Simple Model Assemblages for Website Identification

Charles Hu
czhu06@wm.edu
William & Mary
Williamsburg, Virginia, USA

ABSTRACT

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work. A cite so this compiles.

KEYWORDS

Network trafifc, Network traffic classification, Machine learning, Ensemble models

1 INTRODUCTION

2 PROPOSED METHOD

The following section details the design and rationale behind the methodology used for evaluating and classifying the website of origin for given samples of web traffic using machine learning models.

2.1 Data Preparation

In order to ensure that our developed models are robust and effective at their classification task, we employ a measured and systemized approach towards data collection and preprocessing to ensure that possibility of gathering ideosyncratic or erroneous data is minimized as much as possible.

The primary objective during the design of the data preparation phase was to ensure that the generated data set allows us to construct models that are effectively generalizable and not too overfit. A key principle that allows us to achieve this is confirming that the gathered sample is as representative as possible of the general population; however, this is difficult to verify empiricaly given the complex nature of the overall population. As a result, a preemptive approach was taken which resulted in the design of a two-stage data preparation phase comprised of a precautious data collection stage and a mitigative data preprocessing stage. The data preparation process is summarized in Fig. 1.

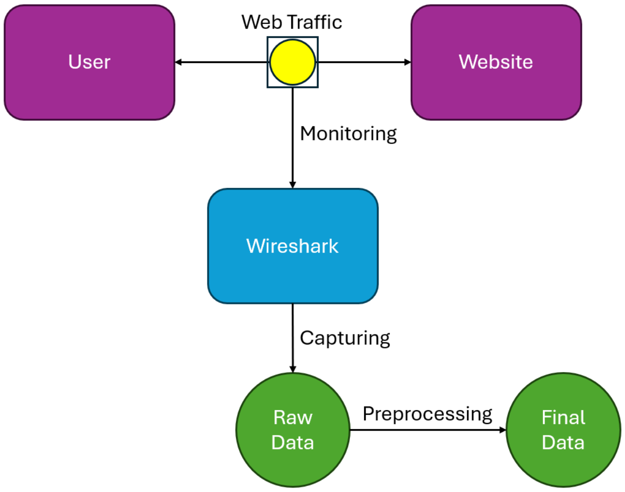
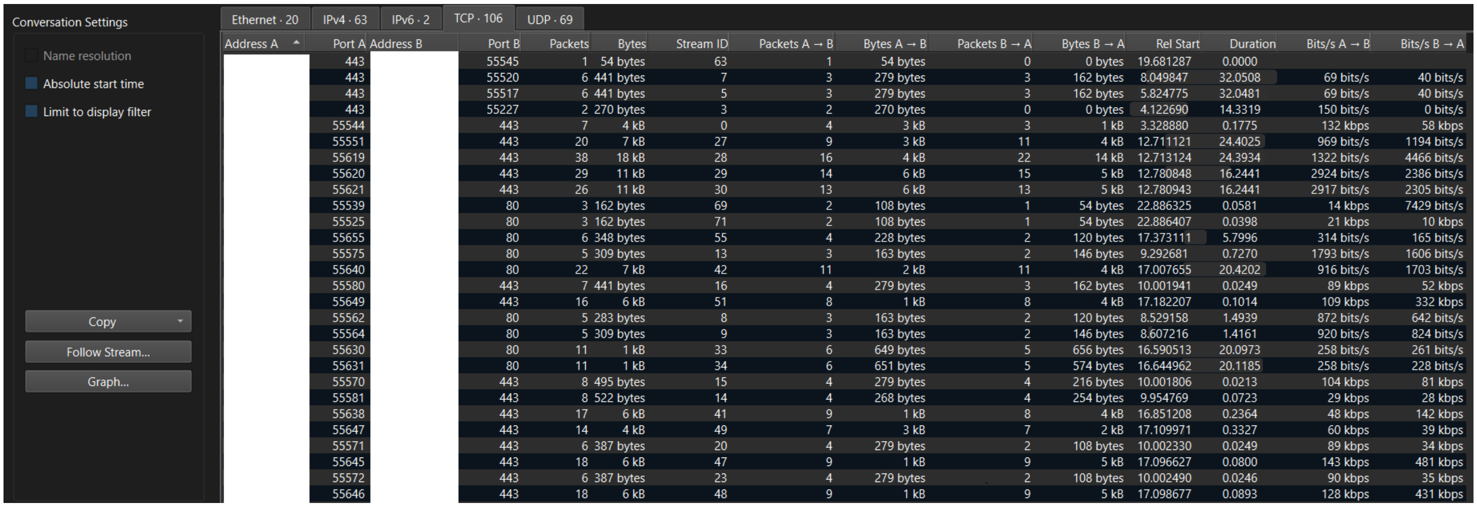


Figure 1: A diagram showing the data preparation process. It is split into 2 parts: A data collection stage and a data preprocessing stage.

2.1.1 Data Collection. Data collection is performed through the monitoring of artificial website activity aimed at emulating real user interactions common to the sampled websites. Website traffic between the user and the server hosting the website is monitored and tracked using Wireshark, an open-source software used for network traffic capture and analysis [2]. Wireshark will specifically target the transport layer of network communication and intercept ongoing TCP streams between the user and the site host. Activity on a website will occur through the controlled usage of a website's typical functionalities. For instance, on a streaming site, the data collector will utilize the site's recommendation algorithm to view a certain number of videos before halting activity. The goal during the activity substage is to interact with the website in a naturalistic manner akin to any typical user but avoid operations which either go beyond the scope of the target website (e.g., entering another website through an embedded link) or are unexpected of a user (e.g., manually performing HTTP operations with the website or abruptly closing the site during an interaction). These precautions should help minimize the amount of collected TCP streams that contain information that are erroneous or overly noisy. Once activity on a website has been concluded, Wireshark will be used to display the TCP streams generated during activity with the website (as seen in Fig. 2) and will export the TCP streams to an external CSV file for each targeted website.



Conversation Settings

☐ Name resolution
☐ Absolute start time
☐ Limit to display filter

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
[redacted]	443	[redacted]	55545	1	54 bytes	63	1	54 bytes	0	0 bytes	19.681287	0.0000		
[redacted]	443	[redacted]	55520	6	441 bytes	7	3	279 bytes	3	162 bytes	8.049847	32.0508	69 bits/s	40 bits/s
[redacted]	443	[redacted]	55517	6	441 bytes	5	3	279 bytes	3	162 bytes	5.824775	32.0481	69 bits/s	40 bits/s
[redacted]	443	[redacted]	55227	2	270 bytes	3	2	270 bytes	0	0 bytes	4.122690	14.3319	150 bits/s	0 bits/s
[redacted]	55544	[redacted]	443	7	4 kB	0	4	3 kB	3	1 kB	3.328880	0.1775	132 kbps	58 kbps
[redacted]	55551	[redacted]	443	20	7 kB	27	9	3 kB	11	4 kB	12.711121	24.4025	969 bits/s	1194 bits/s
[redacted]	55619	[redacted]	443	38	18 kB	28	16	4 kB	22	14 kB	12.713124	24.3934	1322 bits/s	4466 bits/s
[redacted]	55620	[redacted]	443	29	11 kB	29	14	6 kB	15	5 kB	12.780848	16.2441	2924 bits/s	2386 bits/s
[redacted]	55621	[redacted]	443	26	11 kB	30	13	6 kB	13	5 kB	12.780943	16.2441	2917 bits/s	2305 bits/s
[redacted]	55539	[redacted]	80	3	162 bytes	69	2	108 bytes	1	54 bytes	22.886325	0.0581	14 kbps	7429 bits/s
[redacted]	55525	[redacted]	80	3	162 bytes	71	2	108 bytes	1	54 bytes	22.886407	0.0398	21 kbps	10 kbps
[redacted]	55655	[redacted]	80	6	348 bytes	55	4	228 bytes	2	120 bytes	17.373111	5.7996	314 bits/s	165 bits/s
[redacted]	55575	[redacted]	80	5	309 bytes	13	3	163 bytes	2	146 bytes	9.292681	0.7270	1793 bits/s	1606 bits/s
[redacted]	55640	[redacted]	80	22	7 kB	42	11	2 kB	11	4 kB	17.007655	20.4202	916 bits/s	1703 bits/s
[redacted]	55580	[redacted]	443	7	441 bytes	16	4	279 bytes	3	162 bytes	10.001941	0.0249	89 kbps	52 kbps
[redacted]	55649	[redacted]	443	16	6 kB	51	8	1 kB	8	4 kB	17.182207	0.1014	109 kbps	332 kbps
[redacted]	55562	[redacted]	80	5	283 bytes	8	3	163 bytes	2	120 bytes	8.529158	1.4939	872 bits/s	642 bits/s
[redacted]	55564	[redacted]	80	5	309 bytes	9	3	163 bytes	2	146 bytes	8.607216	1.4161	920 bits/s	824 bits/s
[redacted]	55630	[redacted]	80	11	1 kB	33	6	649 bytes	5	656 bytes	16.590513	20.0973	258 bits/s	261 bits/s
[redacted]	55631	[redacted]	80	11	1 kB	34	6	651 bytes	5	574 bytes	16.644962	20.1185	258 bits/s	228 bits/s
[redacted]	55570	[redacted]	443	8	495 bytes	15	4	279 bytes	4	216 bytes	10.001806	0.0213	104 kbps	81 kbps
[redacted]	55581	[redacted]	443	8	522 bytes	14	4	268 bytes	4	254 bytes	9.954769	0.0723	29 kbps	28 kbps
[redacted]	55638	[redacted]	443	17	6 kB	41	9	1 kB	8	4 kB	16.851208	0.2364	48 kbps	142 kbps
[redacted]	55647	[redacted]	443	14	4 kB	49	7	3 kB	7	2 kB	17.109971	0.3327	60 kbps	39 kbps
[redacted]	55571	[redacted]	443	6	387 bytes	20	4	279 bytes	2	108 bytes	10.002330	0.0249	89 kbps	34 kbps
[redacted]	55645	[redacted]	443	18	6 kB	47	9	1 kB	9	5 kB	17.096627	0.0800	143 kbps	481 kbps
[redacted]	55572	[redacted]	443	6	387 bytes	23	4	279 bytes	2	108 bytes	10.002490	0.0246	90 kbps	35 kbps
[redacted]	55646	[redacted]	443	18	6 kB	48	9	1 kB	9	5 kB	17.098677	0.0893	128 kbps	431 kbps

Figure 2: A Wireshark window displaying tracked TCP streams and each stream's corresponding attributes. Note that IP addresses have been censored.

2.1.2 *Data Preprocessing.*

Data preprocessing involves pruning the resulting data set from the data collection stage to remove features that are either explicitly detrimental or extraneous. The removal of these features help improve the generalizability of our developed models by decreasing the overall model complexity and thus the variance inherent to it [3]; thus, if any idiosyncrasies or noise make it through our first stage of data collection, we can still reduce their influence on the overall model performance by lowering the capacity of the model to learn bad data. To ensure that we do not develop models that have too low complexity due to a lack of learned features (and therefore trending towards too much bias in the bias-variance tradeoff), only features that are explicitly detrimental or extraneous will be removed. Removed features are as follows:

- **Address A, Port A, Address B, Port B**: These feature are explicit identifiers for the hosts in the TCP stream. Their inclusion would make the classification task redundant and would result in a model that places high importance on observing the IP and port values over other potentially useful features.
- **Stream ID**: This feature tracks the unique internal tracking ID given by Wireshark to any given TCP stream for later reference. This is Wireshark specific and not pertinent to the TCP stream itself.
- **Total Packets, Percent Filtered**: These features refer to the fitler display feature in Wireshark which filters every available packet to search for some filter specification (in this case, the target website). These are Wireshark specific and not pertinent to the TCP stream itself.
- **Rel Start**: This feature indicates when the TCP stream began relative to the start of the Wireshark network capture session. This is Wireshark specific and not pertinent to the TCP stream itself.

Retained features from the data set demonstrate either potential for aiding a model in identifying what a particular website is or are neutral in their benefit and do not actively harm or mislead the model in any way. Retained features from the raw data set are as follows:

- **Packets**: This feature tracks the total packets transferred. Note that since this is a TCP stream, this feature instead referes to a TCP segment. Different website functionalities may result in different tendencies in frequency and amount of segments transferred during a TCP stream.
- **Bytes**: This feature tracks the total amount of data in bytes that have been transferred during the entire TCP stream. Different website functionalities may result in smaller or larger sized data payloads being sent across the TCP stream.
- **Packets A → B, Bytes A → B**: These features track the packet count and total amount of data in bytes being sent from host A (the user) to host B (the site host). How intensive interactions between the user and the website are may influence these features.
- **Packets B → A, Bytes B → A**: These features track the packet count and total amount of data in bytes being sent from host B (the site host) to host A (the user). How intensive interactions between the website and the user are may influence these features.

Table 1: TCP stream count per website in the final cumulative data set

Website	TCP Stream Count
github	100
google	102
reuters	102
wikipedia	100
youtube	102
Total	506

- **Duration**: The entire time duration of the TCP stream recorded in seconds. Different durations may help indicate the purpose and level of engagement for a website.
- **Bits/s A → B, Bits/s B → A**: These features track the bitrate of the TCP stream. These features may not necessarily help the model as bitrate is subject to a number of factors that cannot be consistently attributed to a website alone, but the feature itself is not inherently detrimental so it has been left in the data set.

2.2 Data Set

Artificial user activity was performed and monitored across 5 websites: github.com, google.com, reuters.com, wikipedia.com, and youtube.com. These websites were specifically chosen due to their distinct functionalities (e.g., youtube.com provides video streaming services, whereas reuters.com provides news content), which may lead to distinctive web traffic behaviors that can help in providing a direction for the trained models in the classification task. Table 1 describes the size of each class in the final data set. TCP stream counts per website were kept roughly the same in order to prevent data set imbalance which can lead to models over prioritizing training on the larger classes.

2.2.1 Data Analysis. Preliminary data analysis was performed on the final data set to observe general trends and behavior. Note that due to the small sample size of the data set, potential idiosyncrasies or noise may be exacerbated in the analysis methods. As such, the results of these methods are only to be taken as basic guidance for handling and interpreting the data and should always be trumped by domain knowledge. Interpretations for each analysis method will not be provided as to ensure that these methods do not influence the interpretation of the final models; however, a basic description of how they operate will be provided. The developed models do not take these analysis results into account.

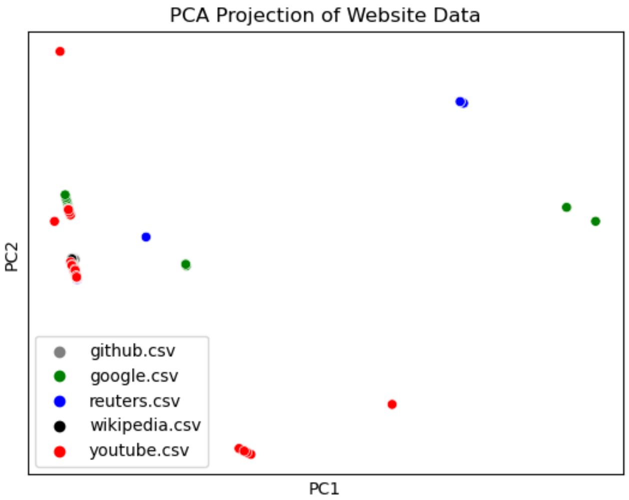


Figure 3: PCA performed on the final data set.

- **Principal component analysis (PCA)**: A PCA was performed on the data set to observe the variability in the features (see Fig. 3). PCA is used to reduce dimensionality in a data set which can help break apart correlated data, reduce complexity, and potentially remove noise. A key aspect of PCA is that while dimensionality is reduced, the variability in the feature space is maintained as much as possible.
- **t-distributed stochastic neighbor embedding (t-SNE)**: Like PCA, t-SNE performs dimensionality reduction but emphasizes the retention of distance-based relationships between points. As a result, t-SNE can sometimes help provide a low dimensional view of high dimensional relationships and groupings among points; however, distortions are likely to occur due to the inability of low dimensions to fully express the complexities of higher dimensional relationships (in a manner similar to Mercator projection maps of Earth distorting the true size of various landmasses). Fig. 4 shows a t-SNE projection performed on the final data set.
- **L1 regularization**: Fig. 5 shows L1 regularization applied to the features of the final data set. L1 regularization is commonly used to derive how important certain features are to the development of the final best fit model for prediction. The regularization is highly dependent on the specificities of the ingested data set; thus, features denoted as important should not be blindly trusted as error or noise may distort the true importance of features in the overall population.

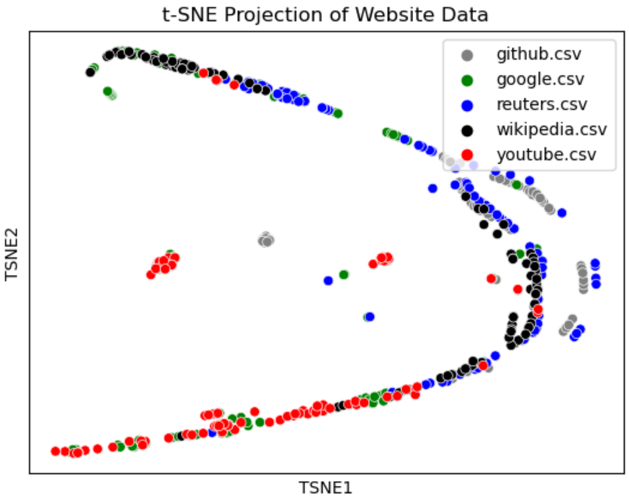


Figure 4: t-SNE performed on the final data set.

2.3 Model Development

The models used for the web traffic classification task will be developed using a mix of simple models and ensemble models. Simple models encompass basic learners (e.g., decision trees and k-nearest neighbors classifiers) which perform the classification task based on their specified learning algorithm. Ensemble models are aggregate designs which use simple models as basic components. They learn by varying either how its constituent component models learn and or how the results of the component models are aggregated. A key principle to how ensemble models work is ensemble theory, which posits that a diversity of learners will result in a stronger overall learning model. This is a result of having each constituent learner cover for its own inherent bias (i.e, weaknesses in learning certain aspects of the data set) by leveraging the strength of other constituent learners to cover for it (i.e., leverage learners which have already learned that part of the data set). The relationship between a simple model and an ensemble model are summarized in Fig. 6.

Complex machine learning models (e.g., neural networks) will be avoided in this project due to concerns over complexity modelling capabilities. In particular, strong complex machine learning models are more expressive in function modelling and thus are able to better capture the target function of a given data set (i.e., they are much better at learning how to replicate the data set). This is undesirable due to our small sample size in our final data set, which means that ideosyncratic or noisy data points have less "good" (i.e., representative of the overall population) data to buffer and suppress their influence on the final model. As a result, complex models trained on this data set have the potential to learn more bad data and generate final models which have internalized such errors and thus are no longer generalizable to our overall population. This is mitigated by using simpler models which have less modelling capacity (i.e., more bias) and thus are less capable of modelling errors in the first place.

Developed models will be compared against a logistic regression model, which will serve as the baseline. Models will be trained using a 8:2 training-test split ratio. Cross-validation and hyperparameter tuning will be used when available to optimize the training of models.

2.4 Model Specifications

6 simple model designs and 5 ensemble model designs will be employed. Ensemble models will be comprised of simple models that are either the best performing or most applicable. The developed models will utilize the model implementations provided by the scikit-learn package [4]. The details of these model designs [1, 5] are summarized in the following subsections.

2.4.1 *Simple Models.*

- **Decision tree classifier (DTC)**: A decision tree is an abstraction similar to a tree data structure where a model follows branching decisions driven by data which cumulatively form a tree-like structure. Nodes on the tree are composed of decisions which split the feature(s) and remove data from the current data set on hand based on the decision criteria. This typically continues until the data set on hand is only composed of a single feature (which is considered a pure node) which then becomes the leaf node of a branch. Classifications are based on the developed paths in the tree, where each path from the root to a leaf node corresponds to the target class which developed the path to begin with.
- **k-nearest neighbors (KNN)**: KNN operates by observing clustering and grouping within a data set. Classification occurs by attempting to find the k most similar points for a given point, and classifying such point based off of the classes of those k most similar points. The particular implementation of this design uses Euclidean distance to find the most similar points.
- **Logistic regression**: Logistic regression works by trying to find the best fit for a logistic function in the training data. Predictions for data are then generated using this best fit function. Note that though the name implies the use of regression, the implementation in scikit-learn is designed as a classifier.

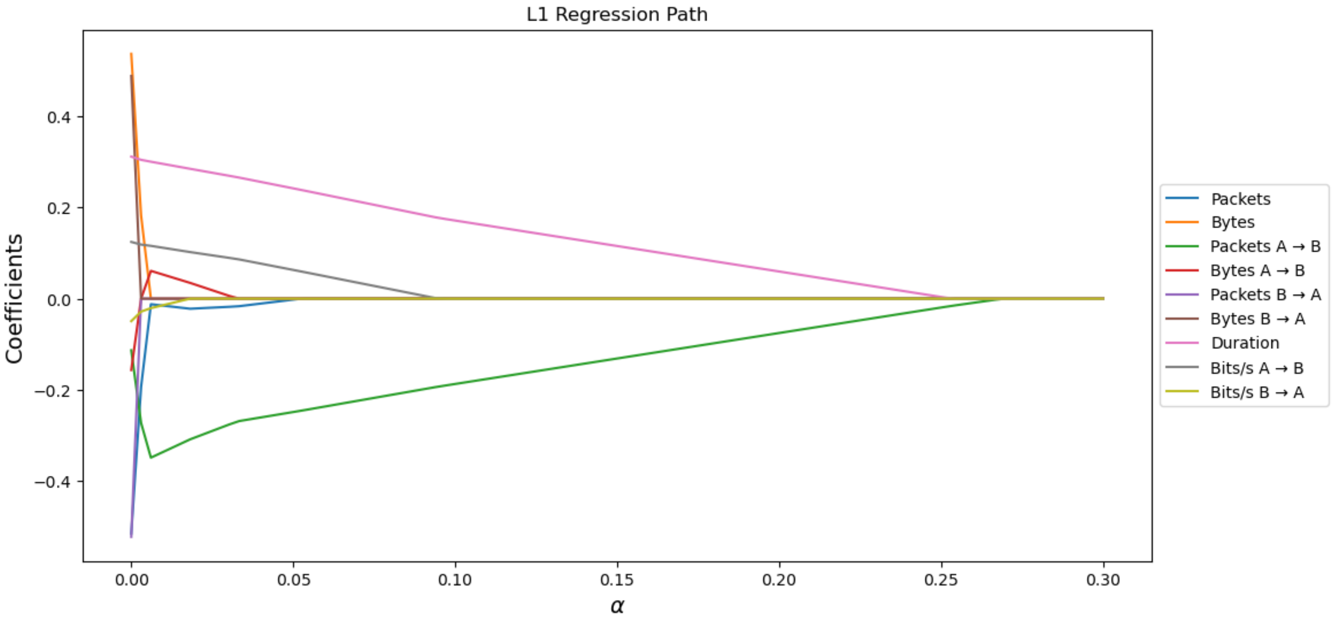


Figure 5: L1 regularization performed on the final data set. The following describes the evolution of feature importance as the regularization penalty increases.

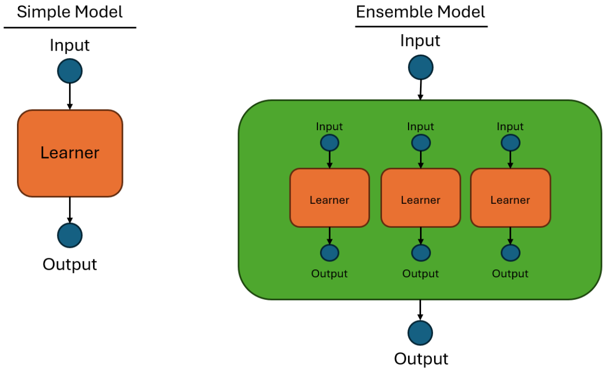


Figure 6: A diagram illustrating the relationship between a simple model and an ensemble model. Note that the described models are for demonstration and are not representative of the entire superset of simple and ensemble models.

- **Naive Bayes classifier**: Naive Bayes leverages Bayes' theorem to perform Bayesian probabilistic predictions of a target output given the surmised distribution of the data set.
- **Stochastic gradient descent (SGD)**: This is a specialized implementation in scikit-learn which uses stochastic gradient descent to optimize the loss function for a classifier given some input training data.
- **Support vector classifier (SVC)**: A variation of a support vector machine designed for classification tasks. It operates by attempting to optimize the separation of different classes in the feature space using hyperplanes.

2.4.2 *Ensemble Models.*

- **AdaBoost**: AdaBoost takes a given weak learner and iteratively trains it using the same data set. It encorporates a system of internal weights where more emphasis is placed on observations which have been misclassified by the produced model. This results in each iteration improving at classifying observations that the preceding models were weak at. These new iterations are then fitted onto the original learner to produce a new strong learner. The AdaBoost implementation in this project is built using DTCs, logistic regression, and SGD classifiers.
- **Bagging**: Bagging takes a learner and trains several instances of the learner using random subsets drawn from the training data with replacement. The individual predictions made by these learners are then aggregated in the final ensemble model. The bagging implementation in this project is built using DTCs, KNNs, and logistic regression.
- **Random forest classifier (RFC)**: RFCs are composed of multiple decision trees built using a subset

of the training data drawn with replacement. The aggregation in the scikit-learn implementation operates by combining the average probabilistic prediction instead of allowing each tree to vote for a final class.

- **Stacking**: Stacking takes multiple learners and stacks them on top of a final learner, where the output of each stacked learner is used as an input for the final learner to compute the final class prediction. The stacking implementation in this project is built using a DTC and a KNN for the stacked learners and logistic regression for the final learner.
- **Voting**: Voting takes several learners and trains them independently. It then takes the outputs of each model and counts them as votes to determine the final classification. The voting implementation in this project is built using a DTC, a KNN, and a logistic regression function. Soft (i.e., probabilistic classification) and hard (i.e., final output-based classification) voting techniques are both used.

3 EVALUATION

3.1 Evaluation Metric

3.2 Results

3.2.1 Baseline.

3.2.2 Simple Model Results.

3.2.3 Ensemble Model Results.

3.2.4 Model Results Overall.

4 DISCUSSION & FUTURE WORK

4.1 Result Interpretations

4.2 Future Work

5 CONCLUSION

REFERENCES

[1] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 108–122.

[2] Wireshark Foundation. 2024. *Wireshark*. https://www.wireshark.org/

[3] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *"Introduction to Information Retrieval"*. Cambridge University Press.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[5] scikit-learn Developers. 2024. *scikit-learn API Reference*. https://scikit-learn.org/stable/modules/classes.html