Implementation of Strategic Classification

Michael Asante

May 2, 2021

Abstract

Classification is a fundamental problem in machine learning. Strategic Classification considers a two-player game: One player aims to construct a rule to find a classification that is as accurate as possible; while the other player aims to game the other player's rule to construct a submission that achieves the highest classification possible. However, each time the player games, it is done at an inherent cost. This is a Python notebook implementation of an algorithm that was developed in the paper "Strategic Classification" by M. Hardt, N. Megiddo, C. Papadimitriou, and M. Wootters to optimize the publisher's classification rule into the Apontador dataset.

Contents

1 Model and Preliminaries **3**
1.1 Definitions 3
1.2 Separability 5
1.3 Gaming-Robust Classification Algorithm 6
1.3.1 Guarantee of Uniform Robustness 7

2 Implementation **8**
2.0.1 Pre-processing 8
2.0.2 Algorithm Implementation 9
2.0.3 Issues 10

3 Conclusion and Related Works **10**

References **11**

A Python Code **13**

Introduction

Suppose a person wants a job and has to submit a resume for their application. When the candidate submits their application to a website, an automated human resource client will determine how qualified the person is, based on different attributes on the resume(Faliagka, Ramantas, Tsakalidis, & Tzimas, 2012). If the attributes of the resume meet a certain criteria, the client will label the application as: 'good candidate.' Otherwise, it will label it as: 'bad candidate.' In machine learning, the human resource client is called a classifier and the labeling of resumes is called classification. A classifier is submitted by a publisher.

Ideally, the classifier is kept secret; meaning, the methodology of classification is not known to the actors being classified. In practice, this is not always the case. This can be for many reasons, such as law compliance, leaks, or simple trial-and-error. Because of this, the actors may try to 'game' the classifier by receiving their classification, manipulating their attributes, and submitting it again to achieve their preferred classification. Using the previous example, the job applicant may know what keywords to use on their new resume in order to achieve the 'good candidate' label. Realizing this, the publisher will change their criteria to prevent successful gaming. This game between the actor and the publisher is strategic classification.

The game is an example of a Stackleberg Competition: see (Korzhyk, Yin, Kiekintveld, Conitzer, & Tambe, 2011); in which the publisher of the classifier can act on their strategy first, while the actor responds to the classifier. The publisher behaviors like a market leader and the actor is the respondent. The objective of this game is to achieve a Stackelberg Equilibrium, where the publisher achieves the greatest payoff, given the best responses of the actor.

Before I explain the formal model used by (Hardt, Megiddo, Papadimitriou, & Wootters, 2016), I will introduce two players to the game. Jury will be the player that will publish the classifier, and Contestant will be the player who games their attributes to achieve a preferred classification. Each time the Contestant games, it is done at an inherent cost. However, the Contestant has a limited budget in which she can spend. The cost function that will be defined is separable.

Motivations

I have spent the past year learning about algorithmic game theory due to personal interest. After taking a class in machine learning, I was curious how the two topics were related. I wondered how decisions of the classified can affect the classification. After asking the question, I stumbled across topics of adversarial games and machine learning. Through reading additional papers on the topic, I found strategic classification; I wanted to implement the project myself to understand the results.

Chapter 1

Model and Preliminaries

We begin by introducing the model described by (Hardt et al., 2016), where Jury has all information. Recall, the players in the game are Jury and Contestant.

1.1 Definitions

In this section, I present formal definitions of the model; this comes from the (Hardt et al., 2016).

Suppose I have population X and a probability distribution $\mathcal{D}$ over X. Also, for $x, y \in X$, a cost function $c(x, y) : X \times X \to \mathbb{R}_+$ and a target classifier $h : X \to \{-1, 1\}$.

The Jury knows the cost function, the probability distribution $\mathcal{D}$, and the true classifier h. Additionally, the Jury publishes a classifier $f : X \to \{-1, 1\}$. On the other hand, the Contestant knows the cost function, the probability distribution, and the published classifier, f. The function $\Delta : X \to X$ is the function produced by the Contestant. This function produced is the movement of the Contestant.

Informally, the best move of the Contestant is:

$$\Delta(x) = \text{argmax} f(y) - c(x, y) \tag{1.1}$$

Note that it costs nothing for Contestant to not game at all as $c(x, x) = 0$. Thus, if $f(x) = 1$, then $\Delta(x) = x$; if $f(x) = -1$ and $y = \text{argmin } c(x, y)$, then

$$\Delta(x) = \begin{cases} y & c(x, y) < 2 \\ x & c(x, y) \geq 2 \end{cases} \tag{1.2}$$

Because of this, the optimal response of Contestant from x to y can only occur if the cost between the two is less than 2. This is because the tradeoff from going to 1 to -1 is $1 - (-1) = 2$.

Note that the payoff of Jury is the probability that the true human classification value is equal to the Jury's classification of Contestant's movement. Formally, it is under the form:

$$\mathbb{P}\text{r}_{x \sim \mathcal{D}}[h(x) = f(\Delta(x))] \tag{1.3}$$

The payoff of the Contestant is the expected value of the difference between Jury's classification value of Contestant's movement and the cost from moving example to the optimal response.

$$\mathbb{E}_{x \sim \mathcal{D}}[f(\Delta(x)) - c(x, \Delta(x))] \tag{1.4}$$

1.2 Separability

By (Hardt et al., 2016), a cost function is separable if it can be written in the form:

$$c(x, y) = \max\{0, c_2(x) - c_1(x)\} \tag{1.5}$$

where $c_1, c_2 : X \to \mathbb{R}$. For intuition, imagine c_1 and c_2 as the appraisal value for each of the elements within the population. The maximum implies a non-zero cost. Since $c_1(X) \subset c_2(X)$, the Contestant can choose not to game. These values can be increased with cost, or decreased without cost. Additionally, it can be increased with weight. To do this, we write the same separable cost function in the form by Hardt et al:

$$c(x, y) = \langle \alpha, (y - x) \rangle_+ \tag{1.6}$$

and for $t \in \mathbb{R}$ and $x \in X$, the Jury will return a classifier $c_2[t]$ under the form

$$c_2[t](x) = \begin{cases} 1 & c_2(x) \geq t \\ -1 & c_2(x) < t \end{cases} \tag{1.7}$$

Consider t as a threshold value in which the Jury will use to classify the Contestant. For context, the Contestant will take this classification and optimally respond; this will produce Δ.

Although the Jury uses the separable cost function for training their classifier, the Contestant does not follow the same separable assumption. Contestant uses an underlying cost function that utilizes both the linear cost function and a Euclidean distance. By (Hardt et al., 2016), this is under the form:

$$c_{true}(x, y) = (1 - \varepsilon)\langle \alpha, (y - x) \rangle_+ + \varepsilon ||x - y||_2^2 \tag{1.8}$$

Under the same note, the Jury does not completely know the true α value. This means that Jury must make up their own weight vector, α', and will not know how much their vector differs from the true α value. Thus, Jury trains the classifier under the linear cost function in the form:

$$c_{assumed}(x, y) = \langle \alpha', (y - x) \rangle_+ . \tag{1.9}$$

1.3 Gaming-Robust Classification Algorithm

```
Algorithm 1: $\mathcal{A}$: gaming -robust classification algorithm for separable cost functions
  Input: Labeled examples $(x_1, h((x_1)), .., (x_m, , h(x_m))$ from $x_i \sim \mathcal{D}$
         i.i.d.. Also, a description of a separable cost function
         $c(x, y) = \max\{0, c_2(y) - c_1(x)\}$.
1 for $i = 1, ..., m$ do
2     $t_i := c_1(x_i)$
3     [1] $s_i(x) := \begin{cases} \max(c_2(X) \cap [t_i, t_i + 2]) & (c_2(X) \cap [t_i, t_i + 2] \neq \emptyset \\ \infty & (c_2(X) \cap [t_i, t_i + 2] = \emptyset \end{cases}$
4 Compute
      $\widehat{\text{err}}(s_i) := \frac{1}{m} \sum_{j=1}^{m} \mathbf{1}\{h(x_j) \neq c_1[s_i - 2](x_j)\}$
5 Find $i^*, 1 \leq i^* \leq m + 1$,that minimizes $\widehat{\text{err}}(s_i)$
6 return: $f := c_2[s_{i^*}]$
```

For clarity, the inputs are each of the examples within the population from the probability distribution $\mathcal{D}$, as well as their true classification. Additionally, the definition of the separable cost function must be included. For each of the elements selected from the probability distribution, take their appraisal value.

The Jury will take the appraisal value, and determine how far the Contestant will go. Recall, the furthest Contestant will move from her current point is 2 units. If there are no values that are within 2 units away from their current position, then s_i will equal ∞ and Contestant will not move. Otherwise, s_i will hold the maximum value that Contestant is willing to switch.

Next, the Jury will calculate the error. For each of the values of x_i, the Jury will take the s_i value and determine the classification from $c_1[t](x)$. The definition of $c_1[t]$ for $t \in \mathbb{R}$ is not important, as $\mathcal{A}$ only has blackbox access to c_1 (Hardt et al., 2016). After determining the classification, the Jury will

[1] As you would expect by now, this algorithm stems from (Hardt et al., 2016)

determine if it is equal to the true classification value. If it is not equal, the value is 1, otherwise, the value is 0. Finally, the Jury will take the average of all values computed and store it in $\widehat{\text{err}}(s_i)$.

The Jury will find the index, i^*, that contains the smallest amount of error. At last, Jury will compute the Contestant's classification, based on the s_{i^*} value. Everything explained is within a gaming environment; meaning, the Contestant will optimally respond to her classification value once it is published by Jury.

1.3.1 Guarantee of Uniform Robustness

As a quick aside, here is the guarantee of uniform strategy robustness for Algorithm 1. This means that the algorithm for jury is optimal and the algorithmic efficiency (complexity and running time) are dependent on the Rademacher complexity of classifiers. The definition and theorem[2] are from (Hardt et al., 2016)

Definition. *Let $\mathcal{F}$ be a class of functions for $f : X \to \mathbb{R}$, the Rademacher complexity of $\mathcal{F}$ with sample size m is:*

$$R_m(\mathcal{F}) := \mathbb{E}_{x_1,...,x_m \sim \mathcal{D}} \mathbb{E}_{\sigma_1,...,\sigma_m} [\sup\{\frac{1}{m} \sum_{i=1}^{m} \sigma_i f(x_i) : f \in \mathcal{F}\}]$$

Where $\sigma_1, ..., \sigma_m$ are Rademacher random variables.

Theorem. *For concept class $\mathcal{H}$, distribution $\mathcal{D}$, separable cost function c, and sample m from Algorithm 1:*

$$\textit{if } R_m(\mathcal{H}) + 2\sqrt{\frac{\log_e(m+1)}{m}} + \sqrt{\frac{log_e(\frac{2}{\delta})}{8m}} \leq \frac{\varepsilon}{8}, \mathcal{A} \textit{ is unifrom strategy robust.}$$

[2]For more information regarding Rademacher complexity, please see (Mohri & Rostamizadeh, 2009) and (Yin, Kannan, & Bartlett, 2019). For a formal proof of the previous theorem, please reference (Hardt et al., 2016).

Chapter 2

Implementation

This is a narration of each of the code that is presented within the *Appendix* section of this paper. The programming language used was Python. I utilized the pandas and numpy library for data analysis, and I used sklearn for modeling. For the Jupyter Notebook that provides in-depth explanation, graphics, analysis of code, ROC, and precision please contact me.

2.0.1 Pre-processing

For the experiments, I attempted to follow the same procedures that was used in (Hardt et al., 2016). I used the Apontador dataset, given and described by (Costa, Merschmann, Barth, & Benevenuto, 2014), to create a gaming scenario between the social media moderators and spammers. The dataset had 7076 labeled examples with 60 features. The top half is non-spam examples, while the bottom half is spam. Only the first 15 most discriminating features were taken into account laid out.

I first labeled each feature by it's description outlined in (Costa et al., 2014). Afterward, I manually ranked each feature and sort them by rank in ascending order within my dataframe. The true classification, $h(x)$, is given the rank 0 for ease of access. I give all irrelevant features a -1 rank. Next, I standardize each feature to have mean, μ, be 0 and standard deviation, σ', to be 1. This allows for the features to be normal. Additionally, I drop the columns containing the value -1 for their rank and remove the last two columns. Remember, these were the were the description of features and the rank, which are irrelevant now that they are sorted.

I define an α vector. I named this *research alpha* which is the cost coefficients described in (Costa et al., 2014). In summary, -1 is very costly to decrease $-.1$ is slightly costly to decrease, and $.1$ is cheap to increase. Next, I define a function that adds Gaussian Noise to the alpha vector. The role of noise and randomness in strategic classification is that it improves equilibria outcomes, meaning it improves accuracy; it also gives an optimal solution to a two player game and increases efficiency (Braverman & Garg, 2020). Because the Jury does not know what *research alpha* is, the Jury must be trained under an α it thinks it knows, as mentioned previously. I will suppose that our noise contribution will be Gaussian with $\mu = 0$. Additionally, I have re-normalized the new noisy vector. Thus, it will have the same Euclidean length as the original vector *research alpha*. This noisy vector is called *my alpha* (This is equivalent to α' for Jury assumed cost function.)

To understand the data more, I check the mean and the standard deviation of the spam and non spam with each weight vector. I notice that the spam and nonspam separated mean values, but have very high standard deviation. I seek to find a weight vector, α', that produces a cost value which provides better separation of spam and not spam. I will investigate Logistic Regression, the Standard Support Vector Classifier as well as Support Vector Regression. Here, our weight vector is not *research alpha*, but the machine learning algorithm's coefficients. Afterward, I compare the means and standard deviations of spam and not spam given the different coefficients in the algorithm.

2.0.2 Algorithm Implementation

Now I implement Algorithm 1. First, define the function of Jury's optimal movements. Recall, the separability assumption is relaxed and includes a Euclidean Norm to their cost function given a ε and a true coefficient α. To put simply, Contestant will move to direction of optimal if it is identified as spam. I create a function for Jury to publish the classifier f using Algorithm 1. Now, I define the cost function for Jury using *my alpha* and define t, the budget of gaming, and ε for Contestant's response. Then, I play the game n times. Finally, I determine the accuracy of the Jury given varying units of ε. I consider a range of different α' values, primarily from the prediction coefficients of logistic regression and support vector regression. For fun, I also use classical machine learning algorithms on the dataset.

2.0.3 Issues

During my implementation, there were a lot of issues that occurred. The biggest one was that there were slight ambiguity with my cost function as well as the true weight vector. Also, I had trouble quantifying the non-linear component of the cost function. The epsilon value had to be reduced in order for Jury to achieve a high accuracy, meaning the Contestant chose optimal movements given a published classifier and the Jury did not find a good equilibrium.

Chapter 3

Conclusion and Related Works

Strategic classification is an interesting topic that is applicable in industry. It mixes the fundamental ideas of game theory and apply them into the context of computer science; in this case, machine learning. As mentioned earlier, human resource systems can use these methods, but many other fields can use it as well. For example, search engines can use strategic classification on websites that aim have a higher query result by utilizing search engine optimization, or email sites determining if an email is spam or not spam. The application of strategic classification can be endless.

Although the paper rigorously introduces the topic, there are still some questions that could be asked. For example, a full information game gives the Contestant a big advantage towards her optimal response and the Jury for computing an equilibrium. One could ask the question: what if the Jury does no know the true features and are completely oblivious to the cost of gaming from the Contestant? Additionally, what if all Contestants in each game are not strategic at all? The only way to solve this problem in the perspective of Jury is to observe the behavior of Contestant, and computing the equilibrium from there. My explanation is a simplification; therefore, please see the related work in (Dong, Roth, Schutzman, Waggoner, & Wu, 2018).

Another big problem that occurs with strategic classification, especially in regard to banking and insurance, is its ethics and social cost. Meaning, the negative characteristics in strategic classification can negatively affect disadvantaged individuals. In the paper (Milli, Miller, Dragan, & Hardt, 2019), the FICO credit score case study demonstrates this idea.

Overall, this short project was very interesting and developed my ability to research advanced topics. Given more time, further works would include an implementation of this topic using the related works that was mentioned previously, including an application of strategic classification on another dataset. I would also try to find alternative approaches to increase precision accuracy, and implement hybrid approaches using different known classification algorithms.

References

Braverman, M., & Garg, S. (2020). The role of randomness and noise in strategic classification. *arXiv preprint arXiv:2005.08377*.

Costa, H., Merschmann, L. H., Barth, F., & Benevenuto, F. (2014). Pollution, bad-mouthing, and local marketing: The underground of location-based social networks. *Information Sciences*, *279*, 123–137.

Dong, J., Roth, A., Schutzman, Z., Waggoner, B., & Wu, Z. S. (2018). Strategic classification from revealed preferences. In *Proceedings of the*

2018 acm conference on economics and computation (pp. 55–70).
Faliagka, E., Ramantas, K., Tsakalidis, A., & Tzimas, G. (2012). Application of machine learning algorithms to an online recruitment system. In *Proc. international conference on internet and web applications and services.*
Hardt, M., Megiddo, N., Papadimitriou, C., & Wootters, M. (2016). Strategic classification. In *Proceedings of the 2016 acm conference on innovations in theoretical computer science* (pp. 111–122).
Korzhyk, D., Yin, Z., Kiekintveld, C., Conitzer, V., & Tambe, M. (2011). Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *Journal of Artificial Intelligence Research*, *41*, 297–327.
Milli, S., Miller, J., Dragan, A. D., & Hardt, M. (2019). The social cost of strategic classification. In *Proceedings of the conference on fairness, accountability, and transparency* (pp. 230–239).
Mohri, M., & Rostamizadeh, A. (2009). Rademacher complexity bounds for non-iid processes.
Yin, D., Kannan, R., & Bartlett, P. (2019). Rademacher complexity for adversarially robust generalization. In *International conference on machine learning* (pp. 7085–7094).

Appendix A

Python Code

```
from scipy.io import arff
import pandas as pd
data_df = pd.DataFrame(data[0])
data_description = data_df.describe()
data_description.iloc[0]
data_class1_unique = data_df['class1'].unique()
data_class1_unique[0] == b'spam'
data_class1_is_bSpam = data_df['class1'] == b'spam'
data_class1_is_bSpam.sum()
data_columns_long = data_df.columns
data_columns_long.shape

import numpy as np

data_columns_long_array = np.array(data_columns_long)
type(data_columns_long_array)
data_columns_long_array_tr = data_columns_long_array.transpose()

data_columns_long_array_tr.shape
data_columns_long_array_tr
data_verbose_df.shape
data_verbose_df.loc[7076] = np.array(data_columns_long)
data_verbose_df.tail()
data_verbose_df.loc[7077] = -1
data_verbose_df.iloc[7076].loc['qLikes_tip'] = 'Clicks on the link
    This tip helped me'
```

```
data_verbose_df.iloc[7077].loc['qLikes_tip'] = 43
data_verbose_df.iloc[7076].loc['qAbuse_tip'] = 'Clicks on the link
    Report abuse'
data_verbose_df.iloc[7077].loc['qAbuse_tip'] = 52
data_verbose_df.iloc[7076].loc['qPlaces_usr'] = 'Number of places
    registered by the user'
data_verbose_df.iloc[7077].loc['qPlaces_usr'] = 20
data_verbose_df.iloc[7076].loc['qTips_usr'] = 'Number of tips
    posted by the user'
data_verbose_df.iloc[7077].loc['qTips_usr'] = 13
data_verbose_df.iloc[7076].loc['qPhotos_usr'] = 'Number of photos
    posted by the user'
data_verbose_df.iloc[7077].loc['qPhotos_usr'] = 17
data_verbose_df.iloc[7076].loc['qClicks_plc'] = 'Number of clicks
    on the place page'
data_verbose_df.iloc[7077].loc['qClicks_plc'] = 35
data_verbose_df.iloc[7076].loc['qTips_plc'] = 'Number of tips on
    the place'
data_verbose_df.iloc[7077].loc['qTips_plc'] = 1
data_verbose_df.iloc[7076].loc['rating_plc'] = 'Place rating'
data_verbose_df.iloc[7077].loc['rating_plc'] = 2
data_verbose_df.iloc[7076].loc['qThumbsdown_plc'] = 'Clicks on the
    link "Thumbs down"'
data_verbose_df.iloc[7077].loc['qThumbsdown_plc'] = 29
data_verbose_df.iloc[7076].loc['qThumbsup_plc'] = 'Clicks on the
    link "Thumbs up"'
data_verbose_df.iloc[7077].loc['qThumbsup_plc'] = 28
data_verbose_df.iloc[7076].loc['simil_avg_tip'] = 'Similarity
    score (avg)'
data_verbose_df.iloc[7077].loc['simil_avg_tip'] = 25
data_verbose_df.iloc[7076].loc['simil_max_tip'] = 'Similarity
    score (max)'
data_verbose_df.iloc[7077].loc['simil_max_tip'] = 19
data_verbose_df.iloc[7076].loc['simil_min_tip'] = 'Similarity
    score (min)'
data_verbose_df.iloc[7077].loc['simil_min_tip'] = 45
data_verbose_df.iloc[7076].loc['simil_median_tip'] = 'Similarity
    score (median)'
data_verbose_df.iloc[7077].loc['simil_median_tip'] = 23
data_verbose_df.iloc[7076].loc['simil_sd_tip'] = 'Similarity score
    (sd)'
data_verbose_df.iloc[7077].loc['simil_sd_tip'] = 27
data_verbose_df.iloc[7076].loc['qunigram_avg_tip'] = 'Number of
    distinct 1-gram'
data_verbose_df.iloc[7077].loc['qunigram_avg_tip'] = 12
data_verbose_df.iloc[7076].loc['qunigram_avg_fraction_tip'] =
    'Fraction of 1-gram'
data_verbose_df.iloc[7077].loc['qunigram_avg_fraction_tip'] = 26
data_verbose_df.iloc[7076].loc['qbigram_avg_tip'] = 'Number of
    distinct 2-gram'
data_verbose_df.iloc[7077].loc['qbigram_avg_tip'] = 59
data_verbose_df.iloc[7076].loc['qbigram_avg_fraction_tip'] =
    'Fraction of 2-gram'
data_verbose_df.iloc[7077].loc['qbigram_avg_fraction_tip'] = 60
data_verbose_df.iloc[7076].loc['qtrigram_avg_tip'] = 'Number of
    distinct 3-gram'
data_verbose_df.iloc[7077].loc['qtrigram_avg_tip'] = 57
data_verbose_df.iloc[7076].loc['qtrigram_avg_fraction_tip'] =
    'Fraction of 3-gram'
data_verbose_df.iloc[7077].loc['qtrigram_avg_fraction_tip'] = 58
data_verbose_df.iloc[7076].loc['qSpamWords_tip'] = 'Number of spam
    words and spam rules'
data_verbose_df.iloc[7077].loc['qSpamWords_tip'] = 31
data_verbose_df.iloc[7076].loc['qCapitalChar_tip'] = 'Number of
    capital letters'
data_verbose_df.iloc[7077].loc['qCapitalChar_tip'] = 15
data_verbose_df.iloc[7076].loc['qNumeriChar_tip'] = 'Number of
    numeric characters'
data_verbose_df.iloc[7077].loc['qNumeriChar_tip'] = 7
data_verbose_df.iloc[7076].loc['qPhone_tip'] = 'Number of phone
    numbers on the text'
data_verbose_df.iloc[7077].loc['qPhone_tip'] = 6
data_verbose_df.iloc[7076].loc['qEmail_tip'] = 'Number of email
    addresses on the text'
data_verbose_df.iloc[7077].loc['qEmail_tip'] = 3
data_verbose_df.iloc[7076].loc['qURL_tip'] = 'Number of URLs on
    the text'
data_verbose_df.iloc[7077].loc['qURL_tip'] = 5
data_verbose_df.iloc[7076].loc['qContacts_tip'] = 'Number of
    contact information on the text'
data_verbose_df.iloc[7077].loc['qContacts_tip'] = 4
data_verbose_df.iloc[7076].loc['qWords_tip'] = 'Number of words'
data_verbose_df.iloc[7077].loc['qWords_tip'] = 10
data_verbose_df.iloc[7076].loc['qCapitalWords_tip'] = 'Number of
    words in capital'
data_verbose_df.iloc[7077].loc['qCapitalWords_tip'] = 34
data_verbose_df.iloc[7076].loc['dist_avg_usr'] = 'Distance among
    all places reviewed by the user (avg)'
data_verbose_df.iloc[7077].loc['dist_avg_usr'] = 38
data_verbose_df.iloc[7076].loc['dist_max_usr'] = 'Distance among
    all places reviewed by the user (max)'
data_verbose_df.iloc[7077].loc['dist_max_usr'] = 42
data_verbose_df.iloc[7076].loc['dist_min_usr'] = 'Distance among
    all places reviewed by the user (min)'
data_verbose_df.iloc[7077].loc['dist_min_usr'] = 46
data_verbose_df.iloc[7076].loc['dist_median_usr'] = 'Distance
    among all places reviewed by the user (median)'
data_verbose_df.iloc[7077].loc['dist_median_usr'] = 36
data_verbose_df.iloc[7076].loc['dist_sd_usr'] = 'Distance among
    all places reviewed by the user (sd)'
data_verbose_df.iloc[7077].loc['dist_sd_usr'] = 24
data_verbose_df.iloc[7076].loc['qOffensWords_tip'] = 'Number of
    offensive words on the text'
data_verbose_df.iloc[7077].loc['qOffensWords_tip'] = 40
data_verbose_df.iloc[7076].loc['hasOffensWords_tip'] = 'Value of
    Has offensive word'
data_verbose_df.iloc[7077].loc['hasOffensWords_tip'] = 32
data_verbose_df.iloc[7076].loc['clust_gph'] = 'Clustering
    coefficient'
data_verbose_df.iloc[7077].loc['clust_gph'] = 39
data_verbose_df.iloc[7076].loc['recip_gph'] = 'Reciprocity'
data_verbose_df.iloc[7077].loc['recip_gph'] = 53
data_verbose_df.iloc[7076].loc['indeg_gph'] = 'Number of followers
    (in-degree)'
data_verbose_df.iloc[7077].loc['indeg_gph'] = 14
data_verbose_df.iloc[7076].loc['outdeg_gph'] = 'Number of
    followees (out-degree)'
data_verbose_df.iloc[7077].loc['outdeg_gph'] = 48
data_verbose_df.iloc[7076].loc['followers_followees_gph'] =
    'Fraction of followers per followees'
data_verbose_df.iloc[7077].loc['followers_followees_gph'] = 11
data_verbose_df.iloc[7076].loc['degree_gph'] = 'Degree'
data_verbose_df.iloc[7077].loc['degree_gph'] = 33
data_verbose_df.iloc[7076].loc['betwee_gph'] = 'Betweenness'
data_verbose_df.iloc[7077].loc['betwee_gph'] = 54
data_verbose_df.iloc[7076].loc['assort_inin_gph'] = 'Assortativity
    (in-in)'
data_verbose_df.iloc[7077].loc['assort_inin_gph'] = 44
data_verbose_df.iloc[7076].loc['assort_inout_gph'] =
    'Assortativity (in-out)'
data_verbose_df.iloc[7077].loc['assort_inout_gph'] = 37
data_verbose_df.iloc[7076].loc['assort_outin_gph'] =
    'Assortativity (out-in)'
data_verbose_df.iloc[7077].loc['assort_outin_gph'] = 41
data_verbose_df.iloc[7076].loc['assort_outout_gph'] =
    'Assortativity (out-out)'
data_verbose_df.iloc[7077].loc['assort_outout_gph'] = 30
data_verbose_df.iloc[7076].loc['pagerank_gph'] = 'Pagerank'
data_verbose_df.iloc[7077].loc['pagerank_gph'] = 49
data_verbose_df.iloc[7076].loc['qAreas_usr'] = 'Number of
    different areas where the user posted a tip'
data_verbose_df.iloc[7077].loc['qAreas_usr'] = 51
data_verbose_df.iloc[7076].loc['focus_area_usr'] = 'Tip focus of a
    user'
data_verbose_df.iloc[7077].loc['focus_area_usr'] = 55
data_verbose_df.iloc[7076].loc['entropy_usr'] = 'Tip entropy of a
    user'
data_verbose_df.iloc[7077].loc['entropy_usr'] = 47
data_verbose_df.iloc[7076].loc['sentiwordnet_tip'] = 'SentiWordNet'
data_verbose_df.iloc[7077].loc['sentiwordnet_tip'] = 16
data_verbose_df.iloc[7076].loc['emoticons_tip'] = 'Emoticons'
data_verbose_df.iloc[7077].loc['emoticons_tip'] = 56
data_verbose_df.iloc[7076].loc['panas_tip'] = 'PANAS-t'
data_verbose_df.iloc[7077].loc['panas_tip'] = 50
data_verbose_df.iloc[7076].loc['sasa_tip'] = 'SASA'
data_verbose_df.iloc[7077].loc['sasa_tip'] = 22
data_verbose_df.iloc[7076].loc['senticnet_tip'] = 'SenticNet'
data_verbose_df.iloc[7077].loc['senticnet_tip'] = 18
data_verbose_df.iloc[7076].loc['happinessindex_tip'] = 'Happiness
    Index'
data_verbose_df.iloc[7077].loc['happinessindex_tip'] = 21
data_verbose_df.iloc[7076].loc['sentistrength_tip'] =
    'SentiStrength'
data_verbose_df.iloc[7077].loc['sentistrength_tip'] = 8
data_verbose_df.iloc[7076].loc['combined_tip'] = 'Combined-method'
data_verbose_df.iloc[7077].loc['combined_tip'] = 9
data_verbose_df.iloc[7077].loc['class1'] = 0
data_verbose_nonneg_df = data_verbose_df[
    data_verbose_df.columns[data_verbose_df.iloc[7077] >= 0] ]
data_verbose_pos_df = data_verbose_df[
    data_verbose_df.columns[data_verbose_df.iloc[7077]
 >= 1] ]
data_verbose_nonneg_columnsorted_df =
    data_verbose_nonneg_df.sort_values( by = 7077,
 axis=1 )
data_verbose_nonneg_columnsorted_df =
    data_verbose_nonneg_columnsorted_df.drop([7076,
 7077])
from sklearn.preprocessing import MinMaxScaler
data_verbose_nonneg_columnsorted_df['class1'] =
 pd.factorize(data_verbose_nonneg_columnsorted_df['class1'])[0]
scaler = MinMaxScaler(feature_range = (-1,1))
scaler.fit(data_verbose_nonneg_columnsorted_df)
data_verbose_nonneg_columnsorted_standardized_df =
pd.DataFrame(scaler.transform
    (data_verbose_nonneg_columnsorted_df))
np.array(data_verbose_nonneg_columnsorted_standardized_df.iloc[0,
    : ].values)
def make_alpha_prime(inputted_vector, delta):
    my_inputted_vector = np.array(inputted_vector)
    my_noise = np.array([ delta*np.random.normal() for entry in
        my_inputted_vector])
    my_noisy_vector_unnormalized = my_inputted_vector + my_noise

    research_alpha_length = np.sqrt(np.dot(inputted_vector,
        inputted_vector))
    my_noisy_vector_unnormalized_length =
        np.sqrt(np.dot(my_noisy_vector_unnormalized,
     my_noisy_vector_unnormalized))
    my_noisy_vector_normalized = my_noisy_vector_unnormalized * \
        (research_alpha_length /
            my_noisy_vector_unnormalized_length)

    return (my_noisy_vector_normalized)
#Alpha vector
research_alpha_true = np.array([-1,-1,-1,-1,-1,-1,-1, 1, 1, 0.1,
    1, 0.1, 0.1, 1, 0.1])
my_alpha_prime = make_alpha_prime(research_alpha_true, 0.05) #
    Earlier I had selected
 delta=0.1
print('research_alpha_true', research_alpha_true)
print('my_alpha_prime', my_alpha_prime)

print('research_alpha_true length: ',
 np.sqrt(np.dot(research_alpha_true,research_alpha_true)))
print('my_alpha_prime length  : ',
    np.sqrt(np.dot(my_alpha_prime,my_alpha_prime)))
data_verbose_nonneg_columnsorted_standardizedtop3538_df =
 data_verbose_nonneg_columnsorted_standardized_df.iloc[:3538]
data_verbose_nonneg_columnsorted_standardizedbottom3538_df =
 data_verbose_nonneg_columnsorted_standardized_df.iloc[3538:]
#dot top half
spam_appraisals =
 np.dot(research_alpha_true,data_verbose_nonneg_columnsorted_standardizedtop3538_df
 .iloc
[:,1:16].values.transpose())
non_spam_appraisals = np.dot(research_alpha_true,
data_verbose_nonneg_columnsorted_standardizedbottom3538_df.
iloc[:,1:16].values.transpose())
non_spam_appraisals
print("spam mean is", spam_appraisals.mean())
print("spam std is ", spam_appraisals.std() )
##NON SPAM
print("not spam mean is",non_spam_appraisals.mean())
print("not spam std is",non_spam_appraisals.std())
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C = 100.0, random_state =1, solver =
    'lbfgs', multi_class
```

```
= 'ovr',max_iter=10000)
data_classlabel_and15features =
data_verbose_nonneg_columnsorted_standardized_df.iloc[:,0:16]
y = data_classlabel_and15features.iloc[: ,0]
x = data_classlabel_and15features.iloc[: ,1 :]
lr.fit( x, y)
x.iloc[0, :].values
lr.predict_proba(x.iloc[:3, : ])
#Here we find the weight vector that lr has created by the fitting
   method
#Is it similar to the alpha that the research gives us?
#Here, our weight vector is not research_alpha_true, but lr.coef
spam_appraisals_lr =
 np.dot(lr.coef_,data_verbose_nonneg_columnsorted_standardizedtop3538_df.
 iloc[:
,1:16].values.transpose())
non_spam_appraisals_lr =
 np.dot(lr.coef_,data_verbose_nonneg_columnsorted_
 standardizedbottom3538_df.iloc[:
,1:16].values.transpose())
print(non_spam_appraisals_lr)
print("spam mean is", spam_appraisals_lr.mean())
print("spam std is ", spam_appraisals_lr.std() )
##NON SPAM
print("not spam mean is",non_spam_appraisals_lr.mean())
print("not spam std is",non_spam_appraisals_lr.std())
from sklearn.svm import SVC
svm = SVC(kernel = 'linear', C= 1.0, random_state = 1)
svm.fit(x,y)
print(svm.coef_)
print(svm.predict(x))
print(y.values)
((y==svm.predict(x)).astype(int).sum())/7076
spam_appraisals_svm =
   np.dot(svm.coef_,data_verbose_nonneg_columnsorted_
standardizedtop3538_df.iloc[:
,1:16].values.transpose())
non_spam_appraisals_svm =
   np.dot(svm.coef_,data_verbose_nonneg_columnsorted_
standardizedbottom3538_df.iloc[:
```

```
,1:16].values.transpose())
print(non_spam_appraisals_svm)
print("spam mean is", spam_appraisals_svm.mean())
print("spam std is ", spam_appraisals_svm.std() )
##NON SPAM
print("not spam mean is",non_spam_appraisals_svm.mean())
print("not spam std is",non_spam_appraisals_svm.std())

spam_predictions=(spam_appraisals >= 0)
.astype(int) *2 -1 #We transform the output to the interval [-1,1]
spam_predictions == -1
unique, counts = np.unique(spam_predictions, return_counts=True)
dict(zip(unique, counts))
non_spam_predictions=(non_spam_appraisals >
= 0).astype(int) *2 -1 #We transform the output to the interval
    [-1,1]
non_spam_predictions == -1
unique, counts = np.unique(non_spam_predictions,
    return_counts=True)
dict(zip(unique, counts))
from sklearn.svm import SVR
regressor = SVR(kernel = 'linear')
regressor.fit(x, y)
regressor.coef_
spam_appraisals_svr =
    np.dot(regressor.coef_,data_verbose_nonneg_columnsorted_
standardizedtop3538_df.iloc[:,1:16].values.transpose())
non_spam_appraisals_svr =
    np.dot(regressor.coef_,data_verbose_nonneg_columnsorted_
standardizedbottom3538_df.iloc[:,1:16].values.transpose())
print("spam mean is", spam_appraisals_svr.mean())
print("spam std is ", spam_appraisals_svr.std() )
##NON SPAM
print("not spam mean is",non_spam_appraisals_svr.mean())
print("not spam std is",non_spam_appraisals_svr.std())
def gamingresponse(datas,eps,weight,unit):
    import math
#Make a copy of the data
#If the classifier is -1, have the contestant switch to the
    optimal value from the optimal variable. This is
```

```
#based off of eff
#If the classifier is 1, then do nothing.
#Contestant cannot go above the restricted units
    lambdaval = math.sqrt(4*unit*eps + pow(eps,2) - 2*eps + 1)
    optimal = 1/(2*eps*lambdaval) * ( (weight) - lambdaval*(1 -
        eps) * (research_alpha_true/norm(research_alpha_true)))
    if(np.dot(optimal, research_alpha_true < 0)):
        optimal = weight -
        ( np.dot(research_alpha_true,weight)\
             *research_alpha_true)/ pow(norm(research_alpha_true),2)
             *research_alpha_true
        optimal = optimal/norm(optimal) * math.sqrt( unit / eps )
    data_verbose_response = datas.copy()
    ask = data_verbose_response.query('Threshold == -1').iloc[:
        ,1:16] + optimal#go by direction of optimal if it is
        identified as spam
    data_verbose_response.update(ask)
    data_verbose_response['Appraisal'] =
        np.dot(research_alpha_true,\np.array(data_verbose_response.iloc[:
        , 1:16].values.transpose()))
    #data_verbose_response['Threshold'] =
        np.select([data_verbose_response['Appraisal'] >
        f],[1],default = -1)
    return(data_verbose_response)
def gamingrobustalg(gaming_data, alpha, eps, budget,amtsim):

    import random
    import math

    each_accuracy = []
    accuracy_sum = 0
    data_verbose_sample = gaming_data.reset_index(drop = True)
    data_verbose_sample.index = data_verbose_sample.index + 1

    tee = budget #budget for gaming
    print(data_verbose_sample)
    s_i_star = m+1
    all_errors = []
    data_verbose_sample['Switch'] = np.nan
    data_verbose_sample['Error'] = np.nan
```

```
for i in range(1, len(data_verbose_sample)+1):
    error_sum_si = 0
    t_i = data_verbose_sample.loc[i]['Appraisal']
    s_i = pd.DataFrame(data_verbose_sample.query('Appraisal >
        {0} & Appraisal < {1}'.format(t_i,
        t_i+tee))['Appraisal']).max().max()
    print(s_i)
    if math.isnan(s_i):
        s_i = math.inf
   # print(s_i)
    for j in range(1,len(data_verbose_sample)+1):
        if data_verbose_sample['Appraisal'][j] < s_i - tee:
            error_prefix = -1
        else:
            error_prefix = 1
        if data_verbose_sample[0][j] != (error_prefix):
            error_sum_si = (error_sum_si + 1)
        else:
            error_sum_si = (error_sum_si + 0)
    final_error_si = error_sum_si/(len(data_verbose_sample))
    all_errors.append(final_error_si)
    #print("The s_i is {:1.1f}".format(s_i),' ', "the error is
        ", final_error_si) - debug
    data_verbose_sample['Switch'][i] = s_i
    data_verbose_sample['Error'][i] = final_error_si

#find i* that minimizes error
s_istar = all_errors.index(min(all_errors))

#compute c_2[s_i*] to find the threshold value.
#store t_i,final_error_si & s_i value to data_verbose_sample
f = data_verbose_sample['Appraisal'].reset_index(drop =
    True)[s_istar]

#Take value of f and hold it as a parameter.
#Make a new column called "eff" and use that as a threshold
data_verbose_final = data_verbose_sample.copy()
data_verbose_final['eff'] = f
#Finally, make another column that determines whether the
    appraisal value is above or below the eff column
    #make that 1 or -1
    data_verbose_final['Threshold'] =
       np.select([data_verbose_final['Appraisal'] > f],[1],default
       = -1)
    #Contestants response
    data_verbose_final =
       gamingresponse(data_verbose_final,eps,alpha,budget)
    return(data_verbose_final)
def accuracy(gaming_sample):
    each_accuracy = []
    each_accuracy.append(
    ((gaming_sample[0]==gaming_sample['Threshold']).astype(int))
    .sum()/len(gaming_sample))
    alg_accuracy =
       ((gaming_sample[0]==gaming_sample['Threshold']).astype(int))
    .sum()/len(gaming_sample)
    return(alg_accuracy)
from numpy.linalg import norm

#Creating alpha prime
my_alpha_prime = make_alpha_prime(research_alpha_true, 0.05)

#The iterations of gaming
target = 2
epsil = .1
data_verbose_nonneg_columnsorted_standardized_apprais['Appraisal']
    = (2/target)*np.dot(my_alpha_prime,\
    np.array(data_verbose_nonneg_columnsorted_standardized_df.iloc[:
    , 1:16].values.transpose()))
da = data_verbose_nonneg_columnsorted_standardized_apprais.copy()

import random
random.seed(10)
m = 20
randlist = random.sample(range(7076), k=m)
each_accuracy = []
accuracy_sum = 0
temporary_dataverbosenon_neg = da.iloc[randlist]
for i in range(0,4):
    temporary_dataverbosenon_neg['Appraisal'] =
        (2/target)*np.dot(my_alpha_prime,\
    np.array(temporary_dataverbosenon_neg.iloc[: ,
        1:16].values.transpose()))
    temporary_dataverbosenon_neg =
        gamingrobustalg(temporary_dataverbosenon_neg,my_alpha_prime,
        epsil, target,10)
    each_accuracy.append(accuracy(temporary_dataverbosenon_neg))
    accuracy_sum = each_accuracy[i] + accuracy_sum
print(accuracy_sum/(i+1))
my_alpha_prime = regressor.coef_.copy()
ques =
    (2/target)*np.dot(my_alpha_prime,np.array(data_verbose_nonneg_columnsorted_
standardized_df.iloc[: , 1:16].values.transpose()))
data_verbose_nonneg_columnsorted_standardized_apprais['Appraisal']
    = ques.transpose()
da = data_verbose_nonneg_columnsorted_standardized_apprais.copy()

import random
random.seed(10)
m = 20
randlist = random.sample(range(7076), k=m)

temporary_dataverbosenon_neg = da.iloc[randlist]
each_accuracy = []
accuracy_sum = 0
for i in range(0,4):
    ques =
        (2/target)*np.dot(my_alpha_prime,np.array(data_verbose_nonneg_columnsorted
    _standardized_df
    .iloc[: , 1:16].values.transpose()))
    data_verbose_nonneg_columnsorted_standardized_apprais['Appraisal']
        = ques.transpose()
    temporary_dataverbosenon_neg =
        gamingrobustalg(temporary_dataverbosenon_neg,my_alpha_prime,
        epsil, target,10)
    each_accuracy.append(accuracy(temporary_dataverbosenon_neg))
    accuracy_sum = each_accuracy[i] + accuracy_sum
temporary_dataverbosenon_neg
print(accuracy_sum/(i+1))
```

```
my_alpha_prime = lr.coef_.copy()
ques =
    (2/target)*np.dot(my_alpha_prime,np.array(data_verbose_nonneg_columnsorted_
standardized_df.iloc[: , 1:16].values.transpose()))
data_verbose_nonneg_columnsorted_standardized_apprais['Appraisal']
    = ques.transpose()
da = data_verbose_nonneg_columnsorted_standardized_apprais.copy()

import random
random.seed(10)
m = 20
randlist = random.sample(range(7076), k=m)

temporary_dataverbosenon_neg = da.iloc[randlist]
each_accuracy = []
accuracy_sum = 0
for i in range(0,4):
    ques =
        (2/target)*np.dot(my_alpha_prime,np.array(data_verbose_nonneg_columnsorted
    _standardized_df
    \.iloc[: , 1:16].values.transpose()))
    data_verbose_nonneg_columnsorted_standardized_apprais['Appraisal']
        = ques.transpose()
    temporary_dataverbosenon_neg =
        gamingrobustalg(temporary_dataverbosenon_neg,my_alpha_prime,
        epsil, target,10)
    each_accuracy.append(accuracy(temporary_dataverbosenon_neg))
    accuracy_sum = each_accuracy[i] + accuracy_sum
temporary_dataverbosenon_neg
print(accuracy_sum/(i+1))
import matplotlib.pyplot as plt
import seaborn as sns

#Importing
from sklearn.metrics import mean_squared_error,confusion_matrix,
    precision_score, recall_score, auc,roc_curve
from sklearn import ensemble, linear_model, neighbors, svm, tree,
    neural_network
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split,
    cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn import svm,model_selection, tree, linear_model,
    neighbors, naive_bayes, ensemble, discriminant_analysis,
    gaussian_process
from sklearn.discriminant_analysis import
    LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
x_train, x_test, y_train,
    y_test=train_test_split(x,y,test_size=0.2, random_state=0)
models = []
models.append(('LR', LogisticRegression(max_iter=10000)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
MLA = [
    #GLM
    linear_model.LogisticRegressionCV(max_iter=10000),
    linear_model.PassiveAggressiveClassifier(),
    linear_model. RidgeClassifierCV(),
    linear_model.SGDClassifier(),
    linear_model.Perceptron(),

    #Ensemble Methods
    ensemble.AdaBoostClassifier(),
    ensemble.BaggingClassifier(),
    ensemble.ExtraTreesClassifier(),
    ensemble.GradientBoostingClassifier(),
    ensemble.RandomForestClassifier(),

    #Gaussian Processes
    gaussian_process.GaussianProcessClassifier(),

    #SVM
    svm.SVC(probability=True),
    svm.NuSVC(probability=True),
    svm.LinearSVC(),

    #Trees
    tree.DecisionTreeClassifier(),

    #Navies Bayes
    naive_bayes.BernoulliNB(),
    naive_bayes.GaussianNB(),

    #Nearest Neighbor
    neighbors.KNeighborsClassifier(),

    ]
MLA_columns = []
MLA_compare = pd.DataFrame(columns = MLA_columns)

row_index = 0
for alg in MLA:

    predicted = alg.fit(x_train, y_train).predict(x_test)
    fp, tp, th = roc_curve(y_test, predicted)
    MLA_name = alg.__class__.__name__
    MLA_compare.loc[row_index,'MLA used'] = MLA_name
    MLA_compare.loc[row_index, 'Train Accuracy'] =
        round(alg.score(x_train, y_train), 4)
    MLA_compare.loc[row_index, 'Test Accuracy'] =
        round(alg.score(x_test, y_test), 4)
    MLA_compare.loc[row_index, 'Precission'] =
        precision_score(y_test, predicted)
    MLA_compare.loc[row_index, 'Recall'] = recall_score(y_test,
        predicted)
    MLA_compare.loc[row_index, 'AUC'] = auc(fp, tp)

    row_index+=1

#MLA_compare.sort_values(by = ['MLA Test Accuracy'], ascending =
    False, inplace = True)
MLA_compare
```

```
MLA_columns = []
MLA_compare = pd.DataFrame(columns = MLA_columns)

row_index = 0
for alg in MLA:

    predicted = alg.fit(x_train, y_train).predict(x_test)
    fp, tp, th = roc_curve(y_test, predicted)
    MLA_name = alg.__class__.__name__
    MLA_compare.loc[row_index,'MLA used'] = MLA_name
    MLA_compare.loc[row_index, 'Train Accuracy'] =
        round(alg.score(x_train, y_train), 4)
    MLA_compare.loc[row_index, 'Test Accuracy'] =
        round(alg.score(x_test, y_test), 4)
    MLA_compare.loc[row_index, 'Precission'] =
        precision_score(y_test, predicted)
    MLA_compare.loc[row_index, 'Recall'] = recall_score(y_test,
        predicted)
    MLA_compare.loc[row_index, 'AUC'] = auc(fp, tp)

    row_index+=1

#MLA_compare.sort_values(by = ['MLA Test Accuracy'], ascending =
    False, inplace = True)
MLA_compare
```