execution of the program was different. In many of the executions, all of the points were assigned to a single cluster. This behavior is expected given how close together all of the records are. Here, EM would view the center of the scatterplot itself as the centroid for the cluster. Another observation is that probabilities were discrete: either 0% or 100%. One reason that might account for this is the sheer quantity of observations and attributes. With a high quantity of data, EM can state which cluster a record falls in with extremely high confidence.

For each execution of the program, a scatterplot was produced through PCA, with the points being color coded by cluster number. I continuously ran the program until the scatterplot that best matched the plot from figure 1 was found: this was considered to be a clustering that was most aligned with a separation by class labels. This plot is the second one below. For a side-by-side comparison, the original plot from figure 1 is also presented below. This is the first plot.

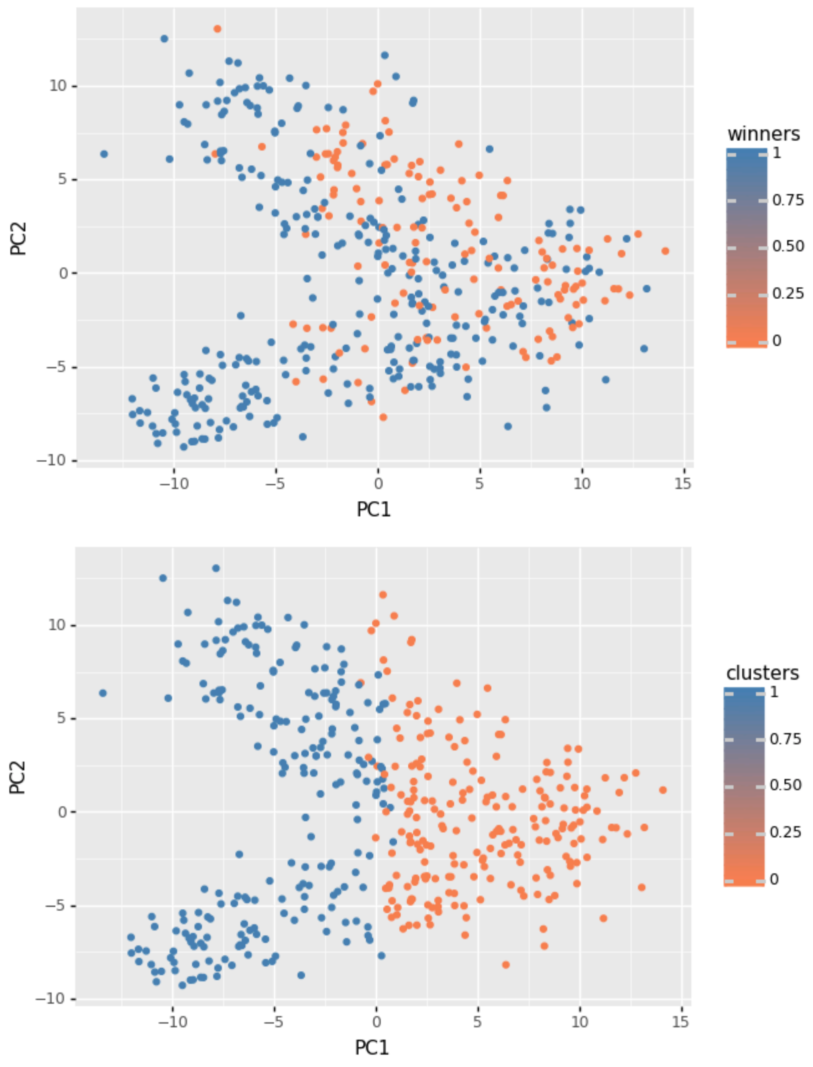


Figure 4: Scatter plot from EM compared with original. The original is the first plot; EM plot is second.

It is reasonable to assume that cluster 0 is associated with class label 0 (i.e. a loss) and cluster 1 is associated with class label 1 (i.e. a win). Given this, the result is insightful. We stated previously that it would not be possible to form a clustering that matches the original plot. The result we get from EM reinforces that statement.

It is important to remember that clustering is unsupervised learning, meaning that the algorithm does not know the values of the target attribute. Noting this, our result is important. This result tells us that there is some degree of natural separation that emerges between winning speeches and losing ones. In other words, some kind of separation between winning and losing speeches *on the basis of similarity* is possible. One indicator of this is the upper-left and bottom-left regions of this plot. The clustering algorithm was able to deduce that these two regions are similar, even though they are individually far apart from one another. Thus, although this result reinforces the notion that the prediction is indeed difficult, it is still encouraging as it has some degree of accuracy.

Although density-based clustering and SVD would be useful, EM has sufficiently given us the necessary information for understanding the underlying structure in the data. In a more in-depth study, it would be useful to explore clusterings of the attributes.

Building predictive models:

The predictors at our disposal are neural networks (NNs), random forests, decision trees, support vector machines (SVMs), Naive Bayes, and k-nearest neighbours (k-NN). All of these models are strong contenders for our analytic task and can give us good results, but some models are *more* suitable than others. The ones that appear to be particularly attractive are neural networks and random forests. Both of these are seriously attempted and analyzed for this project. Comments that outline my rationale for chossing these predictors are outlined below.

1. **Neural Network**: NNs are excellent because they can capture the subtle word-to-word interactions that produce different patterns in language, some of which are associated with a winning speech and some of which are associated with a losing one.
2. **Random Forest**: A random forest is useful because it easily handles irrelevant attributes. Handling irrelevant attributes is important given the sheer number of them in our dataset: the greater the number of attributes, the higher the likelihood many of these attributes would interfere in producing a correct prediction. It is useful to account for such interference. A random forest also allows us to assess our original dataset (i.e. the unmodified dataset that is 431 rows x 1000 columns) in case we inadvertently removed words that are *relevant*. Furthermore, through attribute selection, a random forest lets us address some of the remaining goals and questions in our overall data analytic task: namely, finding some of the words that might be associated with winning speeches, whether deceptive language and simple nouns are predictive of the winning attribute.

The other four models — SVM, k-NN, Naive Bayes, and the decision tree— are good as well. I only seriously attempt and analyze the SVM predictor however. Comments that outline my rationale for the choice of SVM from these four predictors are below.

1. **Support Vector Machine**: An SVM is a good choice because it can still maintain its efficacy when the number of dimensions is greater than the number of samples; this may not be the case for the other models.
2. **K-Nearest Neighbours**: While the kNN predictor is capable of giving good performance, but can be sensitive to outliers. An SVM can handle outliers. Furthermore, a kNN predictor would require much trial-and-error in terms of assessing various parameters; thus, it would be worthwhile to investigate this model in a more extensive project.
3. **Naive Bayes**: Naive Bayes serves as a model that sets the benchmark for the minimum performance we expect, because it is generally known to show good performance on text data. Prior to seriously beginning this project, I made a few brief predictors and saw strong results immediately, eliminating the need for seriously attempting and assessing Naive Bayes.
4. **Decision Tree**: It is not necessary to implement a decision tree as we are implementing a random forest.

In sum, the three predictors attempted and analyzed are neural networks, random forests, and SVMs. Note that methods such as bagging and boosting are not implemented. These are generally used for weak classifiers, which is not the case for the models tested. Random forest is also already an ensemble classifier.

To prevent repetition of code snippets, please note that the training-test split for all the models is done as follows. `df` is a `DataFrame` object that has read in the CSV file containing the data.

```
from sklearn.model_selection import train_test_split
features = df.iloc[:, :(df.shape[1] - 1)]
targets = df.iloc[:, (df.shape[1] - 1)]

trainX, testX, trainY, testY = train_test_split(features, targets,
test_size = 0.30)
```

A training-test split of 70-30 is done for all datasets. This is a good medium between either extreme of 60-40 or 80-20. Note that training-test splits are not adjusted to improve predictor performance: this split will always be used for the models.

The first attempted predictor was a random forest. This is good to start with because it can help us in feature ranking and thus improve the performance of later models. It is important to note that while there are several other algorithms that can rank attributes, a random forest produces a sensible and intuitive ranking. The ranking is based on the information gain produced by each attribute. Information gain tells whether the subset produced after splitting on an attribute is more pure than any of the other subsets produced after splitting on other attributes. The base implementation is as follows.

```
#Both trainX and testX are standardized using StandardScaler.
#trainY and testY are not standardized given that they are binary labels: 0
or 1.

from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 350,  max_features =
400, max_depth = 8, min_samples_leaf = 5)
classifier.fit(trainX, trainY)
predY = classifier.predict(testX)
```

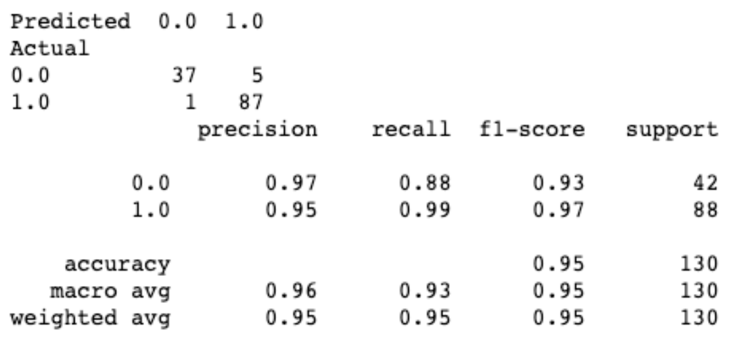
In terms of parameter selection for `classifier`, the parameters indicated above were the ones that provided strongest performance. The following actions were taken regarding each parameter. Avoiding overfitting to the training set was a major concern.

1. `n_estimators`: This is the number of trees. The goal was to continue growing a larger forest until performance improves at small margins. I started at a 100 trees: the default value for `RandomForestClassifier`. The more trees there are, the less likely the algorithm is to overfit.
2. `max_features`: This specifies the size of the subset of attributes each decision tree uses. The fewer the number of features, the less likely the forest is to overfit. An extremely small number however will produce underfitting. I started with 500 attributes and gradually reduced this number until a strong result was achieved.
3. `max_depth`: This parameter indicates the maximum depth of the tree. I started at a depth of 5 and increase until a strong result was achieved.
4. `min_samples_leaf`: This parameter indicates the minimum number of allowed samples for a leaf node. It is important to ensure that this value is greater than one. It has a similar effect as the `max_depth` parameter. It ensures that decision trees stop splitting once the subsets become extraordinarily small. I kept this number at 5.

It is useful to briefly discuss the intuition behind minimizing `max_depth` and maximizing `min_samples_leaf`. A decision tree produces a path to a leaf node when given a new record. This path must have enough supporting evidence from the training set to show that its prediction is viable for this new record; in other words, the subset produced by this path during training should be substantial. If it is not substantial — if the subset consists of one or two records, for example — the decision tree is likely to be overfitted to the training set, too specific or tailored to the records of the training set.

One additional note is that I was incorrect in my analysis during the data pre-processing stage. The original dataset (i.e. mostfreq1000docword.csv) produced substantially better results. This is understood since random forest systematically derives the a ranking in importance of different features. It is based on a clearly defined metric:

information gain. While my process of dataset modification was reasonable, it was not effective. Thus, I ranked attributes and produced a *second* modified dataset for use in our two future predictors. This attribute ranking is discussed more extensively after the presentation of results below. Below are the results of the best exectution of the data.



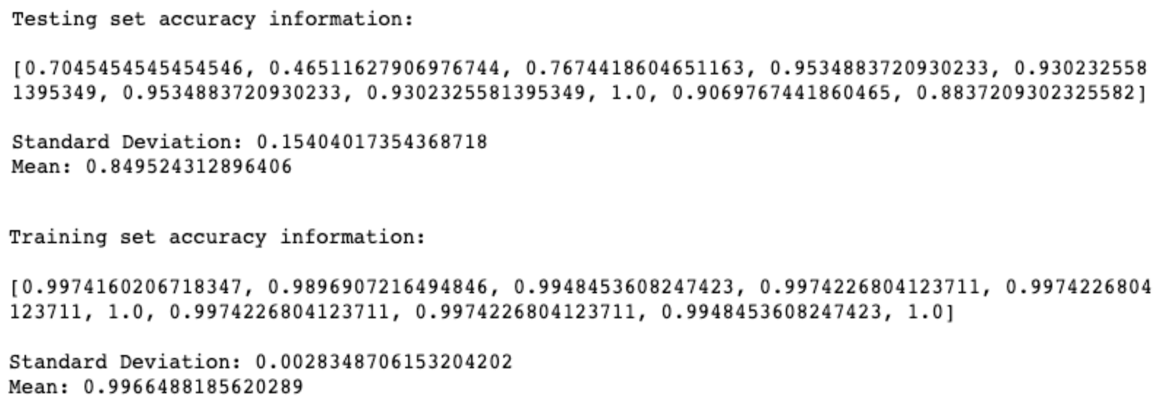
```
Predicted  0.0  1.0
Actual
0.0         37    5
1.0          1   87
              precision    recall  f1-score   support

         0.0       0.97      0.88      0.93        42
         1.0       0.95      0.99      0.97        88

    accuracy                           0.95       130
   macro avg       0.96      0.93      0.95       130
weighted avg       0.95      0.95      0.95       130
```

Figure 5: Strongest results obtained in single execution of random forest.

Note that 0 represents a loss while 1 represents a win. The top-half is the confusion matrix while the bottom-half consists of values for important metrics. The bottom-half is provided per `classification_report` of `scikit.metrics`. The precision, recall, f1-score, and accuracy values are specifically of strong concern to us.

There are some important things to derive in terms of the meaning of these results. This confusion matrix shows more false positives than false negatives. In most of the confusion matrices of the executions of this classifer, false positives were much more common than false negatives, meaning that the classifier was *more* likely to predict a win for a given speech than a loss. Although five false positives out of a total of 93 is not significant, it should not be ignored either. This number might suggest a slight bias of the classifier toward predicting winning speeches. This slight bias might exist because the classifier has seen more examples of winning speeches than losing speeches in the training data. The ratio of winning to losing speeches is 289 to 142. This ratio is not trivial at all. There is an imbalance between the two different types of records in the dataset.

In terms of further tests, to truly verify that the model is robust, I also performed a 10-fold cross-validation. The following results were achieved.



```
Testing set accuracy information:

[0.7045454545454546, 0.46511627906976744, 0.7674418604651163, 0.9534883720930233, 0.930232558
1395349, 0.9534883720930233, 0.9302325581395349, 1.0, 0.9069767441860465, 0.8837209302325582]

Standard Deviation: 0.15404017354368718
Mean: 0.849524312896406


Training set accuracy information:

[0.9974160206718347, 0.9896907216494846, 0.9948453608247423, 0.9974226804123711, 0.9974226804
123711, 1.0, 0.9974226804123711, 0.9974226804123711, 0.9948453608247423, 1.0]

Standard Deviation: 0.0028348706153204202
Mean: 0.9966488185620289
```

Figure 6: Results for random forest after cross-validation.

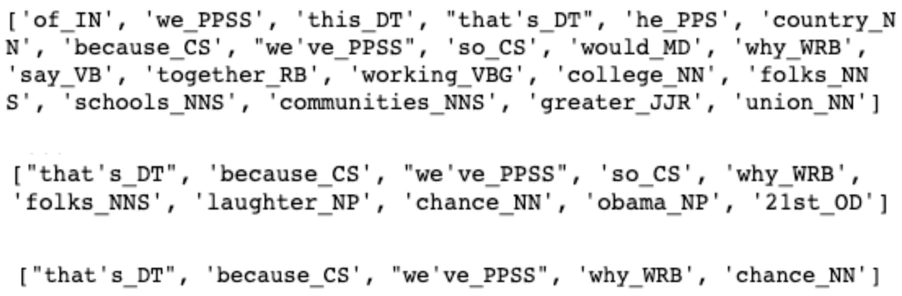
The values enclosed in brackets represent the prediction accuracies of each of the models. Standard deviation was reported because it can convey significantly more information than mean. It tells us that, on average, the model accuracies deviate from one another by 15%. We notice how different the accuracies can be in certain instances.

Regarding the training accuracies, it is noted that these values are extremely close to 100%. This might suggest that the model overfits. This is not true. When I modified the parameters such that training accuracies were reduced, testing accuracies were reduced substantially as well. In other words, reducing training accuracies in an effort to prevent overfitting resulted in low testing accuracies.

Random forest provides us information on which words might be the most predictive. I derived feature importance using random forest with the following attribute retrieval. This is a follow-up on the code above.

```
importance = model.feature_importances_
```

This returns a list of 2-tuples containing the importance of different features measured with a particular value. The attributes to be considered the most important by the predictor were sincerely surprising. The lists of the highest ranked 20, 10, and 5 words are as follows. These words include the Stanford tags.



```
['of_IN', 'we_PPSS', 'this_DT', "that's_DT", 'he_PPS', 'country_N
N', 'because_CS', "we've_PPSS", 'so_CS', 'would_MD', 'why_WRB',
'say_VB', 'together_RB', 'working_VBG', 'college_NN', 'folks_NN
S', 'schools_NNS', 'communities_NNS', 'greater_JJR', 'union_NN']

["that's_DT", 'because_CS', "we've_PPSS", 'so_CS', 'why_WRB',
'folks_NNS', 'laughter_NP', 'chance_NN', 'obama_NP', '21st_OD']

["that's_DT", 'because_CS', "we've_PPSS", 'why_WRB', 'chance_NN']
```

Figure 7: The most predictive 20, 10, and 5 words, in the order presented.

This result is incredibly interesting. It says that simple nouns are more predictive than credited: in fact, many of the words in these three lists *are* simple nouns. Additionally, one evident theme embedded in these three lists is togetherness. It would be logical to assume that higher frequencies of words associated with a theme of togetherness — words such as "we've", "folks", "communities", "country", "union", "together" — would be indicative of a winning speech. Lastly, this result strongly invalidates my modified dataset from the pre-processing stage. Many of these are stop words and are not included in that dataset. Regarding whether deceptive language is predictive, the language indicated in the words above also do not appear to indicate any kind of deception. This suggests that deceptive language is not highly predictive. In general, it would be difficult to accurately measure the level of deceptive language based on solely word frequencies. Detecting deception truly depends on how the words are presented, not what the words themselves are. Also, deception is not commonly measured in text analytics. Many of the libraries available deal with sentiment analysis as oppposed to deception analysis.

In a more extensive study, a good approach to detecting deception would be removing the words or attributes that are *known* to be associated with deception, and assessing the models with such modified datasets. If it makes a major difference in the prediction performance, then deceptive language *would* be considered to be predictive of the target attribute.

The next classifier attempted was a recurrent neural network (RNN). RNNs are good at modelling any kind of data in which very specific patterns matter. In our analytic task, this would be a pattern in language: a set of word frequencies unique to a speech. An RNN also implements a backpropagation algorithm, which is simply the adjustment of weights and biases based on how much a predicted result deviates from the expected result. Furthermore, an RNN uses use a mechanism known as an LSTM — a particular kind of set of network layers. An LSTM allows the network to retain particular combinations of word frequencies in its internal state (or memory). It is not necessary to deeply delve into the details of how RNNs work: it is an involved subject to discuss. Simply put, they are attractive for our task because they are excellent for our task of identifying patterns in language.

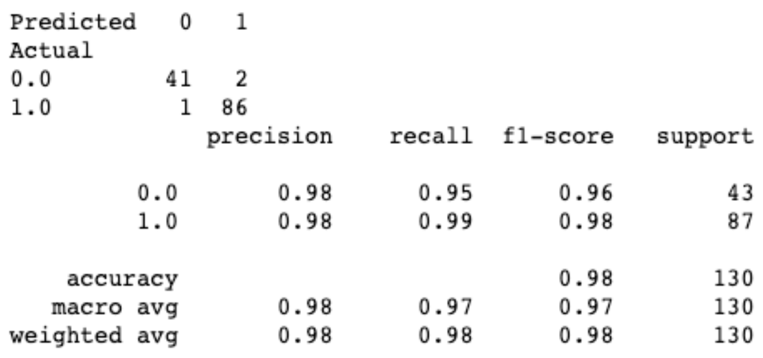
It is important to note, as an aside, that RNNs are *especially* useful when looking at sequential data: any data in which ordering matters. For example, an input such as "Paris is the capital of France" would be viewed differently as "France is the capital of Paris". Our data does not take into account the order in which the words are presented in a speech. If our data did include this, an RNN would be an substantially better model of choice. A real-world example of RNN use is the auto-suggestion feature found on computer text editors. The RNN reads the particular set of words, taking into account that particular permutation of words, and renders a prediction.

That said, our data does show *some* kind of ordering in words. Many of these records contain a 0 frequency for some words. This means that the word is not used at all in the speech. If we were to shrink our dataset to the attributes "people", "jobs", and "union" — in that order in our dataset — one can see that a record of [0, 0.5, 0.4] would be different from [0.5, 0.4, 0.3] in terms of ordering. One record would be ["jobs", "union"] and the other would be ["people", "jobs", "union"].

Discussion about the utility of an RNN aside, my model was implemented as follows. This is a conventional implementation for an RNN. It contains a mix of feedforward layers and LSTM layer sets.

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout, Masking, Embedding

model = Sequential()

model.add(LSTM(128, input_shape = (trainX.shape[1:]), activation = 'relu',
return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(64, activation = 'relu'))
model.add(Dropout(0.2))

model.add(Dense(32, activation = 'relu'))
model.add(Dropout(0.2))

model.add(Dense(10, activation = 'softmax'))

opt = tf.keras.optimizers.Adam(lr = 1e-3, decay = 1e-5)

model.compile(loss = 'sparse_categorical_crossentropy',
             optimizer = opt,
             metrics = ['accuracy'])
model.fit(trainX, trainY, epochs = 50)

rounded_predictions = model.predict_classes(testX, batch_size = 10, verbose
= 0)
```

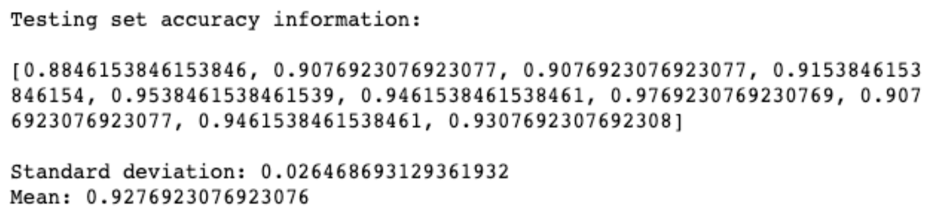
Given the feature rankings from the random forest, multiple kinds of datasets were attempted. The dataset that produced the strongest results had the lowest ranked 250 attributes of the *original* dataset (i.e. mostfreq100docword.csv: 431 rows x 1000 columns) dropped. The modified dataset I manually produced in the data pre-processing stage was good, but not as strong as this. The results are astounding. Below sums the strongest results of a single execution of the RNN. The results are presented in the same format as it was for random forest.



```
Predicted   0   1
Actual
0.0        41   2
1.0         1  86
              precision    recall  f1-score   support

         0.0       0.98      0.95      0.96        43
         1.0       0.98      0.99      0.98        87

    accuracy                           0.98       130
   macro avg       0.98      0.97      0.97       130
weighted avg       0.98      0.98      0.98       130
```

Figure 8: Strongest results obtained in a single execution of the RNN.

The results above indicate strong performance. Regarding verification of the model, I encountered several unresolvable issues in attempting to perform a 10-fold cross-validation in `Keras`. Thus, I manually ran the model ten times, each time generating a unique train-test split. Below is the resulting set of prediction accuracies on the test set.



```
Testing set accuracy information:

[0.8846153846153846, 0.9076923076923077, 0.9076923076923077, 0.9153846153
846154, 0.9538461538461539, 0.9461538461538461, 0.9769230769230769, 0.907
6923076923077, 0.9461538461538461, 0.9307692307692308]

Standard deviation: 0.026468693129361932
Mean: 0.9276923076923076
```

Figure 9: Results for RNN after cross-validation.

Assuming that SVMs do not perform better than this predictor, this may be our model of choice. It confirms our hypothesis that RNNs are strongly suited for this kind of data. Furthermore, our implementation of the random forest reduced the noise in the data drastically. When noise is minimized, neural networks are reputed to perform excellently. A neural network *generally* is good for speech data because it takes the subtle word-to-word interactions into account. With careful design, it can be the final model of choice. That may happen to be the case here.

Similar to my attempt of RNN, I attempted SVM with multiple datasets. Removing the lowest ranked 250 attributes provided strong success. My implementation for SVM is as follows.

```
from sklearn import svm

clf = svm.SVC()
clf.fit(trainX, trainY)

predY = clf.predict(testX)
```

Below sums the strongest results of a single execution of SVM.

```
Predicted  0.0  1.0
Actual
0.0         42    3
1.0          6   79
              precision    recall  f1-score   support

         0.0       0.88      0.93      0.90        45
         1.0       0.96      0.93      0.95        85

    accuracy                           0.93       130
   macro avg       0.92      0.93      0.92       130
weighted avg       0.93      0.93      0.93       130
```

Figure 10: Strongest results obtained in a single execution of the SVM.

This is good and expected, given that the dataset was optimized. The precision is for the loss (0) class is notably low. Only 88% of the total class 0 points were classified correctly. This result is not reflected in the other predictors. They achieved fairly high precisions. Thus, this may value be a consequence of using SVM.

I performed 10-fold cross-validation on the model. The results are below.

```
Testing set accuracy information:

[0.5747126436781609, 0.7906976744186046, 0.8488372093023255, 0.9186046511627907, 0.8023255813953488]

Standard Deviation: 0.11528927637693448
Mean: 0.787035551991446
Training set accuracy information:

[0.9418604651162791, 0.9130434782608695, 0.8956521739130435, 0.8927536231884058, 0.9188405797101449]

Standard Deviation: 0.017755446586175724
Mean: 0.9124300640377486
```

Figure 11: Results for SVM after cross-validation

It is clear that this model performs substantially more poorly than others. SVM would thus be considered a weak classifier. In a more extensive study, it would be useful to do SVM with bagging and SVM with boosting.

It is clear that our final model of choice is a recurrent neural network, using a dataset optimized by the random forest algorithm.

Payoff:

Our process was strong, and resulted in useful insights. In a more in-depth study, it would be useful to assess SVM with bagging and boosting. Some of the other weaker classifiers, such as Naive Bayes, can be assessed in an ensemble method. That said, it would be unlikely that these predictors would outperform our final model of choice.