Journal of Methods and Measurement in the Social Sciences,
Vol. 12, No. 2, 56-94, 2021

Binary Classification: An Introductory Machine Learning Tutorial for Social Scientists

Vivian P. Ta
Lake Forest College

Leonardo Carrico
Illinois Institute of Technology
Lake Forest College

Arthur Bousquet
Lake Forest College

A barrier that prevents many social scientists from pursuing big data research is the lack of technical training required to assemble and organize big data. In an effort to address this barrier, we provide an introductory tutorial into machine learning for social scientists by demonstrating the basic steps and fundamental concepts involved in binary classification. We first describe the data and libraries required for analysis. We then demonstrate data cleaning methods, feature engineering, the model-building process, model assessment, and feature importance. Last, we discuss the ways in which social scientists can use machine learning to complement inference-based approaches and how it can contribute to a richer understanding of social science.

Keywords: machine learning, big data, classification, tutorial

In the age of big data, researchers across many disciplines have begun to obtain massive amounts of data from a wide range of sources to assess behavioral patterns and make predictions. *Big data* refers to the "…large amount of data in the networked, digitized, and sensor-laden, information-driven world" (Chang & Grady, 2019). Although big data has been defined in several ways, it is typically characterized by large volumes of data that are generated at a high speed and come in a wide variety of formats (Kitchin & McArdle, 2016). This is starkly different from the structured, "small" sample data that has been historically ubiquitous in the social sciences (Florescu et al., 2014; Kitchin, 2015). The amount of published journal articles using big data has been increasing and there has been growing interest in machine learning and other methods used to analyze big data (Monaro et al., 2018; Sartori et al., 2019). Although this growth has been extensive in many fields, it has been slower in many areas of social science due to a number of technological, human, and organizational barriers (Lazer & Radford, 2017).

This is problematic for several reasons. A great deal of data is "now being produced, collected, and analyzed at unprecedented speed, breadth, depth, and scale" (Metzler et al., 2016) which may present social scientists with more opportunities to conduct large scale research on topics that were once considered exceptionally difficult and time-consuming to investigate. Much of big data are behavioral in nature and warrant the involvement of social scientists in data analysis. And, perhaps most importantly, big data has raised “a number of complex new social and ethical issues” (Kwok, 2019) that cannot be adequately addressed without insights from the social sciences.

One of the barriers that prevent many social scientists from being involved in big data research is the lack of technical training that is required for collecting, organizing, and analyzing big data, among others (Adjerid & Kelley, 2018). The lack of data science skills also precludes educators from teaching social science students the skills that are necessary to navigate big data. As such, researchers have not only called for greater interdisciplinary cooperation, but also greater accessibility and inclusion of data science skills into the social sciences.

Although a number of resources have been created to address this issue, they are not always easily accessible, inclusive, or appropriately adapted for social scientists. For instance, tutorials tend to assume the reader has coding skills, or do not address the components that social scientists are generally more interested in such as the interpretability and validity of models. In addition, classes may require funding for enrollment and workshops are often constrained around specific dates.

In an effort to address this barrier, we sought to provide a gentle, introductory tutorial for social scientists on machine learning—a method that is commonly used to analyze big data—in the current paper. Specifically, our tutorial focuses on the main concepts involved in machine learning and demonstrates a commonly-used machine learning technique: binary classification. This is a supervised learning approach which focuses on predicting the class or category (y) from inputs (x). Our goal in this tutorial is to predict whether a message was persuasive or not persuasive based on certain linguistic features of the message. As such, we will be building models that learn how the input variables (the linguistic features of each message) relate to the output (whether a message was persuasive or not), which is then used to build a predictive model that can accurately predict if a given message was persuasive or not persuasive. We also assess feature importance—the degree to which a particular input variable contributed to this predictive task.

Although binary classification is only one out of many machine learning techniques, we chose to provide a tutorial on binary classification for several reasons. Classification is a widely-used technique in machine learning that has many applications that are relevant to the social sciences such as document classification, sentiment analysis, fraud detection, and many more. It is a relatively straightforward analysis and lays the foundation for more complicated machine learning techniques, making it a suitable topic to introduce social scientists into machine learning. Moreover, determining how a set of input variables relate to a categorical output variable is an analytic strategy commonly used in the social sciences (e.g., logistic regression), making classification applicable and of interest to a broad range of social science researchers.

The Current Tutorial

The goal of this tutorial is to guide social scientists through the basic steps and fundamental concepts involved in machine learning. This tutorial is appropriate for social scientists who know very little about machine learning, as well as social scientists who are somewhat familiar with big data analytics and are looking to refine their understanding of machine learning fundamentals.

We begin by describing the data used in this tutorial and the libraries that are required for analysis. We then demonstrate data cleaning methods, feature engineering, the model-building process, model assessment, and feature importance. We end by discussing the ways in which machine learning complement the inference-based approaches that are commonly used in the social sciences and how it can contribute to a richer understanding of social science. Because it is impractical to go in-depth into every aspect of machine learning in one tutorial, resources and citations have been placed throughout the tutorial for additional guidance and information.

This tutorial is run and presented using *Scikit-learn* in Python version 3.7. Although proficiency in Python is helpful, it is not required for understanding or successfully completing this tutorial. The data and scripts used in this tutorial are provided on our Github repository[1] as well as in the Supplementary File. We will be referencing the corresponding files and code that are associated with each step of this tutorial throughout the paper. For helpful resources on Python and *Scikit-learn*, see Supplementary A.

The Data

Our data come from /r/ChangeMyView[2], a community on Reddit in which users post their views on any topic (original posters, or OPs) and invite others to debate them. People who debate the OP reply to the OP's original post (referred to as "repliers") and state the reasoning behind their beliefs. The OP will award a delta (Δ) to particular replies that changed their original views (see Supplementary B for examples). /r/ChangeMyView is a moderated community such that all users are required to adhere to its rules of conduct regarding post/reply quality, response timeliness, and civil discourse.

1 https://github.com/amac-lfc/CMV/tree/v1.1 or https://github.com/amac-lfc/CMV/archive/1.1.zip

2 https://www.reddit.com/r/changemyview/

The Python Reddit API Wrapper was used to pull the data (Boe, 2015). The initial dataset consisted of 4,820,112 replies from 104,081 original posts that were posted between 2013 and 2018. We also obtained each reply's metadata including whether a reply was persuasive or not (i.e., awarded a delta or not), which original post (parent post) a given reply came from, the author username of a given reply, and others (see *classifiers/main.py* for the list of all metadata). The raw data, as well as the processed data, can be downloaded from our Github repository[1] (see our note regarding the raw and processed datasets at the end of the *Feature Engineering*).

Libraries

A number of libraries must be installed into Python to execute all of the functions involved in this tutorial. These libraries include *Numpy*, *Pandas*, *Scikit-learn*, and *Imbalanced-learn*. They are included at the very beginning of each script for installation. See Supplementary C for more information about these libraries.

Data Cleaning

As one might expect, the data must be cleaned and assessed for quality before analysis. This typically involves determining if the data are accurate, corrupted, or missing. The corrupt or missing cases could be removed from the dataset entirely or be imputed or modified. Choosing which data cleaning technique(s) to use would depend on the research question at hand, the type of the data at hand, and individual data cleaning practices. See Kelleher, Mac Namee, & D'arcy, (2015) and McCallum (2012) for more information regarding data cleaning.

After assessing our data, it was determined that our data required the following (Supplementary E):

1. *Removing empty replies*: replies that show up as "[deleted]" or empty are typically the result of a user deleting their reply after posting it, a moderator deleting the reply after it was posted due to a violation of /r/ChangeMyView's rules, or a user deactivating their account sometime after posting the reply. We remove these replies given that no content was provided.
2. *Transforming special characters into their textual equivalent*: For example, "¾" might be transformed into "three out of four" and "÷" might be transformed into "divided by". These transformations allow algorithms to process symbols appropriately and with the correct meaning.
3. *Removing punctuation*: This allows all of the text to be processed uniformly and ensures that different forms of the same word are not treated separately (e.g., Hi.; Hi?; Hi!).

Feature Engineering

An important strategy for increasing the predictive accuracy of any machine learning model is through feature engineering. This involves creating new input features from the same existing raw data. Given that machine learning algorithms use these input features for learning, it is important to properly transform the data into a form that is compatible with a given machine learning algorithm. Many feature engineering techniques exist including *binning* (transforming a continuous or categorical variable into different groups/categories), *one-hot encoding* (transforming a categorical variable into multiple variables with binary values, such as 0 and 1, to denote the presence or absence of that variable for a given observation of data), *feature spitting* (extracting particular facets of a variable and representing it as a separate variable), and others. For additional information regarding feature engineering techniques, see Zheng and Casari (2018).

Deciding which feature engineering technique to apply on the data will depend on the kinds of information that should be highlighted for optimal learning and classification. In other words, features are engineered to allow algorithms to focus on and learn information that provides the most signal for learning. Having domain knowledge facilitates the identification and isolation of such information.

Because our goal was to assess if a reply was persuasive or not persuasive based on its linguistic features, we conducted feature engineering by extracting certain information from each reply. Specifically, we extracted the frequency with which terms from certain linguistic categories appeared in each reply. The linguistic categories chosen were ones that contribute to a message's persuasive appeal based on previous research. We also used previous research to construct dictionaries that contained terms that were relevant to each linguistic category.

In all, 25 features were engineered. These input features, plus a feature indicating whether a given reply was awarded a delta or not, comprise the dataset that will be used to train a machine learning model to classify whether a given reply was persuasive or not. More information about these features can be found in Supplementary D.

To obtain the cleaned dataset with all features engineered, the raw data needs to be downloaded and createData() needs to be run in *classifiers/main.py* (Supplementary E). It is important to note that this step can take more than a week to process and complete due to the sheer size of the data. Therefore, the processed data is also available to be downloaded directly 3 . The sections below explain the remaining steps written in *classifiers/main.py*.

Feature Scaling

Once feature engineering is complete, the next step is to scale the features given that they vary in units, range, and magnitude. This is similar to the process of standardizing variables in inference-based approaches. There are different methods for scaling features in machine learning. For instance, *z* score standardization can be applied to transform the mean to equal 0 and the standard deviation equal to 1. Or, normalization can be applied to transform values into a fixed range, typically between 0 and 1. The scaling method used depends on the problem at hand and the machine learning algorithm used. See Zheng and Casari (2018) for further information. In this tutorial, we normalized our features to a range between 0 and 1 (Supplementary F).

Imbalanced Classes

An issue with our data was the imbalanced number of replies that were not persuasive (non-delta-winning replies) and the number of replies that were persuasive (delta-winning replies). After the data were cleaned, there were 4,807,072 replies that were not persuasive and 13,040 replies that were persuasive. Having imbalanced classes can erroneously skew prediction accuracy, leading the model to predict the class that is simply more prevalent. In our case, our model could simply predict that any given reply is not persuasive and still obtain an overall prediction accuracy over 90%. Thus, not addressing class imbalances could produce a biased model with a high rate of false negatives and prevent researchers from building meaningful predictive models.

Common strategies for correcting class imbalances include *class weights, random down-sampling the majority class*, and r*andom up-sampling the minority class*. *Class weights* is a method that assigns different weights to the majority and minority classes and penalizes misclassification by increasing weight for the minority class (i.e., persuasive replies) and reducing weight for the majority class (i.e., replies that were not persuasive; see King and Zeng, 2001). *Random down-sampling the majority class* involves randomly omitting observations from the majority class to prevent it from over-influencing the learning algorithm. *Random up-sampling the minority class* involves randomly replicating observations from the minority class to reinforce its effect.

3 https://bit.ly/2AA6odD

The particular class imbalance strategy or strategies that is/are ultimately implemented should be appropriate for the data and research question at hand. Users may implement multiple strategies or compare model performance indices across different strategies to identify the strategy that produces the most optimal results. In the current tutorial, we implemented two class imbalance strategies (Supplementary F).

In the first strategy, we applied random down-sampling of the majority class by randomly omitting replies that were not persuasive to obtain a more comparable sample of non-persuasive replies (n = 20,000). We then implemented a slightly modified version of the random up-sampling the minority class technique: we increased the number of persuasive replies from 12,138 to 20,000 by using Synthetic Minority Oversampling Technique (SMOTE; Chawla, Bowyer, Hall, & Kegelmeyer, 2002) which combines the features of multiple persuasive replies to create new replies. Thus, the new additions are not simply copies of existing persuasive replies. SMOTE usually outperforms random up-sampling and improves prediction accuracy. In the second strategy, we implemented class weights to balance our classes. We compared model performance indices between these two strategies and assess if one strategy outperformed the other in the *Assessing Model Performance* section.

Preparing Data for Training and Testing

The next step involves partitioning the data for training and testing. An algorithm will use the *training data* to fit the model — that is, it will use the training data to learn how the various linguistic features (i.e., input variables) relate to the output (i.e., whether a reply was awarded a delta or not) in order to build a predictive model that predicts if a given reply was persuasive or not persuasive. The *test data* will be used to validate the model's performance on unseen data — a sample of the overall data that the model has not already assessed during the learning process. Thus, the test data is only used to evaluate model performance after a model is trained with the training data. It is important to use unseen data in testing as this will provide more reliable model performance estimates that are not inflated (James, Witten, Hastie, & Tibshirani, 2013). Several methods exist that can be used in this process and we will outline the methods that are commonly used below.

Bootstrapping involves taking random samples (with replacement) of equal size repeatedly from the training data. A model is fitted on each sample and model performance is conducted on the data that were not included in the given sample. This process is repeated for each bootstrapped sample and model performance metrics are aggregated across each iteration. This method can address model overfit, reduce variance, and improve model accuracy. A greater number of samples is more ideal but is more computationally expensive. See James et al. (2013) and Kuhn and Johnson (2013) for additional information.

The *validation set* approach (also known as *data splitting*) involves splitting the data into a training set and a test (holdout) set. Using the split function in *Scikit-learn*, researchers might apply a 70/30 split to assemble the training and test datasets. That is, 70% of the data would be randomly selected and used for training while the remaining 30% would be used for testing. Other ratio splits, such as 60/40 and 80/20, can also be used. Because there are no straightforward rules regarding which ratio split is the best to use, researchers must consider how much data is available, the type of model that is being trained, and the trade-offs associated with each ratio split to make this decision. For instance, having more training data provides more opportunities for an algorithm to build an accurate predictive model whereas having more test data allows researchers to better assess how well the model generalizes to other unseen data. If researchers are looking to fit a very complex model that needs a great deal of data to learn from, an 80/20 split might be more appropriate compared to a 70/30 or 60/40 split. Or, if researchers are prioritizing generalizability from their model, a 60/40 split might be more appropriate compared to a 70/30 or 80/20 split. Whatever ratio is used, it is important that the training and test datasets are both representative of the entire data and that the test dataset is large enough to produce statistically meaningful outcomes. See Raschka (2018) for additional information regarding ratio splits.

The data may also be split into training, validation, and test datasets (e.g., 70/15/15; 80/10/10; 60/20/20). The validation dataset is used to evaluate model performance from the training dataset while adjusting its hyperparameters to optimize performance. In other words, after training on the training dataset is complete, the researcher may evaluate the model on the validation dataset and tweak the model to improve its performance. The tweaked model is then evaluated on the test dataset. See Kuhn and Johnson (2013) for additional information.

The validation set approach is simpler and less expensive on resources compared to other methods. However, it can yield variable model performance metrics depending on which data points are included in the training and test sets. If the training set is small, this method is also prone to overestimation of model performance metrics.

The *k-fold cross-validation* (CV) approach addresses the limitations of the validation set approach. It involves splitting the data into k equal-sized samples (or folds). One sample is used for testing the model and the remaining k - 1 samples are used for training the model. This process can be repeated k number of times with each sample serving as the test set. The resulting model performance metrics from each iteration are then aggregated. This method tends to yield more accurate, less inflated, and less variable test error estimates. Because of this, it is typically recommended when dealing with small datasets (see James et al., 2013 for additional information). In addition, this method is computationally less expensive compared to leave-one-out cross-validation. A value of 5 or 10 are commonly used for *k* as these values tend to yield estimates that are not too high in bias or variance (see Kuhn & Johnson, 2013 for additional information).

In the *leave-one-out cross-validation* approach, one data point is used for validation while the remaining *n* - 1 data points are used for training. A prediction is made for the single data point and this process is repeated *n* times. In other words, every data point is used for validation, and model performance metrics are aggregated. Training sets are almost identical in size compared to the entire dataset and results from each iteration are similar. This method is less biased compared to *k*-fold CV and is less prone to overestimating model performance metrics. However, it can be very time consuming with very large datasets given that *n* models must be fitted. LOOCV also produces test error estimates that have higher variance than *k*-fold CV. See Sammut and Webb (2010) for additional information.

Readers should consider the strengths and limitations of each method, along with logistical factors such as time availability and the size of the data, to determine which approach to use. Due to the size of our data, we conduct and demonstrate 5-fold cross-validation in this tutorial. Feature scaling, SMOTE, and class weights are executed in *classifiers/main.py* (Supplementary F).

Model Building

The next step involves building a model by selecting an algorithm to assess the training dataset for the learning/training process. The output of the training process is a model that has learned how the input and output variables relate to one another to predict if a reply was persuasive or not. This model is then tested on the test dataset in order to assess its performance. This process echoes the mechanisms involved in associative learning (or conditioning) in which learned behaviors are a result of reinforced stimuli and responses (Shanks, 1995). In fact, the concept of reinforcement learning—a common training approach in machine learning that is based on the maximization of reward (Sugiyama, 2015)—is rooted in B.F. Skinner's theory of operant conditioning: a learning process in which a behavior is modified through the use of reinforcement or punishment (Staddon & Cerutti, 2003).

Typically, several models are built using different algorithms, and model performance metrics are reported and compared with one another. The algorithms selected for testing should be appropriate for the data and task at hand. In other words, given that each algorithm has their respective assumptions, strengths, and weaknesses, the researcher should systematically select algorithms to test. See Brownlee (2019) and Microsoft's guide4 for more information regarding various algorithms and algorithm selection.

We built 8 different models using the following algorithms that were most appropriate for classification: *Decision Tree*, *Random Forest*, *Gradient Boosting*, *Gaussian Naive Bayes*, *Bernoulli Naive Bayes*, *Support Vector Machine* (SVM), *Ada Boost*, and *Logistic Regression*. In the interest of space, we will not delve into each algorithm's framework, strengths, and weaknesses in the current tutorial but encourage readers to refer to the citations provided for each algorithm in Supplementary G, along with *Scikit-learn's* list of classification algorithms5 for additional information. Model building is executed in *classifiers/main.py* and all 8 models are defined in *classifiers/models.py* lines 84-95 (Supplementary G).

Hyperparameter Optimization

Before the training process begins, the *hyperparameters* of each model must be tuned to minimize error, help estimate model parameters, and optimize its learning process. Hyperparameters are configurations of a model that influence the quality and speed of a model's learning process. These hyperparameters cannot be estimated ahead of time from data and thus require manual tuning. Examples of hyperparameters include the number of decision trees that will be constructed and the depth of each tree in a random forest model, and penalty in a logistic regression model.

Each algorithm will have its own set of hyperparameters to tune and the list of hyperparameters for a given algorithm can be found on the *Scikit-learn* website. However, the optimal value for a given hyperparameter or set of hyperparameters in one problem or dataset may not be the same in a different problem or dataset. Users may be tempted to use the practical hyperparameter values that are provided by default for all models in *Scikit-learn*, but practical does not always translate to optimal (Buitinck et al., 2013). How, then, does one identify the best value or set of values for a given hyperparameter or set of hyperparameters, respectively?

A common method is performing a *grid search*. In short, a grid search conducts an exhaustive search on every combination of a model's hyperparameter values to identify the most optimal set of hyperparameters. We conducted a grid search on all 8 models (Table 1) and then used these hyperparameters for each of our models in training and testing. The code for the grid search can be found in *classifiers/grid_search.py*.

4 https://docs.microsoft.com/en-us/azure/machine-learning/how-to-select-algorithms

5 https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

Table 1
Results of grid search

Logistic		SVM	
C	0.001	degree	1
Penalty	l2	Gamma	0.1
solver	liblinear	kernel	’rbf’
		shrinking	True
ADA Boost		**Gradient Boosting**	
algorithm	’SAMME.R’	learning_rate	0.1
learning_rate	0.5	loss	’deviance’
n_estimator	1600	max_features	’sqrt’
k_neighbors	4	max_leaf_nodes	None
sampling_strategy	1.0	max_samples_leaf	1
		max_samples_split	10
		n_estimator	200
Decision Tree		**Random Forest**	
criterion	’gini’	criterion	’gini’
max_depth	10	max_depth	None
max_features	None	max_features	’sqrt’
max_leaf_nodes	50	max_leaf_nodes	None
min_impurity_ decrease	0.0	in_impurity_ decrease	0.0
max_samples_leaf	1	max_samples_leaf	4
max_samples_split	5	ax_samples_split	2
splitter	’best’	boostrap	True
Gaussian Naive Bayes		**Bernouilli Naive Bayes**	
var_smoothing	0.1	alpha	0.001
priors	[0.02, 0.98]	class_priors	[0.88, 0.12]
		fit_prior	True

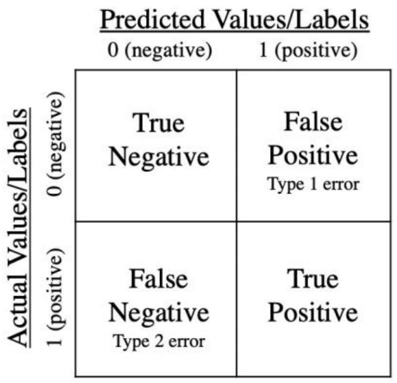
Assessing Model Performance

Each model’s performance can be evaluated using various model performance metrics which will aid in the selection of the best model. We describe the most commonly used metrics below. Selecting or prioritizing particular metrics for evaluation will depend on factors such as the type of model being used, the strengths and limitations of a particular metric, and the overarching goal of the project (see Murphy, 2012 for more information regarding model selection).

Confusion Matrix

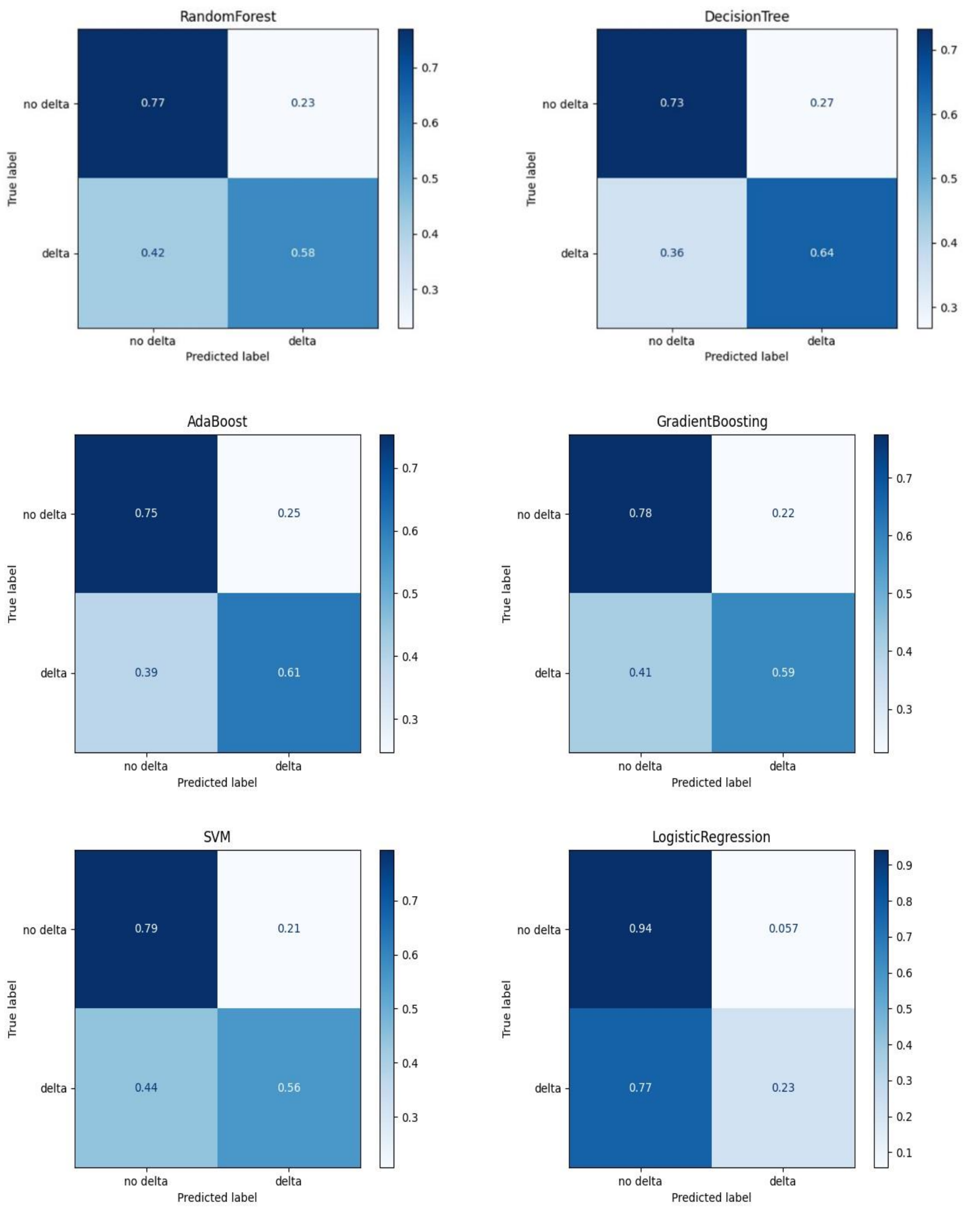
Figure 1 illustrates our *confusion matrix*. The y-axis marks the actual results — whether a reply was awarded a delta (1; the positive label) or not (0; the negative label) — and the x-axis marks the predicted results — the model's predictions of whether a given reply was awarded a delta (1; the positive label) or not (0; the negative label). Thus, the matrix reports a given model's *true positive rate,* which is the proportion of cases in which actual positives were predicted as positives; *true negative rate,* which is the proportion of cases in which actual negatives were predicted as negatives; *false positive rates* (Type I error), which is the proportion of cases in which actual negatives were predicted as positive; and *false negative rates* (Type II error)*,* which is the proportion of cases in which actual positives were predicted as negative. Formulas for each of these rates are reported in Supplementary H.

Figure 1. Confusion matrix components. 1 = delta awarded; 0 = no delta awarded.



With regard to our models, true positives represent the rate in which the model predicted that a reply was awarded a delta when it actually had been awarded a delta; true negatives represent the rate in which the model predicted that a reply was not awarded a delta when it actually had not been awarded a delta; false positives represent the rate in which the model predicted that a reply was awarded a delta when it actually had not been awarded a delta; and false negatives represent the rate in which the model predicted that a reply was not awarded a delta when it actually had been awarded a delta. Figure 2 illustrates the confusion matrices for all 8 models using SMOTE whereas Figure 3 illustrates the confusion matrices for all 8 models using class weights (Supplementary I).:

Figure 2. Confusion matrices for all 8 models using SMOTE.



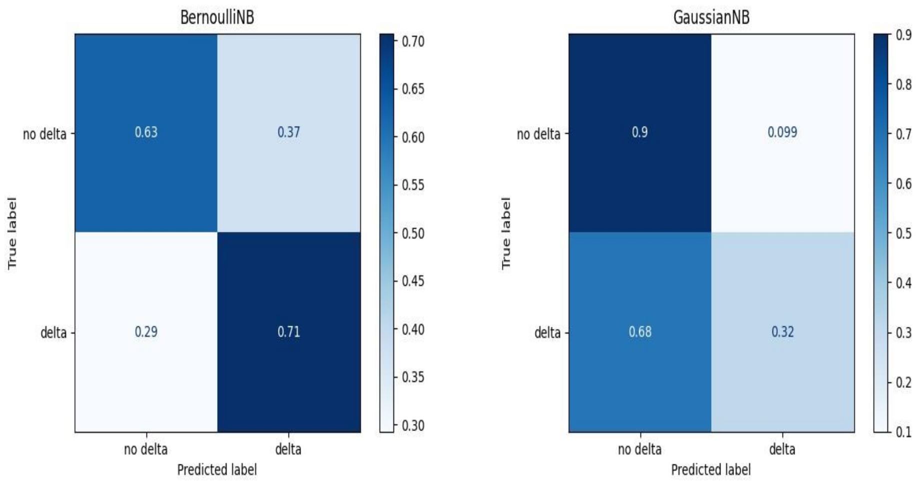
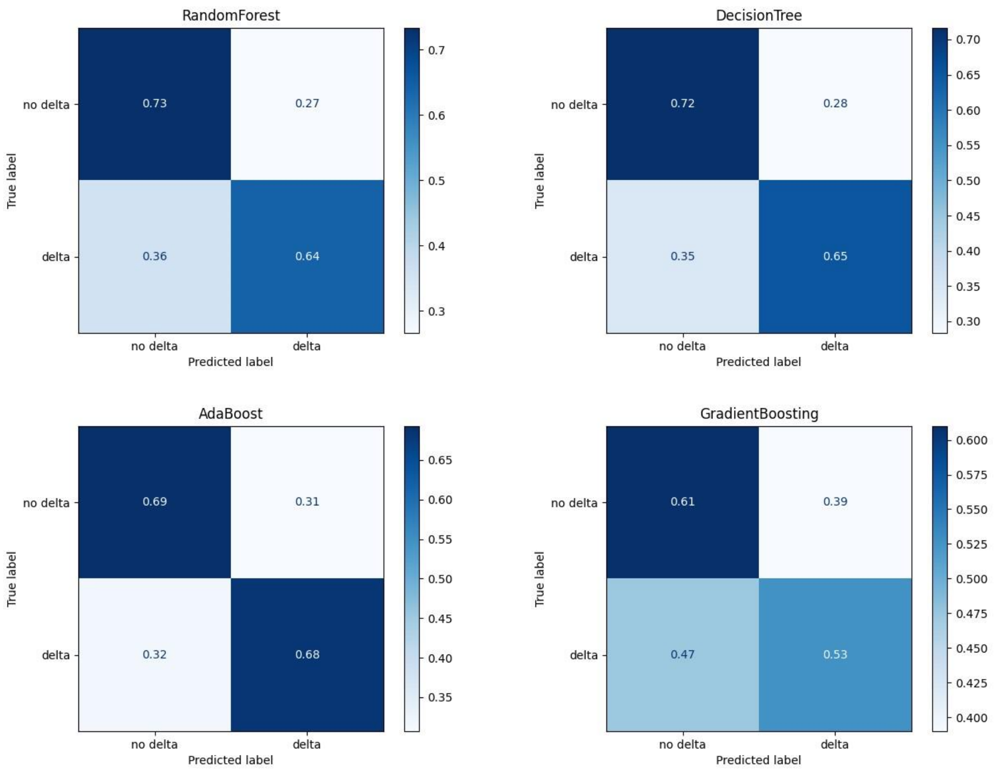
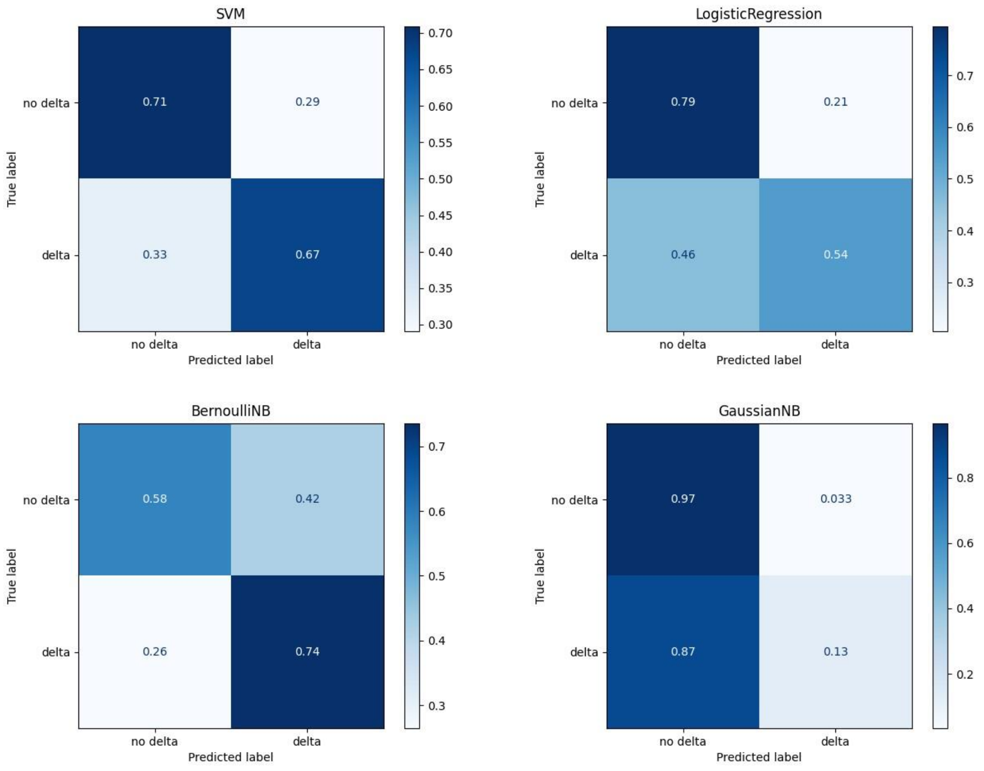


Figure 3. Confusion matrices for all 8 models using weighted classes.





Additional Model Performance Metrics

Other model performance metrics can also be obtained from a confusion matrix to further assess a model's performance. We describe metrics that are commonly used below and formulas can be found in Supplementary H.

Accuracy is the accuracy rate of an overall model which is the proportion of correct predictions made. It is often used in classification and is an effective performance metric when classes are equal in size. *Precision* is the proportion of positive predictions that are actually positive and is an effective metric when there are class imbalances as it calculates the accuracy of the minority class.

Recall/Sensitivity is the proportion of actual positives that were correctly identified as such. Unlike precision, recall/sensitivity illustrates the rate of positive predictions that are missed. It is also an effective metric when there are class imbalances. When used with precision, true negatives are never taken into account. Thus, this should only be used when the detection of the negative class is not important.

Specificity is the proportion of actual negatives that were correctly identified as such. Thus, it measures how well your model identifies negative cases. When used with recall/sensitivity, all entries in the confusion matrix are taken into account which is an advantage when the rate of true positives, true negatives, false positives, and false negatives should all be assessed.

F1 Score is the harmonic mean of precision and recall/sensitivity. The harmonic mean is often used as a measure of central tendency for ratios or rates as it equalizes the weights of each observation. As such, F1 Scores are an effective performance metric when there is a high rate of true negatives, when false negatives and false positives equally impact a model, and the addition of more data does not impact model outcomes.

In an *Area Under the Receiver Operator Characteristic Curve (AUC-ROC Curve)*, the ROC curve graphs the true positive rate and false positive rate at different classification thresholds. The AUC measures the entire area under the ROC curve and reports the probability that a given model ranks a random positive observation more highly than a random negative observation. In short, the AUC is an aggregate measure of a model's performance across all classification thresholds and tells us how well a given model discriminates between two classes. AUC scores range from 0 to 1 with scores closer to 1 indicating more accurate predictions. An advantage of the AUC is that it focuses on how well the predictions are ranked instead of the actual prediction itself. However, it treats false positives and false negatives equally and thus does not take into account the costs between these two types of error.

Precision-Recall (PR) Curve plots a model's precision and recall on the y- and x- axis, respectively, for different thresholds. Because it does not use the number of true negatives, it is advantageous for class-imbalanced datasets.

These various model performance metrics can be examined and compared with each other to aid in the selection of the best model. But how, exactly, does one choose the best model(s)? There is no straightforward answer. The best model(s) will depend on the nature of the data, the various problems and goals at hand, and the time and resources available. In this tutorial, since we are prioritizing classification accuracy, we will primarily focus on choosing the model with the best accuracy. Other model performance metrics, however, can be prioritized or taken into consideration depending on the overarching goal of the project. See Murphy (2012) and Seliya et al. (2009) for more information regarding model selection and classifier performance metrics.

We used 5-fold CV to calculate accuracy scores using class weights (Figure 4) and SMOTE (Figure 5; Supplementary I). Next, we compare model performance across the two class imbalance strategies and examine if a particular class imbalance strategy outperformed the other.

Figure 4. Accuracy scores of each model using 5-fold cross validation using class weights.

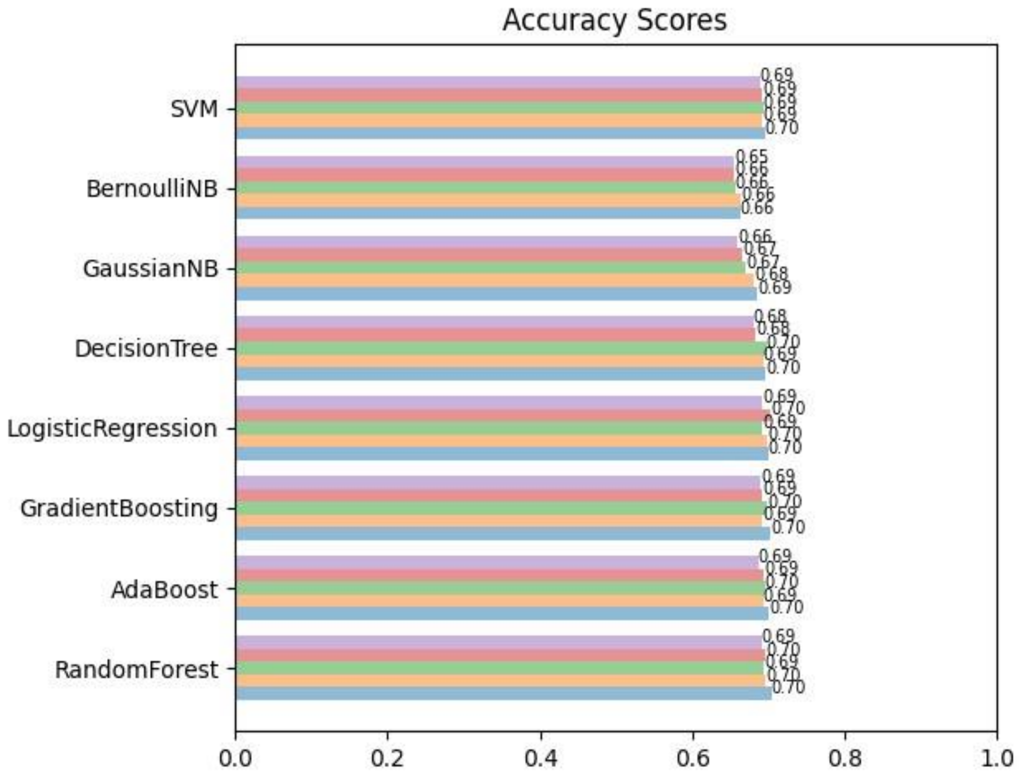
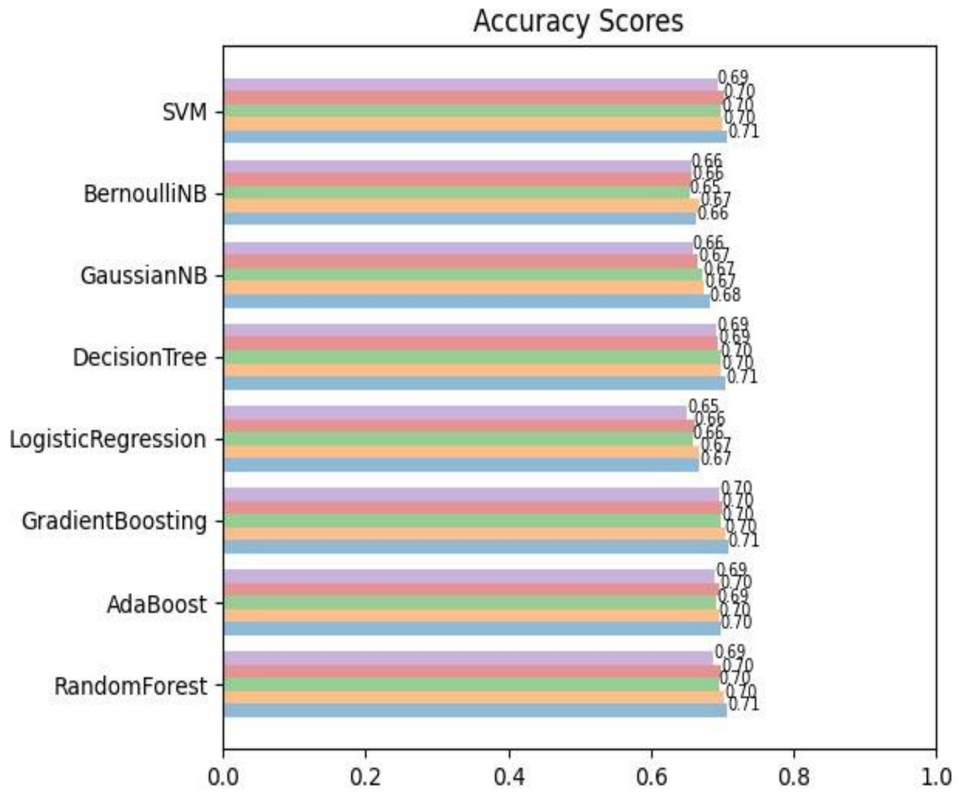


Figure 5. Accuracy scores of each model using 5-fold cross validation using SMOTE.



The accuracy scores across the two class imbalance strategies ranged from .65 to .71. The Logistic Regression and Random Forest models produced the highest aggregate accuracy score when using class weights (69%) whereas the Gradient Boosting model produced the highest aggregate accuracy score when using SMOTE (70.2%). At this point, it may be tempting to automatically select the Gradient Boosting model to report or use for subsequent classification tasks. However, testing many models and selecting only the model with the best accuracy score to use and report without taking into account the accuracy of the other models is referred to as *model-hacking*. This is similar to the concept of p-hacking in inference-based approaches and can lead to biased and flawed conclusions.

Two methods are typically recommended to avoid model-hacking: The first involves comparing all model outcomes with one another. Observing similar performance outcomes across all models would suggest that such results are reliable and robust, especially if similar metrics were obtained using different algorithms. The second involves combining several different classifiers into an ensemble model which typically performs better than single models. For more information regarding ensemble methods, see Brown, 2010.

Looking across both class imbalance strategies, the lowest and highest accuracy scores only differed by 6 percentage points, indicating that all of our models produced similar accuracy scores. As such, if we were to select one model to use for subsequent classification tasks, we should be fairly confident choosing the Gradient Boosting model using SMOTE as it produced the highest accuracy score.

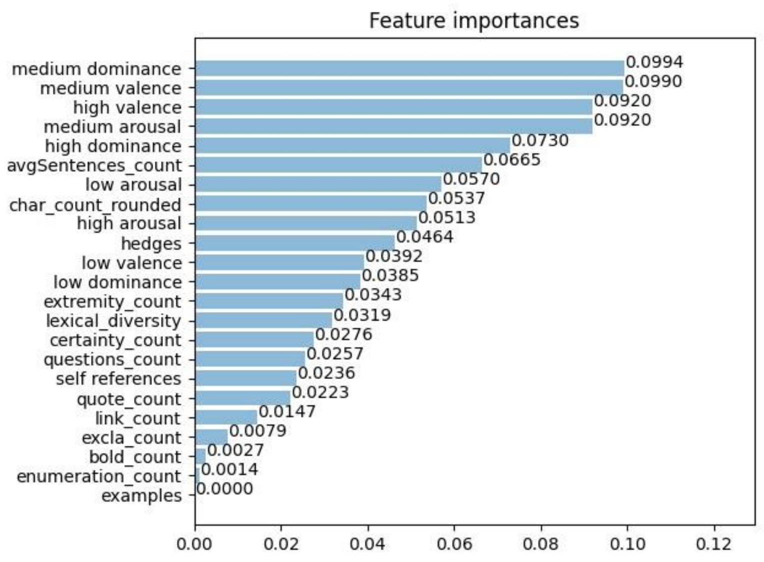
What if accuracy scores greater than 70.2% are desired? Several procedures may help increase accuracy by optimizing learning and classification. Examples include the use of different feature engineering strategies, adding or removing input variables, and implementing ensemble methods. See Patterson and Gibson (2017) for various procedures that can help increase accuracy scores. The code to train and test each model, compute confusion matrices, and compute accuracy scores is in Supplementary I.

Feature Importance

Social scientists may naturally be interested in identifying the input variables that were most important in the prediction process for a given model. In machine learning, this can be done by investigating *feature importance*. In addition to providing useful information about a model, examining feature importance can help simplify the model, enhance model interpretability, improve model performance, mitigate overfitting, and decrease the computational cost of the training process. For the sake of brevity, we generate feature importance only for the Random Forest model using SMOTE (Supplementary J). However, it is beneficial to compare feature importance among all models. Again, observing similar patterns among input variables across all models would indicate greater reliability and confidence in a given input variable's importance and role in classification.

Results are illustrated in Figure 6 with higher numbers indicating greater importance. The top 5 most important features in this model appear to consist of terms related to dominance, arousal, and valence. If the same pattern appears in the other models, this may suggest that certain levels of emotionality may substantially impact what makes a message persuasive. On the other hand, examples and formatting features such as the number of bullet points, bolded words, and exclamation points, do not appear to substantially impact what makes a message persuasive.

Figure 6. Feature ranking of the Random Forest model using SMOTE.



We measured feature importance via Entropy importance (Breiman, 2001). Other techniques can also be used to generate feature importance such as *Local Interpretable Model-Agnostic Explanations* (LIME; Ribeiro, Singh, & Guestrin, 2016), *InterpretML* (Nori, Jenkins, Koch, & Caruana, 2019), and *Shapley Additive Explanations* (SHAP; Lundberg & Lee, 2017). For more information regarding feature importance and model interpretablility, see Kaur et al. (2020).

Discussion

In the current paper, we provided an introductory tutorial into machine learning for social scientists by demonstrating the basic steps and fundamental concepts involved in binary classification. We built several predictive models that predicted whether a textual reply was persuasive or not persuasive. Our accuracy scores ranged from 65% to 70.2% and we identified features that played a larger role in this prediction task.

Social scientists tend to adhere to a more theory-driven approach to analyzing data and, as such, primarily use statistical models to make inferences regarding the relationships between variables. In contrast, big data research typically utilizes techniques that emphasize predictive accuracy such as machine learning. Put another way, inference-focused approaches seek to explain while prediction-focused approaches seek to predict (Bzdok, Altman, & Krzywinski, 2018). Incorporating prediction-focused approaches into social science research can complement inference-focused approaches, and, together, these two approaches can provide for a stronger and well-rounded understanding of social science.

For example, given that model performance tends to be better in training than in testing, this provides a level of protection against reporting more inflated and overly-optimistic estimations (Yarkoni & Westfall, 2017). P-hacking can be minimized through the collective examination of performance among all models that were tested, as well as the use of ensemble methods. In addition, the general emphasis on predictive accuracy enables the "general-purpose learning algorithms to find patterns in often rich and unwieldy data (Bzdok et al., 2018)." And, evaluating both inference and prediction can provide insightful conclusions that cannot be garnered using inference or prediction alone. In our case, researchers can determine which linguistic features make a message persuasive *and* also apply this to predict whether a message will be persuasive or not.

In prediction-focused approaches, the standard model building procedure involves building the model first and then testing its performance on unseen data using test datasets, cross-validation, etc. Thus, the process of examining how well the model generalizes and replicates onto new data are at least partially addressed and "built-in". Generalizability and replication in inference-based approaches, on the other hand, are addressed when researchers test the same statistical model on newly collected data. This process can take years to accomplish — but only if it is even being done in the first place. Thus, prediction-focused approaches can contribute to "maximize accuracy and minimize replicability issues" (Orrù, Monaro, Conversano, Gemignani, & Sartori,, 2020).

In addition to fostering greater involvement of social scientists in the world of big data, knowledge of machine learning techniques can better enable social scientists to facilitate the translation of empirical findings into real-world applications and interventions. For example, we created an app6 based on the analyses that were conducted in the current tutorial. In this app, users are prompted to input any message into the textbox. The machine learning model that we trained in this tutorial then analyzes the user's message and reports the probability of that message being persuasive. It also illustrates how the given message compares with the average persuasive message. Along with practical uses, this can help cultivate engagement and enhance the accessibility of social science research with the general public. Taken together, this can have important implications in shaping laypeople's perceptions and support for science (Rose, Markowitz, & Brossard, 2020) as well as bolster the role of social science in big data.

6 http://cmvcheck.amac.xyz/

Author Notes. The authors would like to thank Liam Connell and Enrique Treviño for their support and feedback. Funding was provided by the James Rocco Quantitative Data Research Scholarship.

References

Adjerid, I., & Kelley, K. (2018). Big data in psychology: A framework for research advancement. *American Psychologist, 73*, 899–917.

Boe, B. (2015). PRAW: The Python Reddit API Wrapper. *URL* https://praw.readthedocs.io/en/latest

Breiman, L. (2001). Random forests. *Machine Learning, 45*, 5–32.

Brown, G. (2010). Ensemble learning. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of machine learning* (pp. 312–320). Boston, MA, Springer US.

Brownlee, J. (2019). A tour of machine learning algorithms. *URL https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/*

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., . . . Varoquaux, G. (2013). API design for machine learning software: Experiences from the scikit-learn project. arXiv preprint arXiv:1309.0238.

Bzdok, D., Altman, N., & Krzywinski, M. (2018). *Points of significance: Statistics versus machine learning*. Nature Publishing Group.

Chang, W. & Grady, N. (2019). NIST Big Data Interoperability Framework: Vol. 1, Definitions, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, https://doi.org/10.6028/NIST.SP.1500-1r2

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research, 16*, 321–357.

Florescu, D., Karlberg, M., Reis, F., Del Castillo, P. R., Skaliotis, M., & Wirthmann, A. (2014, June). Will 'big data' transform official statistics? In European Conference on Quality in Statistics, Vienna (pp. 2-5).

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112). Springer.

Kaur, H., Nori, H., Jenkins, S., Caruana, R., Wallach, H., & Wortman Vaughan, J. (2020). Interpreting interpretability: Understanding data scientists' use of interpretability tools for machine learning, In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, Honolulu, HI, USA, Association for Computing Machinery.

Kelleher, J. D., Mac Namee, B., & D'arcy, A. (2015). *Fundamentals of machine learning for predictive data analytics: Algorithms, worked examples, and case studies*. MIT press.

King, G., & Zeng, L. (2001). Logistic regression in rare events data. *Political Analysis, 9*, 137–163.

Kitchin, R. (2015). The opportunities, challenges and risks of big data for official statistics. *Statistical Journal of the IAOS, 31*, 471-481.

Kitchin, R., & McArdle, G. (2016). What makes Big Data, Big Data? Exploring the ontological characteristics of 26 datasets. *Big Data & Society, 3*. https://doi.org/10.1177/2053951716631130

Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling* (Vol. 26). Springer.

Kwok, R. (2019). AI and the social sciences used to talk more. Now they've drifted apart. *KelloggInsight*. *URL* https://insight.kellogg.northwestern.edu

Lazer, D., & Radford, J. (2017). Data ex machina: Introduction to big data. *Annual Review of Sociology, 43*, 19-39.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions, In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Long Beach, California, USA, Curran Associates Inc.

McCallum, Q. E. (2012). *Bad data handbook: Cleaning up the data so you can get back to work*. O'Reilly Media, Inc.

Metzler, K., Kim, D. A., Allum, N., & Denman, A. (2016). Who is doing computational social science? Trends in big data research (White paper). London, UK: SAGE Publishing. https://doi.org/10.4135/wp160926.

Monaro, M., Galante, C., Spolaor, R., Li, Q. Q., Gamberini, L., Conti, M., & Sartori, G. (2018). Covert lie detection using keyboard dynamics. *Scientific Reports, 8*, 1–10.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT press.

Nori, H., Jenkins, S., Koch, P., & Caruana, R. (2019). InterpretML: A unified framework for machine learning interpretability. arXiv preprint arXiv:1909.09223.

Orrù, G., Monaro, M., Conversano, C., Gemignani, A., & Sartori, G. (2020). Machine learning in psychometrics and psychological research. *Frontiers in Psychology, 10,* 2970.

Patterson, J., & Gibson, A. (2017). *Deep learning: A practitioner's approach*. O'Reilly Media, Inc.

Raschka, S. (2018). Model evaluation, model selection, and algorithm selection in machine learning. arXiv preprint arXiv:1811.12808.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier, In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, USA, Association for Computing Machinery.

Rose, K. M., Markowitz, E. M., & Brossard, D. (2020). Scientists' incentives and attitudes toward public communication. *Proceedings of the National Academy of Sciences, 117*, 1274-1276.

Sammut, C., & Webb, G. I. (2010). Leave-one-out cross-validation. *Encyclopedia of Machine Learning,* 600-601. Springer.

Sartori, G., Pace, G., Orrù, G., Monaro, M., Gnoato, F., Vitaliani, R., Boone, K. B., & Gemignani, A. (2019). Malingering detection of cognitive impairment with the b test is boosted using machine learning. *Frontiers in Psychology, 10*, 1650.

Seliya, N., Khoshgoftaar, T. M., & Van Hulse, J. (2009). A study on the relationships of classifier performance metrics. In *21st IEEE International Conference on Tools with Artificial Intelligence (pp. 59-66)*, IEEE.

Shanks, D.R. (1995). *The psychology of associative learning*. Cambridge University Press.

Staddon, J.E., & Cerutti, D.T. (2003). Operant conditioning. *Annual Review of Psychology, 54*, 115-144.

Sugiyama, M. (2015). *Statistical reinforcement learning: modern machine learning approaches*. CRC Press.

Yarkoni, T., & Westfall, J. (2017). Choosing prediction over explanation in psychology: Lessons from machine learning. *Perspectives on Psychological Science, 12*, 1100–1122.

Zheng, A., & Casari, A. (2018). *Feature engineering for machine learning: Principles and techniques for data scientists*. O'Reilly Media, Inc.

Supplementary A

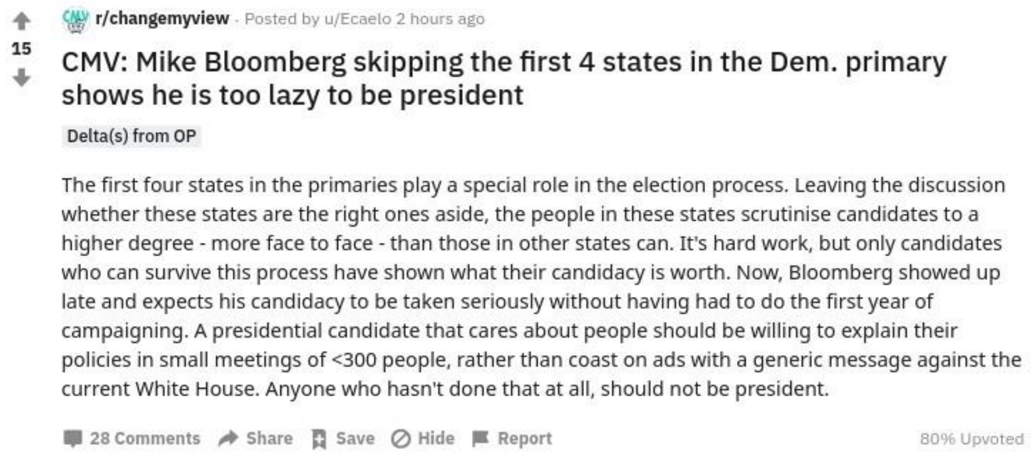
Python is a preferred programming language for machine learning as it possesses a great deal of pre-existing libraries that are available for researchers to quickly and efficiently build and run machine learning models. It is also well-equipped to handle large amounts of data, is open-source, and considered relatively simple and easy to learn compared to other programming languages. In addition, *Scikit-learn* is considered the most comprehensive machine learning package in Python and widely used by data scientists (see Hao & Ho, 2019, for a review of the *Scikit-learn* package). For helpful Python-related resources including installation, see Rhoads, 2019.

Hao, J., & Ho, T. K. (2019). Machine learning made easy: A review of scikit-learn package in python programming language. *Journal of Educational and Behavioral Statistics, 44*, 348–361.

Rhoads, S. (2019). A brief introduction to python for psychological science research. *American Psychological Association. URL:* https://www.apa.org/science/about/psa/2019/07/python-research

Supplementary B

An example of an original post on /r/ChangeMyView.



r/changemyview · Posted by u/Ecaelo 2 hours ago

15

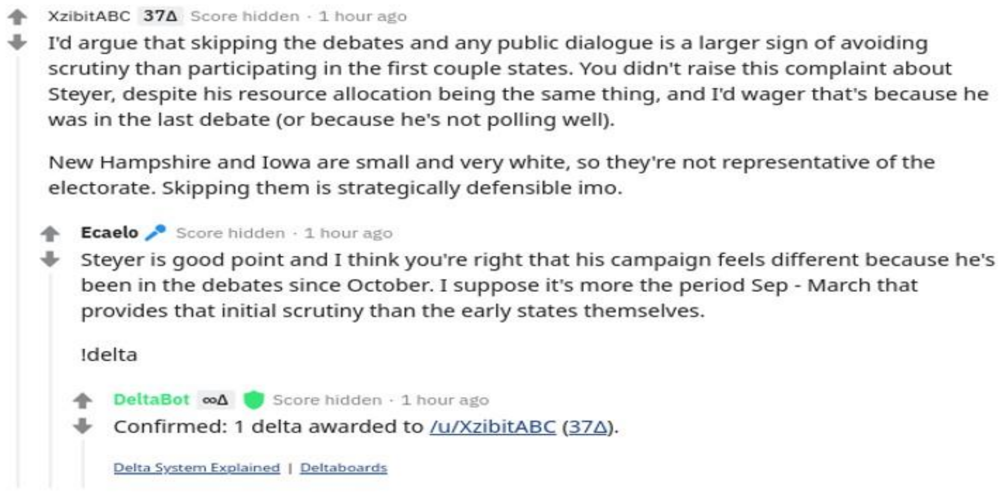
CMV: Mike Bloomberg skipping the first 4 states in the Dem. primary shows he is too lazy to be president

Delta(s) from OP

The first four states in the primaries play a special role in the election process. Leaving the discussion whether these states are the right ones aside, the people in these states scrutinise candidates to a higher degree - more face to face - than those in other states can. It's hard work, but only candidates who can survive this process have shown what their candidacy is worth. Now, Bloomberg showed up late and expects his candidacy to be taken seriously without having had to do the first year of campaigning. A presidential candidate that cares about people should be willing to explain their policies in small meetings of <300 people, rather than coast on ads with a generic message against the current White House. Anyone who hasn't done that at all, should not be president.

28 Comments Share Save Hide Report 80% Upvoted

An example of a delta being awarded to a reply.



XzibitABC 37∆ Score hidden · 1 hour ago

I'd argue that skipping the debates and any public dialogue is a larger sign of avoiding scrutiny than participating in the first couple states. You didn't raise this complaint about Steyer, despite his resource allocation being the same thing, and I'd wager that's because he was in the last debate (or because he's not polling well).

New Hampshire and Iowa are small and very white, so they're not representative of the electorate. Skipping them is strategically defensible imo.

Ecaelo Score hidden · 1 hour ago

Steyer is good point and I think you're right that his campaign feels different because he's been in the debates since October. I suppose it's more the period Sep - March that provides that initial scrutiny than the early states themselves.

!delta

DeltaBot ∞∆ Score hidden · 1 hour ago

Confirmed: 1 delta awarded to /u/XzibitABC (37∆).

Delta System Explained | Deltaboards

Supplementary C

Numpy (Oliphant, 2006; Walt et al., 2011) is used for working with arrays and matrices, as well as performing mathematical operations on them. This is used in the feature engineering stage to create input features.

Pandas (McKinney, 2010) is a software library used mainly for data manipulation and analysis. It is used to import the data (which are contained in CSV files) into Python, as well as to view, inspect, select, filter, sort, join, combine, and clean data.

Scikit-learn (Pedregosa et al., 2011) is used to build the machine learning models. It takes the array features from NumPy and turns them into prediction functions that returns a probability of whether a reply would receive a delta or no delta.

Imbalanced-learn (Lemaître et al., 2017) offers re-sampling techniques that are used to correct for class imbalances in the data.

Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research, 18*, 1–5.

McKinney, W. (2010). Data structures for statistical computing in Python, In *Proceedings of the 9th Python in Science Confere*nce, SciPy.

Oliphant, T. E. (2006). *A guide to Numpy (Vol. 1).* Trelgol Publishing USA.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Et al. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research, 12,* 2825–2830.

Walt, S.V.D., Colbert, S. C., & Varoquaux, G. (2011). The Numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering, 13*, 22–30.

Supplementary D

- *High arousal*: Terms that are “exciting” or elicit high arousal (e.g., abduct, devil, lunatic, stalker)
- *Medium arousal*: Terms that are moderately arousing (e.g., forget, moldy, scorch, unlucky)
- *Low arousal*: Terms that are “calm” or elicit low arousal (e.g., opt, retain, stall, west)
- *High valence*: Terms that are positive (e.g., achieve, consent, friend, marry)
- *Medium valence*: Terms that are neutral (e.g.,catwalk, dusk, retina, wavy)
- *Low valence*: Terms that are negative (e.g., syringe, taunt, prison, kill)
- *High dominance*: Terms that represent high degrees of control (e.g., care, independent, mindful, rise)
- *Medium dominance*: Terms that represent moderate degrees of control (e.g., flea, childless, analyst, cane)
- *Low dominance*: Terms that represent low degrees of control (e.g., mournful, adrift, abandon, dementia)
- *Certainty*: Terms denoting certainty/assertiveness (e.g., truly, unquestionably, precisely, undeniably)
- *Extremity*: Terms denoting how strongly an attitude or judgment is (e.g., extremely, very, seriously, strongly)
- *Examples*: Providing an example (for example, for instance, i.e., e.g.)
- *Hedges*: Terms/phrases used to soften a message (e.g., slight chance, suggests, some possibility, usually not, seems to me)
- *Self-references*: Terms referring to oneself (e.g., I, me, we, us, mine, myself, our)
- *Lexical diversity*: Type token ratio (number of unique words divided by total number of words)
- *Character count*: Total number of characters
- *Reply frequency*: Number of replies spawned from a given reply
- *Nested count*: Number of comments that preceded a given reply
- *Link count*: Number of links provided • Quotes: Number of quotes included
- *Questions*: Number of question marks
- *Bold*: Number of boldface terms

- *Average words per sentence*: Average number of words per sentence
- *Enumeration*: Number of bullet points
- *Exclamation*: Number of exclamation points

Supplementary E

```
def createData():
        inputs = ["/home/shared/CMV/RawData/Comments_MetaData.csv",
            "/home/shared/CMV/RawData/Comments_TextData.csv",
            "/home/shared/CMV/RawData/Submissions_MetaData.csv",
            "/home/shared/CMV/RawData/Submissions_TextData.csv"]

                                                        outputs =
  ['/home/shared/CMV/SlimmedData/Slimmed_Comments_Met
                                                        aData.csv',
            '/home/shared/CMV/SlimmedData/Slimmed_Comments_
TextData.csv',
            '/home/shared/CMV/SlimmedData/Slimmed_Submissions_Meta
Data.csv',
            '/home/shared/CMV/SlimmedData/Slimmed_Submissions_TextD
ata.csv']

        columns_lst = [["name", "parent_id", "author", "link_id"],
                       ["author", "id", "parent_id", "body"],
                       ["url", "id", "author"],
                       ["author", "id", "title", "selftext"]]

        slimmer.slim_all(inputs, outputs, columns_lst)

        input =
        '/home/shared/CMV/SlimmedData/Slimmed_Comme
        nts_TextData.csv'

        deltas_file =
        '/home/shared/CMV/SortedData/delta_winning_ids.txt'
        deltas_data_file =
        '/home/shared/CMV/SortedData/delta_comments_data.csv'
        nodeltas_data_file =
            '/home/shared/CMV/SortedData/nodelta_comments_data.csv'

        labeler.get_deltas(input, deltas_file)

        labeler.create_labels(input, deltas_data_file, nodeltas_data_file,
        deltas_file)

        # get input files
```

```
delta_input =
"/home/shared/CMV/SortedData/delta_comments_data.csv"
nodelta_input =
"/home/shared/CMV/SortedData/nodelta_comments_data.csv"
word_list_input =       "../data/word_list.csv"

# make output files

output_delta =
"/home/shared/CMV/FeatureData/all_delta_feature_data.csv"
output_nodelta =
"/home/shared/CMV/FeatureData/all_nodelta_feature_data.csv"

# generate features

delta_features, nodelta_features =
features.generateFeature([delta_input, nodelta_input],  [output_elta,
output_nodelta, word_list_input, 'con')

print("Writing Features to File with Pandas")
delta_features = pd.DataFrame(data=delta_features,
columns=None) nodelta_features =
pd.DataFrame(data=nodelta_features, columns=None)

delta_features.to_csv(output_delta, index=False)
nodelta_features.to_csv(output_nodelta, index=False)
```

Supplementary F

```
nodelta_file =
"/home/shared/CMV/FeatureData/all_nodelta_feature_data.csv"
delta_file =
"/home/shared/CMV/FeatureData/all_delta_feature_data.csv"

nodelta_data =
pd.read_csv(nodelta_file)
delta_data =
pd.read_csv(delta_file)

# Merge the data set and add labels = 0 (No Delta) 1
(Delta)

data = engineer.merge([nodelta_data, delta_data])

# Split the data between features and labels

X, y = data[: , :-1], data[:, -1]

# Normalize all the features
between 0 and 1

scaler = MinMaxScaler()
X=scaler.fit_transform(X)

print("Shape of all features:", X.shape)
X_train, X_test, y_train, y_test = engineer.train_test_split (X, y,
test_size=0.33)

# Oversampling with SMOTE

X_train,y_train = engineer.smote(X_train, y_train, k_neighbors= 2,
sampling_strategy=0.8)

# To use class weights to balance the classes instead of SMOTE:

class_weight=compute_class_weight(class_weight='balanced',classes=n
p.unique(y),y=y) class_weight={0:class_weight[0],1:class_weight[1]}
print(class_weight)

sample_weight = np.zeros(len(y_train))
sample_weight[y_train==0]=class_weight[0]
sample_weight[y_train==1]=class_weight[1]
```

Supplementary G

Decision Tree:
Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning,* 1, 81–106.

Random Forest:
Breiman, L. (2001). Random forests. *Machine Learning, 45*, 5–32.

Gradient Boosting:
Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis, 38*, 367–378.

Gaussian Naive Bayes:
Hand, D. J., & Yu, K. (2001). Idiot's Bayes: Not so stupid after all? *International Statistical Review / Revue Internationale de Statistique, 69*, 385–398.

Bernoulli Naive Bayes:
Hand, D. J., & Yu, K. (2001). Idiot's Bayes: Not so stupid after all? *International Statistical Review / Revue Internationale de Statistique, 69*, 385–398.

Support Vector Machine:
Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

Ada Boost:
Schapire, R. E. (1999). A brief introduction to boosting, In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, Volume 2, Stockholm, Sweden, Morgan Kaufmann Publishers Inc.

Logistic Regression:
McCullagh, P., & Nelder, J. A. (1989). Generalized linear models. *Monographs on Statistics and Applied Probability*, *37*.

Model Building Code

```
import models
'''
Select which models you want to use. Your options are:
1    "RandomForest"
2    : "AdaBoost"
3    : "GradientBoosting"
4    : "LogisticRegression"
5    : "DecisionTree"
6    : 'GaussianNB' (Gaussian naive Bayes)
7    : 'BernoulliNB' (Bernouille naive Bayes)
8    : 'SVM' (Support Vector Machine)
'''
ModelList= [1,2,3,4,6,7,8]

# You could also directly call the models from classifiers/models.py:

import models
model = models.RandomForest()
```

Supplementary H

True positive = TP	True negative = TN
False positive = FP	False negative = FN

True Positive:

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

True Negative:

$$\frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

False Positive (Type I error):

$$\frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

False Negative (Type II error):

$$\frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}}$$

Accuracy:

$$\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}}$$

Precision:

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Recall/Sensitivity:

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Specificity:

$$\frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

F1 Score:

$$\frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Supplementary I

```
'''
Select which models you want to use. Your options are:
1   : "RandomForest"
2   : "AdaBoost"
3   : "GradientBoosting"
4   : "LogisticRegression"
5   : "DecisionTree"
6   : 'GaussianNB' (Gaussian naive Bayes)
7   : 'BernoulliNB' (Bernouille naive Bayes)
8   : 'SVM' (Support Vector Machine)
'''
ModelList= [1,2,3,4,6,7,8]
...
scores = []

for ModelNumber in ModelList:
        # Define the model
        print (“# # # Model: “+models.names[ModelNumber-1] + “...”)
        model = getattr(models, models.names[ModelNumber-1]) ()

        print(“Fitting Model”)
        model = model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        score = accuracy _score(y_pred, y_test)
        scores.append(score)
        print(“Score:”, score)

        cm = confusion_matrix(y_test, y_pred)
        cm = cm.astype(‘float’) / cm.sum(axis=1) [:, np.newaxis]
#Normalize
        print(“Confusion Matrix: \n”, cm)

        plot_confusion_matrix(model, X_test, y_test,
                display_labels=[‘no delta’, ‘delta’],
                cmap=plt.cm.Blues,
                normalize=’true’)
                plt.title(models.names[ModelNumber-1])

print(“Saving the confusion matrix for {0} as
        confusion_matrix_for{0}.png”.format
(models.names[ModelNumber-1]))
```

```
    plt.savefig("confusion_matrix_for_{0}.png".format(models.names[
ModelNumber-1]))
```

# To obtain the different accuracy scores, run lines 84–153 in classifiers/main.py

# The function that computes accuracy is sklearn.metrics.accuracy_score

Supplementary J

Feature importance for the Random Forest classifier using SMOTE were calculated with the function *getImportances()* in *classifiers/lib.py*, which is called in *classifiers/main.py:*

```
        lib.getImportances(model, delta_data.columns[:-
1],savefig="feature_importance.png")
```