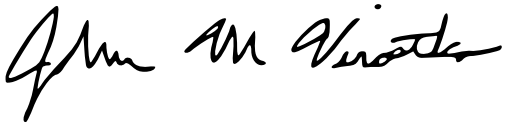
Detection of Scratch Movements Using Wrist-Based Accelerometers

Completed for the Certificate in Scientific Computation
Spring 2023

Aditya Patel
B.S Mathematics
Department of Mathematics
College of Natural Sciences



John Michael Virostko
Assistant Professor
Department of Diagnostic Medicine

Detection of Scratch Movements using Wrist Based Accelerometers

Aditya Patel

Abstract

Atopic Dermatitis (AD), commonly known as eczema, is a chronic inflammatory skin condition that affects almost 12 million persons in the United States. Scratching, or pruritus, is the most common symptom of AD that can lead to discomfort, painful skin, and skin infections. Despite the importance of early diagnosis and treatment to reduce patient morbidity, very few methodologies to objectively assess the progression of treatments exist. Clinical trials are expensive and time-consuming as the gold standard of scratch detection is video recording which requires manual labelling of scratch events. The purpose of this study was to establish a proof-of-concept for the utilization of wrist-based accelerometers for scratch detection. We collected data consisting of simulated scratch movements from 15 subjects using an Apple Watch Series 7 to extract raw accelerometer data. The subjects also performed various other movements such as brushing, washing hands, and typing to make the model robust to similar movements. We extracted features from the raw signals and trained a Random Forest model to classify between scratch motions and other movements. Finally, we performed a thorough feature analysis to obtain insights about the movements. The final model yielded promising results (0.95 ROC Score on train-test split and 0.90 ROC Score on unseen data). There were, however, certain limitations such as limited subjects (n = 15). The smart watch technique we developed is convenient, cost-efficient, and scalable.

Introduction

Atopic Dermatitis (AD), commonly known as eczema, is a chronic inflammatory skin condition that affects almost 12 million persons in the United States. It is characterized by scaly skin lesions which periodically flare, causing severe itching. Itching, or pruritus, is the most common symptom of AD that can lead to frequent scratching, painful skin, and skin infections. Scratching can then lead to further disruption and inflammation of the epidermal skin barrier, causing further, more intense itching.[1] This has been described as the itch-scratch cycle [1]. Scratching occurs more frequently at night and can undermine sleep. Further, since patients do most of their scratching at night, they may be unaware of a flare until excessive skin damage has been caused [2]. Since no cure for AD exists, treatments are focused around relieving the symptoms, mainly the itch, to improve quality of life. Early diagnosis and treatment may prevent significant morbidity sleep disturbances, chronic post-inflammatory skin changes, scarring from picking and scratching, and the development of secondary skin infections [1]. In spite of the importance of early diagnosis and treatment to reduce patient morbidity, very few methodologies to objectively assess severity of the disease and progression of treatments exist.

The most widely used index to assess severity of Atopic Dermatitis is SCORAD. The SCORAD is a measure that combines observations of skin lesion characteristics, estimation of the extent of affected area using “rule of nines. This involves dividing parts of the body into sections by multiples of 9% each to measure the total surface area of the body that eczema lesions take up coupled with patient reported symptoms. It is commonly used and often considered the gold standard, but concerns have been raised about its accuracy and inconsistent results due to the subjective nature of the criteria used [3]. In clinical trials for drugs aimed at reducing the itch, video recording is considered the gold standard. Here, scratch is used as a proxy for itch, and the method involves video recording of subjects in sleep labs followed by manual annotation of scratch events by an observer. This process is highly expensive and time-consuming, making it impractical in clinical settings [4].

Wrist Actigraphy is an emerging method for scratch detection. It involves the use of wrist-based accelerometers to monitor wrist movements and classify events of scratch. Wrist Actigraphy is a convenient, cost efficient, and accurate method to assess severity of AD for clinical and non-clinical uses. There have been some promising studies exploring actigraphy for scratch detection, however, each of them has shown to be lacking in certain aspects. Feuerstein et al. [5] used features derived from accelerometer signals to create a K-means clustering model. The study involved 12 healthy subjects performing simulated scratching, walking, and restless sleep movements. The achieved a sensitivity of 0.90 in predicting scratch movements.

However, the test involved simulated scratch movements in healthy patients with almost no variance introduced between subject movements. Further, this study involved subjects scratching for 30 seconds at a time, which is unrealistic in real-life situations, and suggests that the model may not work in realistic environments. The model classified scratch movements, walking, and restless sleep. Shortly after, Petersen et al. [6] used the same data from the previous study to train a binary classifier which would classify only between scratch and non-scratch movements, achieving a higher sensitivity of 0.96, Again, given the highly simulated movements, the authors stated this model was likely to fail in a real-world setting.

To address the shortcomings of the previous studies, Moreau et al [7] conducted a similar study, using a combination of healthy subjects [6] and AD subjects [8]. The study was conducted over 5 nights in a sleep lab, recording patients with an IR camera in a completely unsimulated setup. The IR video was manually scored to annotate scratch events. Twenty-four recordings were collected to build the dataset. Further, this study tested the logistic regression model created by Petersen et al. on the new dataset, and the model achieved a sensitivity of only 0.14, proving the model ineffective for practical use. Moreau trained the model using a Bi-directional RNN model, receiving a combined sensitivity of 0.66. The main drawbacks to this study were misclassification of smoother, less intense scratch movements which fell below the thresholds set by the model. Additionally, there was a general lack of interpretability of Sequential neural networks [8], which may be required in clinical uses. Interpretability of a model refers to the degree that a human can explain the decisions made by a model. The most recent study by Mahadevan et al. proposes a combination of heuristic and machine learning algorithms to predict events of scratch in an unsimulated setup. The data collection involved 33 AD patients recorded via thermal video in a sleep laboratory. The team used a hierarchical approach, involving detection of sleep states, hand movements, and finally presence of scratch movements. It achieved a mean accuracy of 73% and sensitivity of 0.61. The drawbacks noted in the study were similar to Moreau, where the misclassified scratch movements were often smoother or less intense scratch movements.

With the aforementioned studies in mind, we aimed to create a model using signals derived from micro-accelerometers with high generalizability. Since an unsimulated study is not possible due to budgetary limitations, we simulated scratch movements incorporated with confounding movements which are commonly performed during the nighttime, such as typing on a laptop and phone, brushing teeth, washing hands, and combing hair. Further, we introduce variability in the duration and intensity of all scratch and non-scratch movements to improve the generalizability of our model. Additionally, we explored the attachment of accelerometers to the back of the hand as opposed to the wrist, in order to capture the lower intensity wrist and finger dominant scratch motions. The purpose of the study was to establish a proof-of-concept for the application of accelerometers in detecting scratching. To this end, we focused on interpretability when selecting and training our model, so that the results of the model can be explained with great detail to aid further studying of the use of accelerometers in scratch detection. Data was collected from a total of 15 participants, using an Apple Watch Series 7 to extract raw accelerometer data with corresponding timestamps of manually recorded scratching events. The data was annotated using a rudimentary algorithm, and windowed into 2.5 second intervals. Features were then derived from the processed data in the time and frequency domain.

Lastly, we combined the machine learning model with a deterministic algorithm that aggregates individual scratch events into episodes of scratch (for example, all scratch movements less than 3 seconds apart combine to form a scratch episode), to further aid the process of testing effectiveness of medications by measuring the duration of individual scratch events. We demonstrate the most important features derived from the signals and analyze their contributions towards the model's decisions to interpret our model and gain valuable insights. Further, we examined the performance of our model using a range of metrics and analyze the misclassifications and potential causes. We conclude with discussions of limitations to this technology and discuss the possible extensions of our work.

Materials and Methods.

Data Collection

In this study, a total of 15 datasets were collected to be used in the experiment. The data collection process involved using an Apple smartwatch worn by the subjects to extract raw accelerometer data captured motion in the x, y and z axes collected at 40 Hz, with a range of +/- 2g. The data was extracted using an application, Sensor Logger, which exported the data in CSV format to the iPhone. The smartwatch was chosen due to its convenience and portability, making it easy for the subjects to wear it during the data collection process and a convenient option for future use. While an Apple watch was used for the experiment, any smartwatch with accelerometers and access to sensor data can be used.

To ensure the variability and generalizability of the data, the subjects were required to perform simulated scratch movements that incorporated confounding movements. The subjects were given instructions to refrain from performing any scratch or no-scratch movements other than the documented movements prior to the data collection process to ensure accuracy and consistency. The scratch movements included scratches on the hands, neck, and forehead, using different permutations of high intensity, low intensity, small area, and large area scratch movements.

To further improve the variability of the data, the duration of the scratch movements was not fixed but lasted on average between 1-3 seconds. The confounding movements included various hand movements that would commonly occur at nighttime and could be mislabeled as scratch movements such as typing, brushing, combing hair, and washing hands. The subjects were asked to complete 6 random scratch movements separate from the documented movements. This was done to ensure that the dataset included scratch movements that were not part of the documented movements, further increasing the variability of the data.

The data was collected at a frequency of 40 Hz to ensure a high level of detail in the collected data. Forty Hz frequency was selected to minimize dependency of the hardware in the performance of the model. To ensure accuracy and consistency in the data labeling process, a custom-built script was used to annotate the collected data.

Data Processing

Table 1. Representation of raw dataset

timestamp	elapsed (s)	x-axis	y-axis	z-axis
1.66E+09	0	0.337	0.47	0.845
1.657135e+09	0.021	0.313	0.465	0.851
1.657135e+09	0.041	0.309	0.455	0.842
1.657135e+09	0.062	0.315	0.450	0.837
1.657135e+09	0.082	0.320	0.453	0.831

Table 1 displays a representation of the raw data before it is processed. The timestamps are in units of milliseconds, there is a column for elapsed time since start and one column each for the acceleration measured in different axes in units of G-forces (g). The primary Python libraries used for data processing were Pandas and Numpy.

In order to detect scratch movements using accelerometers, a data processing pipeline was developed (Figure 1). The goal of this pipeline was to extract relevant features from the raw accelerometer data that would allow for accurate detection of scratch movements while minimizing the impact of device orientation on the model performance.

The first step in the data processing pipeline was the standardization and compilation of all datasets. This was done to ensure that all data was in a consistent format. The data was manually annotated at the time of the experiment by recording the start and end timestamps of each scratch motion. Table 2 displays an example from the labelling dataset which was used to create the labels.

Table 2. Example from the labelling dataset

start time	**end time**	**Comment**	**Location**
1.657727e+09	1.657727e+09	quit	Left_hand_Fast_Small
1.657727e+09	1.657727e+09	NaN	right_neck
1.657727e+09	1.657727e+09	NaN	left_ neck
1.657727e+09 1.657727e+09	1.657727e+09 1.657727e+09	NaN NaN	forehead right_cheek
1.657727e+09	1.657727e+09	NaN	left_cheek

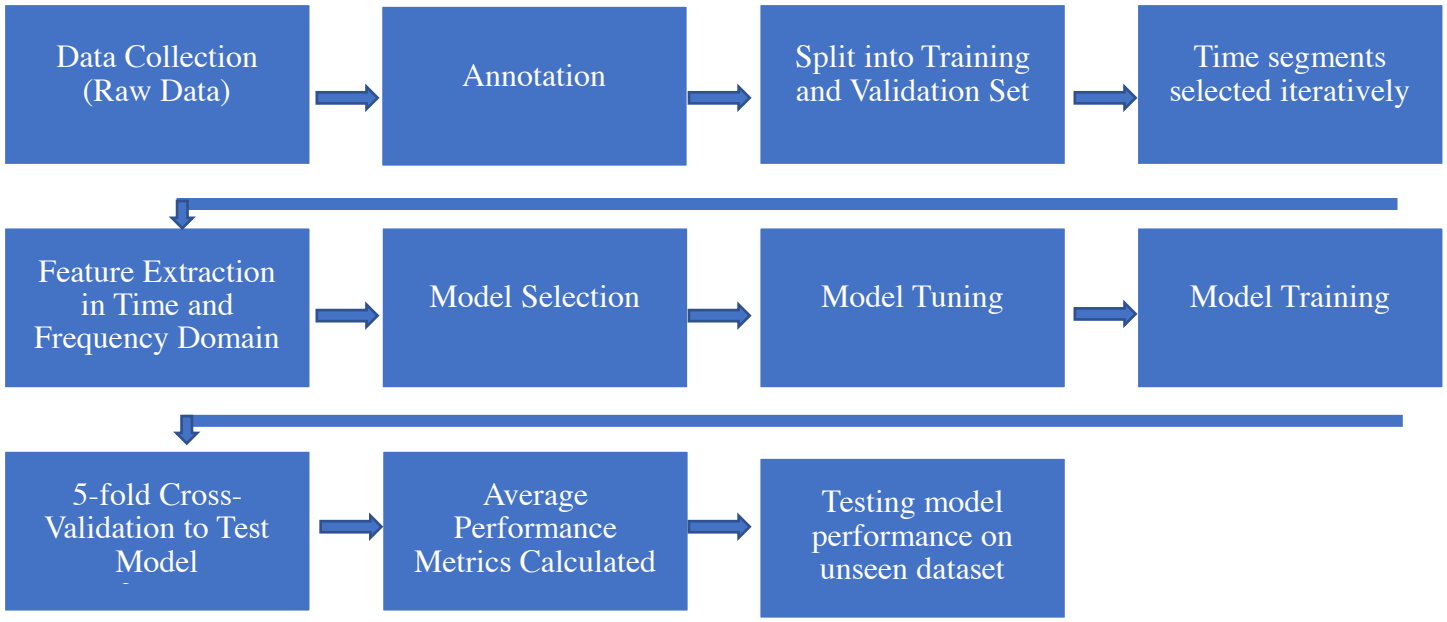


Figure 1 provides a pipeline for the model.

In preprocessing, labels of 1 and 0 were created using the annotated timestamps for each datapoint, 1 signifying positive scratch event. Instead of using the raw accelerometer signals, a decision was made to use the Signal Vector Magnitude (SVM) (computed as $\sqrt{x^2 + y^2 + z^2}$) and 2 principal components. By using these quantities instead of the raw signals, we were able to reduce the effect of orientation on the performance while still capturing maximum variability.

Figure 2 shows the derived components from the signals, with red signifying scratch events and blue signifying non-scratch events. The two principal components capture most of the variability in the data, as shown by large deviations from the baseline amplitude.

To extract relevant features from the data, windows of 2.5 seconds were created with a 50% overlap. This allowed for multiple observations to be made within a single recording, thereby increasing the amount of data available for analysis. The 2.5 second windows which contained approximately 100 datapoints per window allowed us to extract features and trends to be detected in these windows. Features were extracted in both the time and frequency domain. A total of 24 features were derived in the time and frequency domain. Table 3 displays all of the features derived.

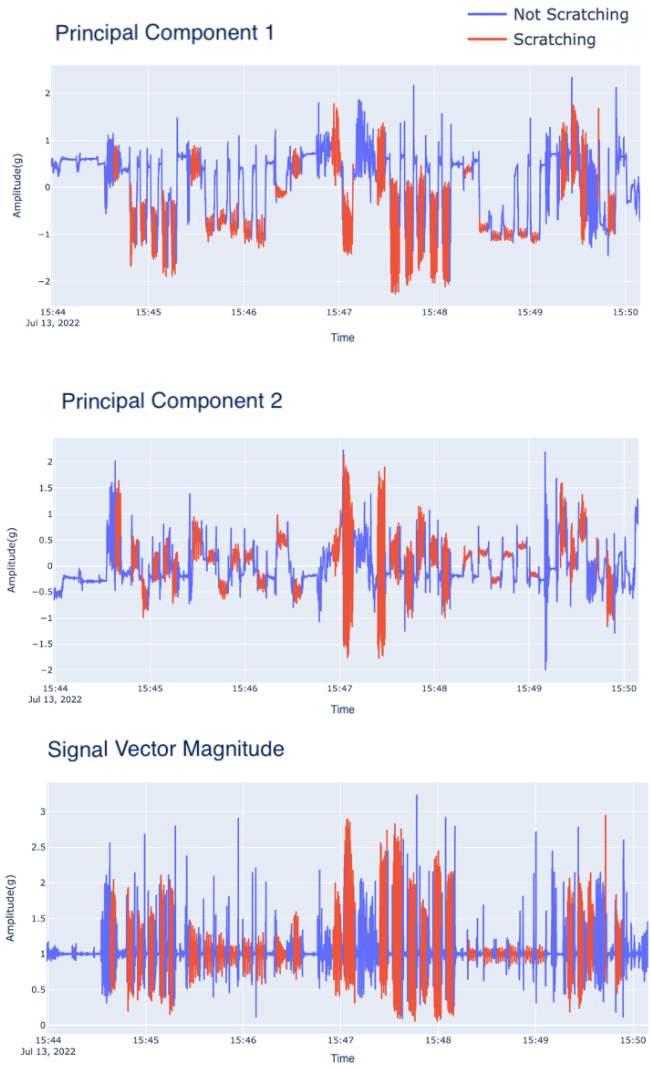


Figure 2. Example time courses of the signal vector magnitude and two principal components in response to scratching stimuli. Scratching time windows are shown in red with periods without scratching are shown in blue

Table 3. List of features derived from time course data

Features List	Description
pc1_skewness	Skewness measures the asymmetry or distortion of the data in a given window.
pc2_skewness	
svm_skewness	
svm_signal_range	Signal Range measures the range of the extremes of motion in a given window.
pc1_acv_IQR	IQR or InterQuartile Range measures the spread of the data in a given window.
pc2_acv_IQR	
svm_acv_IQR	
pc1_mean_cross_rate	Mean Cross Rate computes the mean of the signal in a given window and measures the frequency that the signal crosses this mean value
pc2_mean_cross_rate	
svm_mean_cross_rate	
pc1_zero_cross_rate	Zero Cross Rate measures the frequency that the signal turns from positive to negative or vice-versa in a given window
pc2_zero_cross_rate	
pc1_dom_freq_val	Dominant Frequency Value measures the value of the frequency which has the highest magnitude in the given window.
pc2_dom_freq_val	
pc1_dom_freq_mag	Dominant Frequency Magnitude measures the magnitude of the dominant frequency in a given window.
pc2_dom_freq_mag	
pc1_spectral_arclength	Spectral Arc Length calculates the smoothness of motion in a given window [9]
pc2_spectral_arclength	
svm_spectral_arclength	
pc1_jerk	Dimensionless Jerk calculates the lack of smoothness in motion in a given window. [9]
pc2_jerk	
svm_jerk	
pc1_log_jerk	
pc2_log_jerk	
svm_log_jerk	

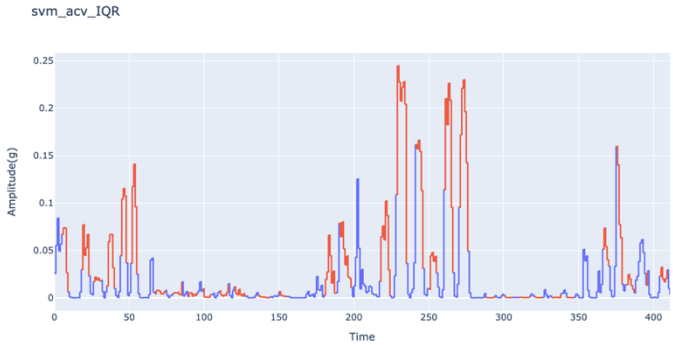


Figure 3. Example time course of the SVM interquartile range for one representative study participant.

Figure 3 shows the graph for SVM Interquartile range identifying events of scratch. As can be seen in the graph, most of the peaks in the IQR signify a scratch event. However, there are some scratch events (true positive cases) which are flat lines in the graph. This points out one area where this feature fails to perform. When compared to the original signal, we note that the flat lines refer to the low intensity scratch movements which we are unable to classify accurately. In that case a feature like log-dimensionless jerk may be better suited for detection of those movements, as given in Figure 4.

Comparing figure 3 to figure 4 we see that log dimensionless jerk of the signal vector magnitude can better detect specific scratch movements than the interquartile range.

Model Selection and Training

In our study, we used scikit-learn to train and compare various machine learning models. We evaluated the performance of logistic regression, decision trees, random forests, adaptive boost, KNN, and SVM. Below is a brief overview of the models we compared:

- Decision trees: a non-parametric model for classification and regression.[12]
- Random forests: an ensemble method that combines multiple decision trees for improved performance. [12]
- K-Nearest Neighbors (KNN): a non-parametric algorithm for classification and regression that predicts the class or value of a new observation based on the closest k neighbors in the training data. [12]
- Support Vector Machines (SVM): a powerful algorithm for classification and regression that works by finding the hyperplane that maximally separates the data points. [12]
- Adaptive boost: an iterative algorithm that builds models sequentially, with each model focusing on the samples that were misclassified by the previous model. [12]
- Logistic regression: a simple linear model for binary classification. [12]

In order to compare the models, we used default hyperparameters and assessed the performance based on the ROC score. ROC score was chosen since it provides a quantitative measure of the overall performance of a binary classification model. Given below in Table 4 is the individual performance of all models based on an 80:20 train-test split.

Table 4. ROC scores for each machine learning model applied in this study.

Model Name	**ROC Score**
Decision Tree	0.80
Random Forest	0.95
K- Nearest Neighbors	0.76
Support Vector Machines	0.76
Adaptive Boost Classifier	0.91
Logistic Regression	0.73

Based on Table 4, the decision was made to choose between Random Forests and Adaptive Boost given the large drop in performance for the other models. Here, Random Forests was picked over Adaptive Boost for two reasons:

1. Random Forests Model is more robust to noise and outliers, and as such, would be better equipped to differentiate between positive and negative events of scratch given the rather high sensitivity of the accelerometers.[10][11]
2. Random Forests can handle high-dimensional data (data with a large number of features) without the need for dimension reduction (we had 24 features).

To further optimize the performance of the Random Forests model, we used a library called *HalvingRandomSearchCV* for hyperparameter tuning. *HalvingRandomSearchCV* is a scikit-learn library used for hyperparameter tuning. It is an implementation of the Successive Halving algorithm, which is a method for hyperparameter optimization that reduces the number of evaluated models as the optimization progresses. It works by training a large number of models with random hyperparameter settings, and then selecting a subset of the models based on their performance. The selected models are then trained on more data, and the process is repeated until a single model is selected as the final model. This approach can be much more efficient than traditional grid search methods, as it avoids evaluating many poorly performing models.

Using *HalvingRandomSearchCV*, we were able to identify the optimal set of hyperparameters for our Random Forests model, which improved the model's performance.

The parameters chosen for the model are:

Bootstrap = 'True': Each decision tree in the random forest is be built on a random subset of the training data, with replacement. This is done to reduce overfitting and computational cost.[12]

criterion='*Gini*': 'Gini' refers to the Gini impurity, which measures how often a randomly chosen element would be incorrectly classified if

it were randomly labeled according to the distribution of labels in the set. The Gini impurity is used as the criteria to evaluate the quality of each split [12].

max_depth=*80*: This parameter limits the maximum depth of the tree. A deeper tree can capture more complex relationships in the data, but it can also lead to overfitting. The value for this parameter was chosen based on our dataset and the performance of the model.[12]

max_features= *'sqrt'*: This parameter determines the number of features to consider when looking for the best split. In this case, algorithm will consider the square root of the total number of feature [12].

min_samples_leaf=*2*: This parameter specifies the minimum number of samples required to be at a leaf node. A smaller value enables the model to be more flexible at the risk of overfitting [12].

n_estimators=*2000*: This parameter determines the number of trees in the forest. A higher number of trees is able to improve the performance of the model, but also increases the training time [12].

Model Testing

The model was tested using an 80:20 train-test split. To evaluate the model, we primarily chose ROC AUC (Receiver Operating Characteristic Area Under Curve) as our scoring metric. It evaluates the performance of a classifier by measuring the area under the ROC curve, which plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification thresholds. The reason this was chosen is because it provides a comprehensive evaluation of the classifier; ROC AUC summarizes the performance of a classifier over all possible classification thresholds. This means that it provides a more complete picture of how well the classifier is doing than other metrics that only look at one specific threshold.

The other metrics that were used to test the performance of the model include:

Accuracy – Refers to the correct predictions made by the model relative to all the predictions made.

F1 Score – It is the harmonic mean of the precision (Positive Predictive Value) and recall (Sensitivity) of a test.

Sensitivity – Percentage of True Positives

Specificity – Percentage of True Negatives

Positive Predictive Value – Refers to the proportion of positively classified cases that were truly positive.

Negative Predictive Value - Refers to the proportion of negatively classified cases that were truly negative.

Results

Given the data is in time-series format and highly correlated, an 80:20 split was chosen over cross-validation in the evaluation of the model. The 80:20 split involves splitting the dataset into a training and testing set. 80% of the data is used for training and 20% is kept aside for testing.

Given below in Table 5 are the performance metrics on the testing set.

Table 5. Performance Metrics Based on Testing Set.

Performance Metric	Score
AUROC	0.95
Accuracy Score	0.88
F1 Score	0.84
Sensitivity	0.83
Specificity	0.91
Positive Predictive Value	0.85
Negative Predictive Value	0.90

Based on our test set performance, we have promising results. An AUROC score of 95% suggests that the model can sufficiently differentiate between positive and negative samples.

The AUROC and Accuracy give us the measure of performance of the model on a general level, while the other metrics allow us to better understand the model's ability to predict samples correctly.

The lower value of sensitivity suggests that the model is struggling to accurately find all the true positives. However, the specificity and negative predictive value suggest that the model correctly identifies non scratch events 90% of the times it identifies an event as non-scratch, and out of all the true non-scratch events, the model identified 91%

correctly. This leads us to infer that there are certain movements of scratch which our model fails to identify.

We then performed an in-depth feature analysis to give us further insights about the working of the model.

Feature Analysis

Shown below in Fig 5 is the SHAP Summary Plot [9]. This is a form of visualization which displays the feature importance of the various features contributing to a model. SHAP stands for SHapley Additive exPlanations, and it is a method for interpreting the output of complex models.

The SHAP summary plot shows the importance of each feature in the model's output, by plotting the features on the y-axis and the SHAP values on the x-axis. The SHAP value represents the contribution of each feature to the final prediction made by the model.

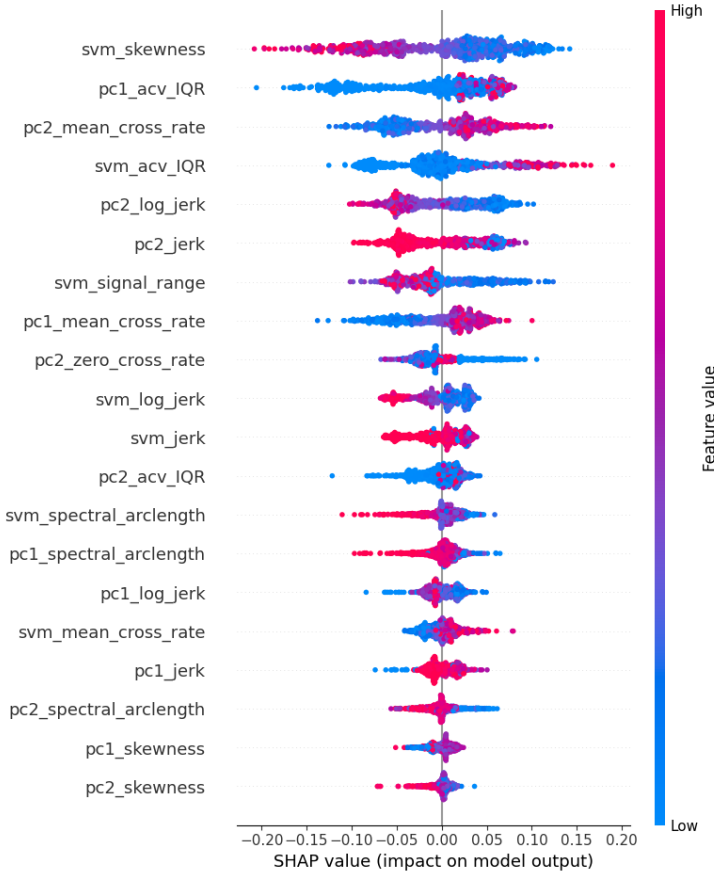


Figure 5. SHAP summary plot displaying the relative contribution of each feature in the model.

The range on the x-axis refers to the contribution made by the certain feature towards the predictions of the model, and red color suggests a higher feature value and blue suggests a lower feature value.

From Figure 5 we see that the *skewness* is the most important feature in the model's performance. We can interpret that the skewness has a negative correlation to the model's predictions. That is, a lower value of skewness leads to a prediction of scratch and a higher value of skewness leads to a prediction of non-scratch. Intuitively, this tells us that the scratch movements are more symmetric and generally the hand movements in the event of scratch are periodic without much deviation.

Conversely, the *mean cross rate* has a direct relationship with the model predictions. A higher mean cross rate leads to a positive scratch prediction and a lower mean cross rate leads to a negative scratch prediction. The mean cross rate does a more robust job at classifying scratch movements when compared to the zero cross rate since it accounts for varying intensities of motion.

We observed that lower values *Spectral Arc Length* leads to a positive prediction of scratch, suggesting that scratch movements are less smooth compared to confounding movements.

The feature analysis suggests that the model is more likely to predict a movement as a scratch movement if the motion is a continuous, periodic, less smooth, motion with a high frequency and amplitude of oscillation.

Testing on Unseen Dataset.

A new dataset was used to test the model for overfitting and generalizability. The performance metrics are given below in Table 6. As expected, there was a slight drop in performance compared to the original set, however, the model performed moderately well. (Note, the data from the new dataset was collected in the exact same method as the previous sets).

Discussion

We explored the use of wrist-based accelerometers to detect scratching movements in individuals. The goal of the study was to assess the viability of using Machine Learning in combination with smartwatches to measure the progression of Atopic

Dermatitis (Eczema) by capturing the number of scratch events over time.

Table 6. Comparison of 80:20 Split Performance vs Performance on Unseen Set

Metric	Unseen Set Performance	Original Performance
AUROC	0.90	0.95
Accuracy Score	0.85	0.88
F1 Score	0.83	0.84
Sensitivity	0.82	0.83
Specificity	0.87	0.91
Positive Predictive Value	0.83	0.85
Negative Predictive Value	0.86	0.90

Fifteen datasets were collected for the study consisting of simulated scsratch and other confounding movements. Raw accelerometer data was extracted from wrist-worn smartwatches at a 40 Hz rate. From the annotated data, we derived various signal-based features and trained a binary classifier. We then tested the performance of the classifier based on a train-test split as well as on an unseen dataset. A thorough retroactive feature analysis gave us valuable insights towards the behavior of the model

Based on Table 6, our model performs well on the validation and test sets. The nominal drop in performance suggests that the model is not overfitted and performs adequately with new unseen data. However, all data consisted of simulated scratch movements from healthy subjects, and the results would likely be different in the case of true scratch motions from AD patients.

Overall, this suggests that scratch detection using wrist-based accelerometers is viable on simulated scratch movements, however, these results may differ in events of true scratch movements in real life scenarios.

However, looking at the metrics, there are some areas where the performance of the model lacks, specifically in the case of false negatives. We saw that in Fig 5 the Interquartile range is an important feature in the model's predictions. As we note in Fig 4, the graph showing the amplitude of IQR over time, there are some events of true scratch with an IQR value of close to 0. We suspect that these are the cases of low-amplitude, finger dominant scratch movements that are being misclassified as false negatives. Unfortunately, there is no simple fix to this since increasing the sensitivity of the accelerometers would introduce a large amount of noise which would greatly affect the accuracy of the model. This points to a ceiling effect, as suggested by Mahadevan et al. [9] on the use of wrist-based accelerometers.

There are a few methods however, that we believe may address this issue, requiring further research. The first is a hierarchical model which separates the low intensity movements from the higher intensity movements and uses differently tuned models to classify them. As discussed in the feature analysis, presently, the model classifies scratch movements as continuous, periodic, less smooth, highly oscillating movements. Since most scratch movements fit in this category, the model does a good job overall, however, to accurately predict all events of scratch, a second model may be tuned to detect the low intensity, smoother scratch movements. A second possible improvement is to in the location placement of the accelerometer, such as the back of the hand, to capture the less intense movements more accurately. In this case, instead of a smartwatch, the device could be a modified glove with fingers cut-out and a mounted accelerometer, as illustrated below in Fig 6.

Figure 6. (Taken from FoxGloves)

A third option is a glove fitted with accelerometer and flex sensors to measure the bending of fingers, combined with a hierarchical model which would check for hand movements and pick the sensor to be used for the prediction based on a heuristic algorithm.

Our study is subject to a number of limitations. Firstly, the dataset used for the study was generated through simulated scratch movements. Although our findings are promising, we expect that the scratch motions of individuals with Atopic Dermatitis may differ from those captured in our dataset. Secondly, our experiment was conducted using only one type of smartwatch, and additional testing may be needed to assess the versatility of our approach across different hardware platforms. Thirdly, our sample size was relatively small, consisting of only 15 subjects. Therefore, larger-scale studies involving more participants are necessary to evaluate the effectiveness of our method in a clinical setting.

All the alternate methods provided above maintain the simplicity of use for the end user, as well as maintaining cost effectiveness for the researcher. The end goal remains the use of machine learning and accelerometers to detect scratching in a convenient, cost-efficient, and scalable manner.

In conclusion we found that the despite the limitations, our proposed method presents a promising proof of concept for utilizing wrist-based accelerometers for scratch detection.

Acknowledgements

I'm extremely grateful to Dr. Jack Virostko for his guidance, support and invaluable insights throughout the course of the research paper.

I would like to extend my sincere thanks to Dr. Nandkumar Selvaraj and Sylvia Zhang for their contributions toward my practical knowledge on the subject.

Literature Cited

1. Berke, Rebecca, Arshdeep Singh, and Mark Guralnick. "Atopic Dermatitis: An Overview." American Family Physician, July 1, 2012. https://www.aafp.org/pubs/afp/issues/2012/0701/p35.html#afp20120701p35-b5.

2. K. L. Hon, M. C. Lam, T. F. Leung, C. M. Chow, E. Wong, and A. K. Leung, "Assessing itch in children with atopic dermatitis treated with tacrolimus: objective versus subjective assessment," Adv Ther, vol. 24, pp. 23-8, 2007.

3. Angelova-Fischer I;Bauer A;Hipler UC;Petrov I;Kazandjieva J;Bruckner T;Diepgen T;Tsankov N;Williams M;Fischer TW;Elsner P;Fluhr JW; "The Objective Severity Assessment of Atopic Dermatitis (Osaad) Score: Validity, Reliability and Sensitivity in Adult Patients with Atopic Dermatitis." The British journal of dermatology. U.S. National Library of Medicine. Accessed April 3, 2023. https://pubmed.ncbi.nlm.nih.gov/16181458/.

4. Yang AF;Nguyen M;Li AW;Lee B;Chun KS;Wu E;Fishbein AB;Paller AS;Xu S; "Use of Technology for the Objective Evaluation of Scratching Behavior: A Systematic Review." JAAD international. U.S. National Library of Medicine. Accessed April 3, 2023. https://pubmed.ncbi.nlm.nih.gov/34816131/.

5. TL;, Feuerstein J;Austin D;Sack R;Hayes. "Wrist Actigraphy for Scratch Detection in the Presence of Confounding Activities." Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference. U.S. National Library of Medicine. Accessed April 3, 2023. https://pubmed.ncbi.nlm.nih.gov/22255131/.

6. Yang AF;Nguyen M;Li AW;Lee B;Chun KS;Wu E;Fishbein AB;Paller AS;Xu S; "Use of Technology for the Objective Evaluation of Scratching Behavior: A Systematic Review." JAAD international. U.S. National Library of Medicine. Accessed April 3, 2023. https://pubmed.ncbi.nlm.nih.gov/34816131/.

7. Moreau A;Anderer P;Ross M;Cerny A;Almazan TH;Peterson B;Moreau

A;Anderer P;Ross M;Cerny A;Almazan TH;Peterson B; “Detection of Nocturnal Scratching Movements in Patients with Atopic Dermatitis Using Accelerometers and Recurrent Neural Networks.” *IEEE Journal of Biomedical and Health Informatics*, U.S. National Library of Medicine, https://pubmed.ncbi.nlm.nih.gov/28613187/.

8. Gandin, Ilaria, et al. “Interpretability of Time-Series Deep Learning Models: A Study in Cardiovascular Patients Admitted to Intensive Care Unit.” *Journal of Biomedical Informatics*, Academic Press, 27 July 2021, https://www.sciencedirect.com/science/article/pii/S1532046421002057.

9. Mahadevan, Nikhil, et al. “Development of Digital Measures for Nighttime Scratch and Sleep Using Wrist-Worn Wearable Devices.” *Nature News*, Nature Publishing Group, 3 Mar. 2021, https://www.nature.com/articles/s41746-021-00402-x.

10. Breiman, Leo. “Random Forests - Machine Learning.” *SpringerLink*, Kluwer Academic Publishers, https://link.springer.com/article/10.1023/a:1010933404324.

11. Brownlee, Jason. *Long Short-Term Memory Networks with Python: Develop Sequence Prediction Models with Deep Learning*. Jason Brownlee, 2017.

12. Pedregosa, Fabian, et al. “Scikit-Learn: Machine Learning in Python.” *Journal of Machine Learning Research*, 1970, https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html.

13. Raymond, R. (n.d.). python - Plotly: one line, different colors. Stack Overflow. Retrieved April 18, 2023, from https://stackoverflow.com/questions/69705455/plotly-one-line-different-colors

[redacted]

In [2]:
```python
import numpy as np
import pandas as pd
from scipy import stats
from scipy import signal
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.io as pio
import plotly.graph_objects as go
import datetime
import inspect
from sklearn.metrics import confusion_matrix
import tensorflow as tf
import os
import sys
from sklearn.metrics import classification_report
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn import metrics
import plotly.io as pio
pio.renderers.default='notebook'
import sys
import os
```

```
2023-04-13 15:39:14.921973: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to
use the following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

In [3]:
```python
def read_dataframe(path):
    df = pd.read_csv(path)
    df0 = pd.read_csv(path)
    df = df.rename(columns = {"x-axis (g)" : "x-axis" ,"y-axis (g)" : "y-axis", "z-axis (g)" : "z-axis", "epoc (ms)" : "timestamp" })
    df["pdtime"] = pd.to_datetime(df["timestamp"], unit = "ms")
    df["time"] = df["timestamp"].apply(lambda x: datetime.datetime.fromtimestamp(x/1000).time())
    df["timestamp"] = df['timestamp'].apply(lambda x : x /1000)
    df = df.drop(columns = ["timestamp (-0400)"])
    df.resample("32ms", on = "pdtime").mean().reset_index().drop(columns = ["pdtime"])
    df["time"] = pd.to_datetime(df["timestamp"], unit = "s")

    return df, df0
```

In [4]:
```python
fileno = [str (i) for i in range(1,7)]
run = [1, 2]
for r in fileno:
    for p in run:
        try:
            filepath = "/Users/adi/Desktop/Scratch_Model/Scratch_Simulation_Study/Raw_Data/data{}_{}.csv".format(r,p)
            df, df0= read_dataframe(filepath)


        except FileNotFoundError:

            try:

                filepath = "/Users/adi/Desktop/Scratch_Model/Scratch_Simulation_Study/Raw_Data/data{}_1+2.csv".format(r)
                df, df0= read_dataframe(filepath)

            except FileNotFoundError:
                continue

        df["time"] = pd.to_datetime(df["timestamp"], unit = "s")

        df




        df_labels = pd.read_csv("/Users/adi/Desktop/Scratch_Model/Scratch_Simulation_Study/Raw_Data/Timestamps_data{}_{}.csv".format(r,p))
        df_labels


        ## Write the indexes of the rows to be skipped in the line below. If none, comment out the line.
        if int(r) > 1 :
            df_labels = df_labels.drop(df_labels.index[[12,13,14,15]]).reset_index()
        df_labels = df_labels[df_labels["Annotation"].isna()].reset_index()
        assert(len(df_labels["Annotation"].unique()) == 1)



        from sklearn.decomposition import PCA
        pca = PCA(n_components = 2)
        pca.fit(df[["x-axis" , "y-axis", "z-axis"]])
        pca_arr = pca.transform(df[["x-axis" , "y-axis", "z-axis"]])


        df["PC1"] = np.array(pd.DataFrame(pca_arr)[0])
        df["PC2"] = np.array(pd.DataFrame(pca_arr)[1])
        #df["PC3"] = np.array(pd.DataFrame(pca_arr)[2])





        df["SVM"] = np.sqrt(df["x-axis"].apply(lambda x : x**2)
                            + df["y-axis"].apply(lambda x : x**2)
                            + df["z-axis"].apply(lambda x : x**2))



        def create_labels(training_df,df_labels ):


        #training_df : dataframe containing the x,y z from accelerometer.

        #df_labels : dataframe generated from timestamper.py script


            label_arr = (np.zeros(len(training_df), dtype = np.int8))
            is_first = True
            for i in range(len(df_labels)):
                for j in range(len(training_df)):
                    if (training_df["timestamp"].loc[j] >= (df_labels["start_time"].loc[i] )
                        and training_df["timestamp"].loc[j] <= (df_labels["end_time"].loc[i])):

                        label_arr[j] = int(1)

                        idx = j

                        if is_first:
                            start_idx = j
                            is_first = False

            df["label"] = label_arr
            df0 = df.loc[start_idx - 200:idx+150].reset_index()



            return df0



        df = create_labels(df, df_labels)




        pos_df = df.loc[(df['label'] == 1)]
        neg_df = df.loc[(df['label'] != 1)]



        a = 75
        b = 38


        pc1_list = []
        pc2_list = []
        #pc3_list = []
        svm_list = []
        x_list = []
        y_list = []
        z_list = []
        labels = []
        times = []
        window_size = a
        step_size = b

        for i in range(0, len(pos_df) - window_size, step_size):
            pc1tmp = pos_df["PC1"].values[i : i + window_size]
            pc2tmp = pos_df["PC2"].values[i : i + window_size]
            #pc3tmp = pos_df["PC3"].values[i : i + window_size]
            svmtmp = pos_df["SVM"].values[i : i + window_size]
            xtmp = pos_df["x-axis"].values[i : i + window_size]
            ytmp = pos_df["y-axis"].values[i : i + window_size]
            ztmp = pos_df["z-axis"].values[i : i + window_size]
            time = pos_df["time"].values[i : i + window_size]
            label = 1

            pc1_list.append(pc1tmp)
            pc2_list.append(pc2tmp)
           # pc3_list.append(pc2tmp)
            svm_list.append(svmtmp)
            x_list.append(xtmp)
            y_list.append(ytmp)
            z_list.append(ztmp)
            labels.append(int(label))
            times.append(time)


        ## Each element in the list is an array consisting 60 values totalling .5 seconds.


        pc1_list_1 = []
        pc2_list_1 = []
        #pc3_list_1 = []
        svm_list_1 = []
        x_list_1 = []
        y_list_1 = []
        z_list_1 = []
        labels_1 = []
        times_1 = []


        for i in range(0, len(neg_df) - window_size, step_size):
            pc1tmp = neg_df["PC1"].values[i : i + window_size]
            pc2tmp = neg_df["PC2"].values[i : i + window_size]
            #pc3tmp = neg_df["PC3"].values[i : i + window_size]
            svmtmp = neg_df["SVM"].values[i : i + window_size]
            xtmp = neg_df["x-axis"].values[i : i + window_size]
            ytmp = neg_df["y-axis"].values[i : i + window_size]
            ztmp = neg_df["z-axis"].values[i : i + window_size]
            time = neg_df["time"].values[i : i + window_size]
            label = 0

            pc1_list_1.append(pc1tmp)
            pc2_list_1.append(pc2tmp)
           # pc3_list_1.append(pc2tmp)
            svm_list_1.append(svmtmp)
            x_list_1.append(xtmp)
            y_list_1.append(ytmp)
            z_list_1.append(ztmp)
            labels_1.append(int(label))
            times_1.append(time)


        ## Each element in the list is an array consisting 60 values totalling .5 seconds.

        pos_labels = np.ones(len(pc1_list), dtype = np.int8)
        neg_labels = np.zeros(len(pc1_list_1), dtype = np.int8)
        labels = np.concatenate((pos_labels,neg_labels))
        for i in pc1_list_1:
            pc1_list.append(i)

        for i in pc2_list_1:
            pc2_list.append(i)

        #for i in pc3_list_1:
        #    pc3_list.append(i)

        for i in svm_list_1:
            svm_list.append(i)

        df_train = pd.DataFrame()


        # Measure of assymetry in signal
        df_train["pc1_skewness"] = pd.Series(pc1_list).apply(lambda x : stats.skew(x))
        df_train["pc2_skewness"] = pd.Series(pc2_list).apply(lambda x : stats.skew(x))
        #df_train["pc3_skewness"] = pd.Series(pc3_list).apply(lambda x : stats.skew(x))
        df_train["svm_skewness"] = pd.Series(svm_list).apply(lambda x : stats.skew(x))

        # Diference between extreme values

        df_train["svm_signal_range"] = pd.Series(svm_list).apply(lambda x: x.max() - x.min())



        # Inter-quartile Range of Auto-Covariance
        from  statsmodels.tsa.stattools import acovf
        pc1_acv = pd.Series(pc1_list).apply(acovf)
        pc2_acv = pd.Series(pc2_list).apply(acovf)
        #pc3_acv = pd.Series(pc3_list).apply(acovf)

        svm_acv = pd.Series(svm_list).apply(acovf)
        x_acv = pd.Series(x_list).apply(acovf)
        y_acv = pd.Series(y_list).apply(acovf)
        z_acv = pd.Series(z_list).apply(acovf)


        df_train['pc1_acv_IQR'] = pc1_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
        df_train['pc2_acv_IQR'] = pc2_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
        #df_train['pc3_acv_IQR'] = pc3_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))

        df_train['svm_acv_IQR'] = svm_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
        #df_train['x-axis_acv_IQR'] = x_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
        #df_train['y-axis_acv_IQR'] = y_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
        #df_train['z-axis_acv_IQR'] = z_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))


        def zero_cross(lst):
            count = 0
            psv = True
            for i in lst:
                if i > 0:
                    if not psv:
                        count += 1
                        psv = True
                    else:
                        continue
                elif i < 0:
                    if psv:
                        count += 1
                        psv = False
                    else:
                        continue
            return count/len(lst)

        def mean_cross(lst):
            count = 0
            psv = True
            for i in lst:
                if i > np.mean(lst):
                    if not psv:
                        count += 1
                        psv = True
                    else:
                        continue
                elif i < np.mean(lst):
                    if psv:
                        count += 1
                        psv = False
                    else:
                        continue
            return count/len(lst)



        # HIGH mean cross rate values indicate high frequency movements in short duration, higher probability of scratch.
        df_train["pc1_mean_cross_rate"] = pd.Series(pc1_list).apply(mean_cross)
        df_train["pc2_mean_cross_rate"] = pd.Series(pc2_list).apply(mean_cross)
        #df_train["pc3_mean_cross_rate"] = pd.Series(pc3_list).apply(mean_cross)
        df_train["svm_mean_cross_rate"] = pd.Series(svm_list).apply(mean_cross)
        df_train["pc1_zero_cross_rate"] = pd.Series(pc1_list).apply(zero_cross)
        df_train["pc2_zero_cross_rate"] = pd.Series(pc2_list).apply(zero_cross)
        #df_train["pc3_zero_cross_rate"] = pd.Series(pc3_list).apply(zero_cross)

        #df_train["x-axis_mean_cross_rate"] = pd.Series(x_list).apply(lambda x : stats.skew(x))
        #df_train["y-axis_mean_cross_rate"] = pd.Series(y_list).apply(lambda x : stats.skew(x))
        #df_train["z-axis_mean_cross_rate"] = pd.Series(z_list).apply(lambda x : stats.skew(x))

        def dom_freq_val(lst):
            w = np.fft.fft(lst)
            w = w/len(w)
            freqs = np.fft.fftfreq(len(lst))
            i = np.argmax(abs(w))
            dom_freq = freqs[i]
            dom_freq_hz = abs(dom_freq * 25)
            return dom_freq_hz

        #df_train['pc1_dom_freq_val'] = pd.Series(pc1_list).apply(dom_freq_val)
        #df_train['pc2_dom_freq_val'] = pd.Series(pc2_list).apply(dom_freq_val)


        def dom_freq_mag(lst):
            w = np.fft.fft(lst)
            val = dom_freq_val(lst)
            return (val/np.sum(abs(np.delete(w, np.argmax(abs(w))))))*100



        df_train['pc1_dom_freq_val'] = pd.Series(pc1_list).apply(dom_freq_val)
        df_train['pc2_dom_freq_val'] = pd.Series(pc2_list).apply(dom_freq_val)
        # df_train['pc3_dom_freq_val'] = pd.Series(pc3_list).apply(dom_freq_val)

        df_train["pc1_dom_freq_mag"] = pd.Series(pc1_list).apply(dom_freq_mag)
        df_train["pc2_dom_freq_mag"] = pd.Series(pc2_list).apply(dom_freq_mag)
        #df_train["pc3_dom_freq_mag"] = pd.Series(pc3_list).apply(dom_freq_mag)

        def spectral_arclength(movement, fs = 0.05, padlevel=4, fc=10.0, amp_th=0.05):

            # Number of zeros to be padded.
            nfft = int(pow(2, np.ceil(np.log2(len(movement))) + padlevel))

            # Frequency
            f = np.arange(0, fs, fs/nfft)
            # Normalized magnitude spectrum
            Mf = abs(np.fft.fft(movement, nfft))
            Mf = Mf/max(Mf)

            # Indices to choose only the spectrum within the given cut off frequency Fc.
            # NOTE: This is a low pass filtering operation to get rid of high frequency
            # noise from affecting the next step (amplitude threshold based cut off for
            # arc length calculation).
            fc_inx = ((f <= fc)*1).nonzero()
            f_sel = f[fc_inx]
            Mf_sel = Mf[fc_inx]

            # Choose the amplitude threshold based cut off frequency.
            # Index of the last point on the magnitude spectrum that is greater than
            # or equal to the amplitude threshold.
            inx = ((Mf_sel >= amp_th)*1).nonzero()[0]
            fc_inx = range(inx[0], inx[-1]+1)
            f_sel = f_sel[fc_inx]
            Mf_sel = Mf_sel[fc_inx]

            # Calculate arc length
            new_sal = -sum(np.sqrt(pow(np.diff(f_sel)/(f_sel[-1] - f_sel[0]), 2) +
                                   pow(np.diff(Mf_sel), 2)))
            return new_sal#, (f, Mf), (f_sel, Mf_sel)


        df_train["pc1_spectral_arclength"] = (pd.Series(pc1_list).apply(spectral_arclength))
        df_train["pc2_spectral_arclength"] = (pd.Series(pc2_list).apply(spectral_arclength))
        #df_train["pc3_spectral_arclength"] = (pd.Series(pc3_list).apply(spectral_arclength))

        df_train["svm_spectral_arclength"] = (pd.Series(svm_list).apply(spectral_arclength))


        def dimensionless_jerk(movement, fs = 0.05):

            # first enforce data into an numpy array.
            movement = np.array(movement)

            # calculate the scale factor and jerk.
            movement_peak = max(abs(movement))
            dt = 1./fs
            movement_dur = len(movement)*dt
            jerk = np.diff(movement, 2)/pow(dt, 2)
            scale = pow(movement_dur, 3)/pow(movement_peak, 2)

            # estimate dj
            return - scale * sum(pow(jerk, 2)) * dt

        df_train["pc1_jerk"] = pd.Series(pc1_list).apply(dimensionless_jerk)
        df_train["pc2_jerk"] = pd.Series(pc2_list).apply(dimensionless_jerk)
        # df_train["pc3_jerk"] = pd.Series(pc3_list).apply(dimensionless_jerk)
        df_train["svm_jerk"] = pd.Series(svm_list).apply(dimensionless_jerk)

        def log_dimensionless_jerk(movement, fs = 0.05):
            return -np.log(abs(dimensionless_jerk(movement, fs)))

        df_train["pc1_log_jerk"] = pd.Series(pc1_list).apply(log_dimensionless_jerk)
        df_train["pc2_log_jerk"] = pd.Series(pc2_list).apply(log_dimensionless_jerk)
        #df_train["pc3_log_jerk"] = pd.Series(pc3_list).apply(log_dimensionless_jerk)

        df_train["svm_log_jerk"] = pd.Series(svm_list).apply(log_dimensionless_jerk)

        df_train["label"]  = labels
        count = 0
        while True:

                filepath = "Processed_Data/data{}.csv".format(count)


                if os.path.exists(os.path.join(sys.path[0],filepath)):

                    count += 1


                    continue


                df_train.to_csv(os.path.join(sys.path[0], filepath))

                break

        print(r,p)
```

```
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

1 2
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

2 1
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

2 2
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

3 1
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

3 2
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

4 1
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

4 2
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

5 1
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

5 2
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

6 1
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86582/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select
only columns which should be valid for the function.

6 2
```

In [ ]:

In [1]:
```
import numpy as np
import pandas as pd
from scipy import stats
from scipy import signal
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.io as pio
import plotly.graph_objects as go
import datetime
import inspect
from sklearn.metrics import confusion_matrix
import tensorflow as tf
import os
import sys
from sklearn.metrics import classification_report
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn import metrics
import plotly.io as pio
pio.renderers.default='notebook'
import sys
import os
```

```
2023-04-13 15:49:48.430574: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is opti
mized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-cri
tical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

In [2]:
```
def read_dataframe(path):
    df = pd.read_csv(path)
    df0 = pd.read_csv(path)
    df = df.rename(columns = {"x-axis (g)" : "x-axis" ,"y-axis (g)" : "y-axis", "z-axis (g)" : "z-axis", "epoc
    df["pdtime"] = pd.to_datetime(df["timestamp"], unit = "ms")
    df["time"] = df["timestamp"].apply(lambda x: datetime.datetime.fromtimestamp(x/1000).time())
    df["timestamp"] = df['timestamp'].apply(lambda x : x /1000)
    df = df.drop(columns = ["timestamp (-0400)"])
    df.resample("32ms", on = "pdtime").mean().reset_index().drop(columns = ["pdtime"])
    df["time"] = pd.to_datetime(df["timestamp"], unit = "s")

    return df, df0
```

In [6]:
```
filepath = "/Users/adi/Desktop/Scratch_Model/Scratch_Simulation_Study/Raw_Data/data{}_{}.csv".format("2","2")
df, df0= read_dataframe(filepath)

df_labels = pd.read_csv("/Users/adi/Desktop/Scratch_Model/Scratch_Simulation_Study/Raw_Data/Timestamps_data{}_{
```

```
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86560/1922652035.py:9: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only wil
l default to False. Either specify numeric_only or select only columns which should be valid for the function.
```

In [7]:
```
df
```

Out[7]:

	timestamp	elapsed (s)	x-axis	y-axis	z-axis	pdtime	time
0	1.657727e+09	0.000	0.065	-0.431	0.918	2022-07-13 15:43:58.814	2022-07-13 15:43:58.813999872
1	1.657727e+09	0.021	0.059	-0.435	0.907	2022-07-13 15:43:58.835	2022-07-13 15:43:58.835000064
2	1.657727e+09	0.041	0.055	-0.400	0.891	2022-07-13 15:43:58.855	2022-07-13 15:43:58.855000064
3	1.657727e+09	0.062	0.064	-0.397	0.886	2022-07-13 15:43:58.876	2022-07-13 15:43:58.876000000
4	1.657727e+09	0.082	0.062	-0.381	0.901	2022-07-13 15:43:58.896	2022-07-13 15:43:58.896000000
...	...	...	...	...	...	...	...
18059	1.657727e+09	370.398	0.697	0.131	-0.641	2022-07-13 15:50:09.212	2022-07-13 15:50:09.212000000
18060	1.657727e+09	370.418	0.397	0.160	-1.048	2022-07-13 15:50:09.232	2022-07-13 15:50:09.232000000
18061	1.657727e+09	370.439	0.338	0.240	-1.071	2022-07-13 15:50:09.253	2022-07-13 15:50:09.252999936
18062	1.657727e+09	370.459	0.400	0.193	-0.969	2022-07-13 15:50:09.273	2022-07-13 15:50:09.272999936
18063	1.657727e+09	370.480	0.455	0.210	-0.899	2022-07-13 15:50:09.294	2022-07-13 15:50:09.293999872

18064 rows × 7 columns

In [8]:
```
df_labels
```

Out[8]:

	Unnamed: 0	start_time	end_time	Annotation	Location
0	0	1.657727e+09	1.657727e+09	quit	Left_hand_Fast_Small
1	1	1.657727e+09	1.657727e+09	NaN	right_neck
2	2	1.657727e+09	1.657727e+09	NaN	left_neck
3	3	1.657727e+09	1.657727e+09	NaN	forehead
4	4	1.657727e+09	1.657727e+09	NaN	right_cheek
5	5	1.657727e+09	1.657727e+09	NaN	left_cheek
6	6	1.657727e+09	1.657727e+09	NaN	Left_hand_Slow_Large
7	7	1.657727e+09	1.657727e+09	NaN	right_neck
8	8	1.657727e+09	1.657727e+09	NaN	left_neck
9	9	1.657727e+09	1.657727e+09	NaN	forehead
10	10	1.657727e+09	1.657727e+09	NaN	right_cheek
11	11	1.657727e+09	1.657727e+09	NaN	left_cheek
12	12	1.657727e+09	1.657727e+09	NaN	Typing_Phone
13	13	1.657727e+09	1.657727e+09	NaN	typing_laptop
14	14	1.657727e+09	1.657727e+09	walk	wash_hands
15	15	1.657727e+09	1.657727e+09	wash hands	Brush
16	16	1.657727e+09	1.657727e+09	NaN	Left_hand_Fast_Large
17	17	1.657727e+09	1.657727e+09	NaN	right_neck
18	18	1.657727e+09	1.657727e+09	NaN	left_neck
19	19	1.657727e+09	1.657727e+09	NaN	forehead
20	20	1.657727e+09	1.657727e+09	NaN	right_cheek
21	21	1.657727e+09	1.657727e+09	NaN	left_cheek
22	22	1.657727e+09	1.657727e+09	NaN	Left_hand_Slow_Small
23	23	1.657727e+09	1.657727e+09	NaN	right_neck
24	24	1.657727e+09	1.657727e+09	NaN	left_neck
25	25	1.657727e+09	1.657727e+09	NaN	forehead
26	26	1.657727e+09	1.657727e+09	NaN	right_cheek
27	27	1.657727e+09	1.657727e+09	NaN	left_cheek
28	28	1.657727e+09	1.657727e+09	NaN	Random
29	29	1.657727e+09	1.657727e+09	NaN	Random
30	30	1.657727e+09	1.657727e+09	NaN	Random
31	31	1.657727e+09	1.657727e+09	NaN	Random
32	32	1.657727e+09	1.657727e+09	skip	Random
33	33	1.657727e+09	1.657727e+09	NaN	Random

Principal Components & Signal Vector Magnitude

In [9]:
```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components = 2)
pca.fit(df[["x-axis" , "y-axis", "z-axis"]])
pca_arr = pca.transform(df[["x-axis" , "y-axis", "z-axis"]])


df["PC1"] = np.array(pd.DataFrame(pca_arr)[0])
df["PC2"] = np.array(pd.DataFrame(pca_arr)[1])

df["SVM"] = np.sqrt(df["x-axis"].apply(lambda x : x**2)
                                + df["y-axis"].apply(lambda x : x**2)
                                + df["z-axis"].apply(lambda x : x**2))
```

Annotating Data and labelling for 1(positive) and 0 (negative)

In [10]:
```
def create_labels(training_df,df_labels ):


        #training_df : dataframe containing the x,y z from accelerometer.

        #df_labels : dataframe generated from timestamper.py script


            label_arr = (np.zeros(len(training_df), dtype = np.int8))
            is_first = True
            for i in range(len(df_labels)):
                for j in range(len(training_df)):
                    if (training_df["timestamp"].loc[j] >= (df_labels["start_time"].loc[i] )
                        and training_df["timestamp"].loc[j] <= (df_labels["end_time"].loc[i])):

                        label_arr[j] = int(1)

                        idx = j

                        if is_first:
                            start_idx = j
                            is_first = False

            df["label"] = label_arr
            df0 = df.loc[start_idx - 200:idx+150].reset_index()



            return df0
```

In [11]:
```
df
```

Out[11]:

	timestamp	elapsed (s)	x-axis	y-axis	z-axis	pdtime	time	PC1	PC2	SVM
0	1.657727e+09	0.000	0.065	-0.431	0.918	2022-07-13 15:43:58.814	2022-07-13 15:43:58.813999872	0.557282	-0.522849	1.016223
1	1.657727e+09	0.021	0.059	-0.435	0.907	2022-07-13 15:43:58.835	2022-07-13 15:43:58.835000064	0.558063	-0.514915	1.007648
2	1.657727e+09	0.041	0.055	-0.400	0.891	2022-07-13 15:43:58.855	2022-07-13 15:43:58.855000064	0.520247	-0.514393	0.978216
3	1.657727e+09	0.062	0.064	-0.397	0.886	2022-07-13 15:43:58.876	2022-07-13 15:43:58.876000000	0.515090	-0.507234	0.972986
4	1.657727e+09	0.082	0.062	-0.381	0.901	2022-07-13 15:43:58.896	2022-07-13 15:43:58.896000000	0.504894	-0.525854	0.980207
...	...	...	...	...	...	...	...	...	...	...
18059	1.657727e+09	370.398	0.697	0.131	-0.641	2022-07-13 15:50:09.212	2022-07-13 15:50:09.212000000	-0.519934	0.884163	0.955956
18060	1.657727e+09	370.418	0.397	0.160	-1.048	2022-07-13 15:50:09.232	2022-07-13 15:50:09.232000000	-0.651865	1.087136	1.132039
18061	1.657727e+09	370.439	0.338	0.240	-1.071	2022-07-13 15:50:09.253	2022-07-13 15:50:09.252999936	-0.729916	1.055242	1.148427
18062	1.657727e+09	370.459	0.400	0.193	-0.969	2022-07-13 15:50:09.273	2022-07-13 15:50:09.272999936	-0.658233	1.011369	1.065932
18063	1.657727e+09	370.480	0.455	0.210	-0.899	2022-07-13 15:50:09.294	2022-07-13 15:50:09.293999872	-0.656595	0.970897	1.029236

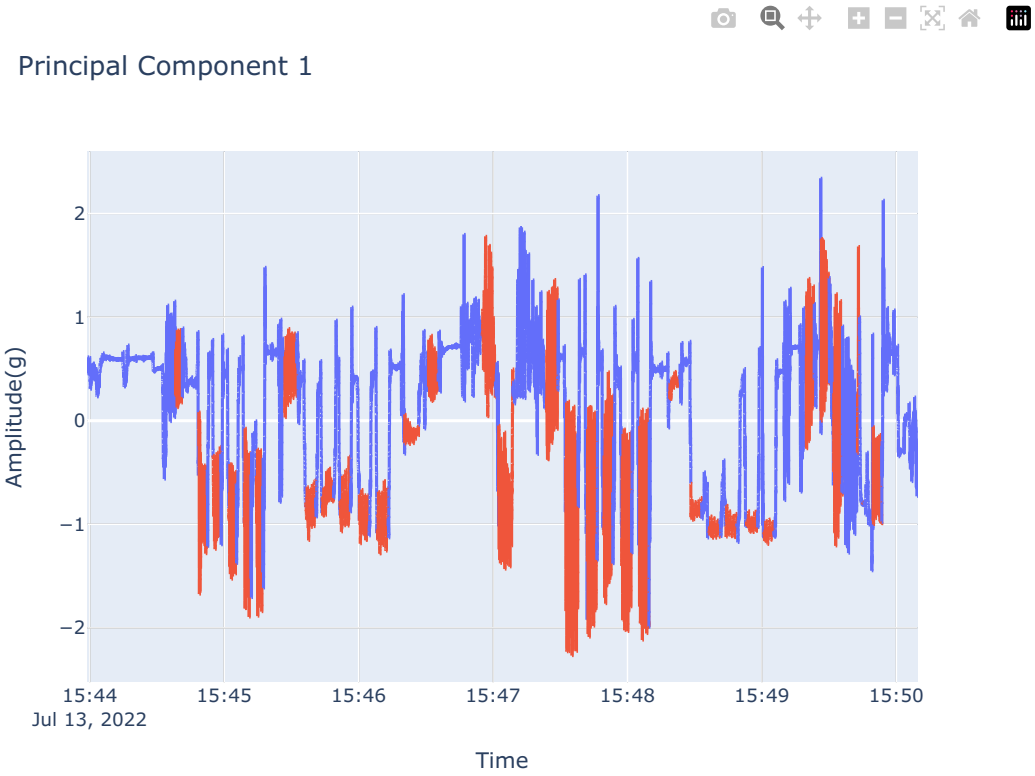
18064 rows × 10 columns

In [12]:
```
df_labs = create_labels(df,df_labels)
```

In [13]:
```
def scratching_plots_up(df_og,df, labs, name):
    color_list = [int(i) for i in labs]
```

```
    x_list = df_og["time"]
    y_list = df
    fig = go.Figure(
    [
        go.Scatter(
            x=x_list[tn : tn + 2],
            y=y_list[tn : tn + 2],
            line_shape="hv",
            line_color=px.colors.qualitative.Plotly[color_list[tn]],
            showlegend=False,
            mode = "lines"
        )
        for tn in range(len(x_list))
    ]
    )
    fig.update_layout(title_text=name, yaxis_title = "Amplitude(g)", xaxis_title = "Time")


    fig.show()
```

In [14]:
```
#px.line(df["x-axis"])

#scratching_plots_up(df,df["x-axis"], df["label"], "X-Axis")
#scratching_plots(df["y-axis"], df["label"], "Y-Axis")
#scratching_plots(df["z-axis"], df["label"], "Z-Axis")
scratching_plots_up(df,df["PC1"], df["label"], "Principal Component 1")
scratching_plots_up(df,df["PC2"], df["label"], "Principal Component 2")
scratching_plots_up(df,df["SVM"], df["label"], "Signal Vector Magnitude")
```



In [15]:

```
window_size = 75
step_size = 38
pc1_list = []
pc2_list = []
svm_list = []
x_list = []
y_list = []
z_list = []
labels = []
times = []

for i in range(0, len(df_labs) - window_size, step_size):
            pc1tmp = df_labs["PC1"].values[i : i + window_size]
            pc2tmp = df_labs["PC2"].values[i : i + window_size]
            svmtmp = df_labs["SVM"].values[i : i + window_size]
            xtmp = df_labs["x-axis"].values[i : i + window_size]
            ytmp = df_labs["y-axis"].values[i : i + window_size]
            ztmp = df_labs["z-axis"].values[i : i + window_size]
            time = df_labs["time"].values[i : i + window_size]
            label = stats.mode(df_labs['label'][i: i + window_size])[0][0]
```

```
            pc1_list.append(pc1tmp)
            pc2_list.append(pc2tmp)
            svm_list.append(svmtmp)
            x_list.append(xtmp)
            y_list.append(ytmp)
            z_list.append(ztmp)
            labels.append(label)
            times.append(time)
```

```
/var/folders/xb/y4zxmn391z50_m52dczhjl800000gn/T/ipykernel_86560/837432911.py:20: FutureWarning:

Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be ac
cepted. Set `keepdims` to True or False to avoid this warning.
```

In [16]:
```
df_train = pd.DataFrame()

# Measure of assymetry in signal
df_train["pc1_skewness"] = pd.Series(pc1_list).apply(lambda x : stats.skew(x))
df_train["pc2_skewness"] = pd.Series(pc2_list).apply(lambda x : stats.skew(x))
#df_train["pc3_skewness"] = pd.Series(pc3_list).apply(lambda x : stats.skew(x))
df_train["svm_skewness"] = pd.Series(svm_list).apply(lambda x : stats.skew(x))

# Diference between extreme values

df_train["svm_signal_range"] = pd.Series(svm_list).apply(lambda x: x.max() - x.min())



# Inter-quartile Range of Auto-Covariance
from  statsmodels.tsa.stattools import acovf
pc1_acv = pd.Series(pc1_list).apply(acovf)
pc2_acv = pd.Series(pc2_list).apply(acovf)
#pc3_acv = pd.Series(pc3_list).apply(acovf)

svm_acv = pd.Series(svm_list).apply(acovf)
x_acv = pd.Series(x_list).apply(acovf)
y_acv = pd.Series(y_list).apply(acovf)
z_acv = pd.Series(z_list).apply(acovf)


df_train['pc1_acv_IQR'] = pc1_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
df_train['pc2_acv_IQR'] = pc2_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
#df_train['pc3_acv_IQR'] = pc3_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))

df_train['svm_acv_IQR'] = svm_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
#df_train['x-axis_acv_IQR'] = x_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
#df_train['y-axis_acv_IQR'] = y_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
#df_train['z-axis_acv_IQR'] = z_acv.apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
```

In [17]:
```
def scratching_plots(df, labs, name):
    color_list = [int(i) for i in labs]
    x_list = [i for i in range(len(df))]
    y_list = df
    fig = go.Figure(
    [
        go.Scatter(
            x=x_list[tn : tn + 2],
            y=y_list[tn : tn + 2],
            line_shape="hv",
            line_color=px.colors.qualitative.Plotly[color_list[tn]],
            showlegend=False,
            mode = "lines"
        )
        for tn in range(len(x_list))
    ]
    )
    fig.update_layout(title_text=name, yaxis_title = "Value", xaxis_title = "Time")
    fig.show()
```

In [18]:
```
df_train
```

Out[18]:

	pc1_skewness	pc2_skewness	svm_skewness	svm_signal_range	pc1_acv_IQR	pc2_acv_IQR	svm_acv_IQR
0	0.191292	0.933464	0.613037	1.581392	0.014490	0.026348	0.025919
1	0.233533	0.640613	0.682881	1.797950	0.021391	0.061409	0.055351
2	0.205543	0.420867	0.597229	1.797950	0.027899	0.116289	0.083869
3	0.320899	0.492260	0.898477	2.144972	0.019736	0.081057	0.049248
4	0.423115	0.492490	0.952874	2.136564	0.018197	0.082947	0.056797
...	...	...	...	...	...	...	...
407	0.305313	0.356897	-0.032648	0.833690	0.027720	0.026176	0.016830
408	0.267170	0.077243	0.209002	0.854719	0.029813	0.021112	0.020384
409	0.778078	0.381616	1.129303	2.488058	0.238012	0.015770	0.029355
410	-0.046775	0.374545	1.577664	2.488058	0.014428	0.009367	0.009561
411	-0.224817	1.310857	-0.185467	0.856390	0.004562	0.002646	0.003389

412 rows × 7 columns

In [19]:
```
scratching_plots(df_train["svm_signal_range"], labels, "Signal Range")
```

In [20]:
```
def zero_cross(lst):
    count = 0
    psv = True
    for i in lst:
        if i > 0:
            if not psv:
                count += 1
                psv = True
            else:
                continue
        elif i < 0:
            if psv:
                count += 1
                psv = False
            else:
                continue
    return count/len(lst)

def mean_cross(lst):
    count = 0
    psv = True
    for i in lst:
        if i > np.mean(lst):
            if not psv:
                count += 1
                psv = True
            else:
```

```
                continue
        elif i < np.mean(lst):
            if psv:
                count += 1
                psv = False
            else:
                continue
    return count/len(lst)



# HIGH mean cross rate values indicate high frequency movements in short duration, higher probability of scratc
df_train["pc1_mean_cross_rate"] = pd.Series(pc1_list).apply(mean_cross)
df_train["pc2_mean_cross_rate"] = pd.Series(pc2_list).apply(mean_cross)
#df_train["pc3_mean_cross_rate"] = pd.Series(pc3_list).apply(mean_cross)
df_train["svm_mean_cross_rate"] = pd.Series(svm_list).apply(mean_cross)
df_train["pc1_zero_cross_rate"] = pd.Series(pc1_list).apply(zero_cross)
df_train["pc2_zero_cross_rate"] = pd.Series(pc2_list).apply(zero_cross)
#df_train["pc3_zero_cross_rate"] = pd.Series(pc3_list).apply(zero_cross)

#df_train["x-axis_mean_cross_rate"] = pd.Series(x_list).apply(lambda x : stats.skew(x))
#df_train["y-axis_mean_cross_rate"] = pd.Series(y_list).apply(lambda x : stats.skew(x))
#df_train["z-axis_mean_cross_rate"] = pd.Series(z_list).apply(lambda x : stats.skew(x))

def dom_freq_val(lst):
    w = np.fft.fft(lst)
    w = w/len(w)
    freqs = np.fft.fftfreq(len(lst))
    i = np.argmax(abs(w))
    dom_freq = freqs[i]
    dom_freq_hz = abs(dom_freq * 25)
    return dom_freq_hz

#df_train['pc1_dom_freq_val'] = pd.Series(pc1_list).apply(dom_freq_val)
#df_train['pc2_dom_freq_val'] = pd.Series(pc2_list).apply(dom_freq_val)


def dom_freq_mag(lst):
    w = np.fft.fft(lst)
    val = dom_freq_val(lst)
    return (val/np.sum(abs(np.delete(w, np.argmax(abs(w))))))*100



df_train['pc1_dom_freq_val'] = pd.Series(pc1_list).apply(dom_freq_val)
df_train['pc2_dom_freq_val'] = pd.Series(pc2_list).apply(dom_freq_val)
# df_train['pc3_dom_freq_val'] = pd.Series(pc3_list).apply(dom_freq_val)

df_train["pc1_dom_freq_mag"] = pd.Series(pc1_list).apply(dom_freq_mag)
df_train["pc2_dom_freq_mag"] = pd.Series(pc2_list).apply(dom_freq_mag)
#df_train["pc3_dom_freq_mag"] = pd.Series(pc3_list).apply(dom_freq_mag)

def spectral_arclength(movement, fs = 0.05, padlevel=4, fc=10.0, amp_th=0.05):

    # Number of zeros to be padded.
    nfft = int(pow(2, np.ceil(np.log2(len(movement))) + padlevel))

    # Frequency
    f = np.arange(0, fs, fs/nfft)
    # Normalized magnitude spectrum
    Mf = abs(np.fft.fft(movement, nfft))
    Mf = Mf/max(Mf)

    # Indices to choose only the spectrum within the given cut off frequency Fc.
    # NOTE: This is a low pass filtering operation to get rid of high frequency
    # noise from affecting the next step (amplitude threshold based cut off for
    # arc length calculation).
    fc_inx = ((f <= fc)*1).nonzero()
    f_sel = f[fc_inx]
    Mf_sel = Mf[fc_inx]

    # Choose the amplitude threshold based cut off frequency.
    # Index of the last point on the magnitude spectrum that is greater than
    # or equal to the amplitude threshold.
    inx = ((Mf_sel >= amp_th)*1).nonzero()[0]
    fc_inx = range(inx[0], inx[-1]+1)
    f_sel = f_sel[fc_inx]
    Mf_sel = Mf_sel[fc_inx]

    # Calculate arc length
    new_sal = -sum(np.sqrt(pow(np.diff(f_sel)/(f_sel[-1] - f_sel[0]), 2) +
                           pow(np.diff(Mf_sel), 2)))
    return new_sal#, (f, Mf), (f_sel, Mf_sel)
```

```
df_train["pc1_spectral_arclength"] = (pd.Series(pc1_list).apply(spectral_arclength))
df_train["pc2_spectral_arclength"] = (pd.Series(pc2_list).apply(spectral_arclength))
#df_train["pc3_spectral_arclength"] = (pd.Series(pc3_list).apply(spectral_arclength))

df_train["svm_spectral_arclength"] = (pd.Series(svm_list).apply(spectral_arclength))


def dimensionless_jerk(movement, fs = 0.05):

    # first enforce data into an numpy array.
    movement = np.array(movement)

    # calculate the scale factor and jerk.
    movement_peak = max(abs(movement))
    dt = 1./fs
    movement_dur = len(movement)*dt
    jerk = np.diff(movement, 2)/pow(dt, 2)
    scale = pow(movement_dur, 3)/pow(movement_peak, 2)

    # estimate dj
    return - scale * sum(pow(jerk, 2)) * dt

df_train["pc1_jerk"] = pd.Series(pc1_list).apply(dimensionless_jerk)
df_train["pc2_jerk"] = pd.Series(pc2_list).apply(dimensionless_jerk)
# df_train["pc3_jerk"] = pd.Series(pc3_list).apply(dimensionless_jerk)
df_train["svm_jerk"] = pd.Series(svm_list).apply(dimensionless_jerk)

def log_dimensionless_jerk(movement, fs = 0.05):
    return -np.log(abs(dimensionless_jerk(movement, fs)))

df_train["pc1_log_jerk"] = pd.Series(pc1_list).apply(log_dimensionless_jerk)
df_train["pc2_log_jerk"] = pd.Series(pc2_list).apply(log_dimensionless_jerk)
#df_train["pc3_log_jerk"] = pd.Series(pc3_list).apply(log_dimensionless_jerk)

df_train["svm_log_jerk"] = pd.Series(svm_list).apply(log_dimensionless_jerk)
```

In [21]:
```
scratching_plots(df_train["svm_jerk"], labels, "SVM Dimensionless Jerk")
```

In [22]:
```
df_train
```

Out[22]:

	pc1_skewness	pc2_skewness	svm_skewness	svm_signal_range	pc1_acv_IQR	pc2_acv_IQR	svm_acv_IQR	pc1_mean_cross_rate	pc2_
0	0.191292	0.933464	0.613037	1.581392	0.014490	0.026348	0.025919	0.146667	
1	0.233533	0.640613	0.682881	1.797950	0.021391	0.061409	0.055351	0.173333	
2	0.205543	0.420867	0.597229	1.797950	0.027899	0.116289	0.083869	0.160000	
3	0.320899	0.492260	0.898477	2.144972	0.019736	0.081057	0.049248	0.173333	
4	0.423115	0.492490	0.952874	2.136564	0.018197	0.082947	0.056797	0.173333	
...	...	...	...	...	...	...	...	...	
407	0.305313	0.356897	-0.032648	0.833690	0.027720	0.026176	0.016830	0.160000	
408	0.267170	0.077243	0.209002	0.854719	0.029813	0.021112	0.020384	0.160000	
409	0.778078	0.381616	1.129303	2.488058	0.238012	0.015770	0.029355	0.026667	
410	-0.046775	0.374545	1.577664	2.488058	0.014428	0.009367	0.009561	0.186667	
411	-0.224817	1.310857	-0.185467	0.856390	0.004562	0.002646	0.003389	0.053333	

412 rows × 25 columns

In [23]:
```
lst = []
for i in df_train:
    lst.append(i)
```

In [24]:
```
lst
pd.DataFrame(lst).to_csv("features.csv")
```

In [25]:
```
for i in lst:
    scratching_plots(df_train[i], labels, i)
```

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import pandas as pd
import plotly.io as pio
pio.renderers.default='notebook'
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import sys
import os
from functools import reduce
from sklearn.model_selection import cross_validate
import time
from pactools.grid_search import GridSearchCVProgressBar
from sklearn.experimental import enable_halving_search_cv  # noqa
from sklearn.model_selection import HalvingRandomSearchCV
```

In [2]:
```python
lst = []
indices = dict()
tmp = 0
count = 0
for i in range(0,9):
    end = (len(pd.read_csv("Processed_Data/data{}.csv".format(i))))
    indices[i] = (tmp, tmp + end )
    tmp += end
    lst.append(pd.read_csv("Processed_Data/data{}.csv".format(i)))
df_train = reduce(lambda a,b : pd.concat([a,b], ignore_index = True, sort = False),lst)
```

In [3]:
```python
labels = np.array(df_train["label"])
df_train = df_train.drop(columns = ["Unnamed: 0", "label"])
```

In [4]:
```python
df_train
```

Out[4]:

	pc1_skewness	pc2_skewness	svm_skewness	svm_signal_range	pc1_acv_IQR	pc2_acv_IQR	svm_acv_IQR	pc1_mean_cross_rate	pc2_mean_cross_rate	svm_mean_cross_rate	...	pc2_dom_freq_mag	pc1_s
0	0.217169	0.427684	-0.267858	2.288246	0.047415	1.006595	0.086815	0.293333	0.186667	0.373333	...	0.404574	
1	0.334265	0.323507	-0.401467	2.291532	0.132898	1.313510	0.157584	0.213333	0.186667	0.360000	...	0.364872	
2	0.183891	0.444156	-0.412860	2.272023	0.150718	1.140971	0.093732	0.280000	0.173333	0.373333	...	0.379567	
3	-0.127863	0.634129	-0.715171	2.432092	0.084887	1.228101	0.061760	0.280000	0.173333	0.346667	...	0.359823	
4	-0.195138	0.516711	-0.693328	2.410430	0.105364	1.597534	0.130668	0.213333	0.186667	0.346667	...	0.398151	
...	...	...	...	...	...	...	...	...	...	...	...	...	
3824	-2.752360	2.716700	2.636396	0.675728	0.000371	0.003032	0.000547	0.040000	0.026667	0.066667	...	0.000000	
3825	0.198934	-0.375018	2.003070	2.501907	0.089026	0.062560	0.010948	0.093333	0.066667	0.160000	...	0.178014	
3826	0.049758	-0.263374	2.342345	2.501907	0.059870	0.025049	0.007552	0.080000	0.040000	0.173333	...	0.000000	
3827	-4.099096	0.746744	0.169340	0.501452	0.000102	0.000209	0.000083	0.186667	0.093333	0.200000	...	0.000000	
3828	0.041662	-1.069579	-0.577920	0.038737	0.000008	0.000006	0.000003	0.160000	0.066667	0.186667	...	0.000000	

3829 rows × 25 columns

In [5]:
```python
np.unique(labels, return_counts = True)
```

Out[5]:
```
(array([0, 1]), array([2363, 1466]))
```

In [6]:
```python
X_train, X_test, y_train, y_test = train_test_split(df_train, np.array(labels), test_size=0.2, random_state=5)
```

In [7]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression

tree= DecisionTreeClassifier(random_state=24)
forest= RandomForestClassifier(random_state=24)
knn= KNeighborsClassifier()
svm= SVC(random_state=24, probability = True)
log = LogisticRegression()
adb = AdaBoostClassifier()

# now, create a list with the objects
models= [tree, forest, knn, svm, adb, log]
models_perf = []

for model in models:
    model.fit(X_train, y_train) # fit the model
    y_pred= model.predict(X_test) # then predict on the test set
    score= metrics.roc_auc_score(y_test, model.predict_proba(X_test)[:,1]) # this gives us how often the algorithm predicted correctly
    clf_report= metrics.classification_report(y_test, y_pred) # with the report, we have a bigger picture, with precision and recall for each class
    print(f"The ROC Score of model {type(model).__name__} is {score:.2f}")
    print(clf_report)
    print("\n")
    models_perf.append([model, score])
```

```
The ROC Score of model DecisionTreeClassifier is 0.80
              precision    recall  f1-score   support

           0       0.86      0.83      0.84       476
           1       0.74      0.77      0.75       290

    accuracy                           0.81       766
   macro avg       0.80      0.80      0.80       766
weighted avg       0.81      0.81      0.81       766



The ROC Score of model RandomForestClassifier is 0.95
              precision    recall  f1-score   support

           0       0.89      0.91      0.90       476
           1       0.85      0.82      0.84       290

    accuracy                           0.88       766
   macro avg       0.87      0.87      0.87       766
weighted avg       0.88      0.88      0.88       766



The ROC Score of model KNeighborsClassifier is 0.76
              precision    recall  f1-score   support

           0       0.74      0.79      0.77       476
           1       0.62      0.55      0.58       290

    accuracy                           0.70       766
   macro avg       0.68      0.67      0.68       766
weighted avg       0.70      0.70      0.70       766



The ROC Score of model SVC is 0.76
              precision    recall  f1-score   support

           0       0.73      0.84      0.78       476
           1       0.65      0.48      0.55       290

    accuracy                           0.70       766
   macro avg       0.69      0.66      0.67       766
weighted avg       0.70      0.70      0.69       766



The ROC Score of model AdaBoostClassifier is 0.91
              precision    recall  f1-score   support

           0       0.87      0.85      0.86       476
           1       0.76      0.80      0.78       290

    accuracy                           0.83       766
   macro avg       0.82      0.82      0.82       766
weighted avg       0.83      0.83      0.83       766



The ROC Score of model LogisticRegression is 0.73
              precision    recall  f1-score   support

           0       0.80      0.24      0.37       476
           1       0.42      0.90      0.57       290

    accuracy                           0.49       766
   macro avg       0.61      0.57      0.47       766
weighted avg       0.66      0.49      0.44       766
```

Grid Search For hyperparameters

Randomized Grid search for best hyperparameters.

In [8]:
```python
param_grid = {'bootstrap': [True, False],
              'max_depth': [10, 20, 40, 80, None],
              'max_features': [None, 'sqrt'],
              'criterion' : ["gini", "entropy"],
              'min_samples_leaf': [1, 2, 4],
              'min_samples_split': [2, 5, 10],
              'n_estimators': [200,400,800,2000]}
```

In [9]:
```python
rf = RandomForestClassifier()
```

rf_search = HalvingRandomSearchCV(rf, param_grid, resource='n_samples', max_resources='auto', factor = 10, verbose = 20, random_state = 28).fit(X_train, y_train)

rf_randomsearch = RandomizedSearchCV(estimator = rf, param_distributions = param_grid, n_iter = 100, cv = 3, verbose=20, random_state= None, scoring = "roc_auc")

rf_randomsearch.fit(df_train, labels)

In [11]:
```python
clf = RandomForestClassifier(bootstrap = True, criterion='gini', max_depth=80, max_features="sqrt",
                             min_samples_leaf=2, min_samples_split = 2, n_estimators=2000)
clf.fit(X_train, y_train)
```

Out[11]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=80, min_samples_leaf=2, n_estimators=2000)
```

5 fold Cross-Validation

In [12]:
```python
lst00 = []
X = np.array(df_train)
y = np.array(labels)
cv_auroc = []
cv_accuracy = []
cv_f1 = []
cv_sens = []
cv_specificity = []
cv_ppv = []
cv_npv = []
from sklearn.model_selection import KFold
kf = KFold(n_splits = 5)
for train_idx, test_idx in kf.split(X):
    print("a")
    train_X, train_y, test_X, test_y = X[train_idx], y[train_idx], X[test_idx], y[test_idx]
    clf.fit(train_X, train_y)
    print("b")
    tn, fp, fn, tp = metrics.confusion_matrix(test_y, clf.predict(test_X)).ravel()
    cv_auroc.append(metrics.roc_auc_score(test_y, clf.predict_proba(test_X)[:,1]))
    cv_accuracy.append( metrics.accuracy_score(test_y, clf.predict(test_X)))
    cv_f1.append(metrics.f1_score(test_y, clf.predict(test_X)))
    cv_sens.append(tp/(tp+fn))
    cv_specificity.append(tn/(tn + fp))
    cv_ppv.append(tp/(tp+fp))
    cv_npv.append(tn/(tn+fn))
    y_pred_proba = clf.predict_proba(test_X)[:,1]
    fpr, tpr, thresholds = metrics.roc_curve(test_y, y_pred_proba)
    lst00.append((fpr,tpr))
```

```
a
b
a
b
a
b
a
b
a
b
```

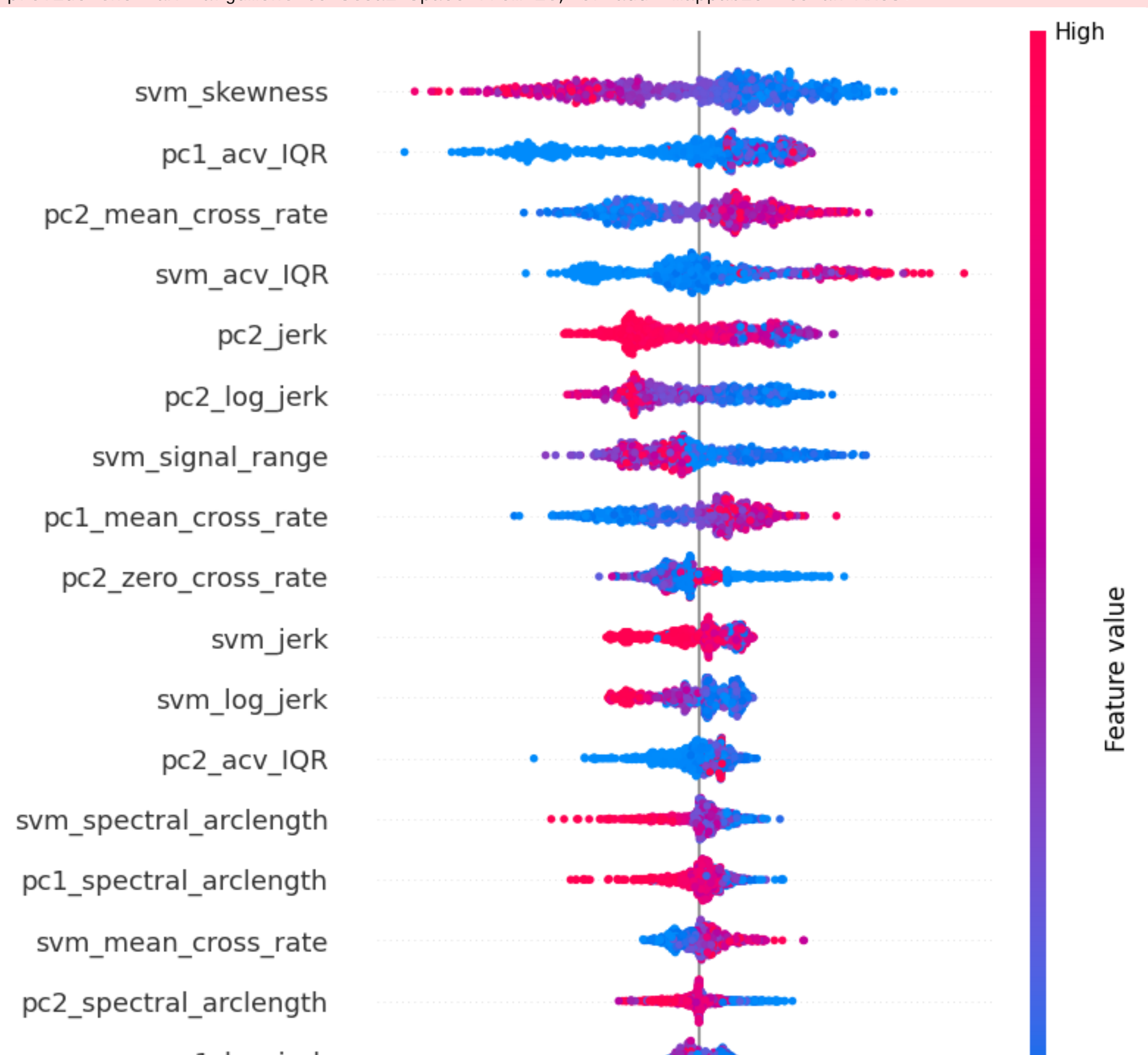
In [13]:
```python
import shap
explainer = shap.TreeExplainer(clf)
start = time.time()
shap_values = explainer(X_test)
```

```
/usr/local/lib/python3.10/site-packages/tqdm/auto.py:22: TqdmWarning:

IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html

Importing display from IPython.core.display is deprecated since IPython 7.14, please import from IPython display
Importing display from IPython.core.display is deprecated since IPython 7.14, please import from IPython display
Importing display from IPython.core.display is deprecated since IPython 7.14, please import from IPython display
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you speci
fically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you speci
fically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
```

SHAPley Analysis for top 20 feature importances

In [14]:
```python
shap_values = np.array(explainer.shap_values(X_test))
shap.summary_plot(shap_values[1],X_test)
print("Time Taken in s:", time.time() - start)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored
Unable to determine Axes to steal space for Colorbar. Using gca(), but will raise in the future. Either provide the *cax* argument to use as the Axes for the Colorbar,
provide the *ax* argument to steal space from it, or add *mappable* to an Axes.
```




```
Time Taken in s: 322.1995027065277
```

In [15]:
```python
clf.fit(X_train, y_train)
```

Out[15]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=80, min_samples_leaf=2, n_estimators=2000)
```

Testing set Performance

All metrics given at the end

In [16]:
```python
y_test
```

Out[16]:
```
array([0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0])
```

In [17]:
```python
y_pred_proba = clf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_proba)
lst00.append((fpr,tpr))
```

In [18]:
```python
preds = clf.predict(X_test)
print("Confusion Matrix:")
tn, fp, fn, tp = metrics.confusion_matrix(y_test, preds).ravel()
print(f'True Positives: {tp}')
print(f'False Positives: {fp}')
print(f'True Negatives: {tn}')
print(f'False Negatives: {fn}')

print("\nClassification Report:")

print(metrics.classification_report(y_test, preds))
```

```
Confusion Matrix:
True Positives: 242
False Positives: 44
True Negatives: 432
False Negatives: 48

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.91      0.90       476
           1       0.85      0.83      0.84       290

    accuracy                           0.88       766
   macro avg       0.87      0.87      0.87       766
weighted avg       0.88      0.88      0.88       766
```

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import pandas as pd
import plotly.io as pio
pio.renderers.default='notebook'
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import sys
import os
from functools import reduce
from sklearn.model_selection import cross_validate
import time
from pactools.grid_search import GridSearchCVProgressBar
from sklearn.experimental import enable_halving_search_cv  # noqa
from sklearn.model_selection import HalvingRandomSearchCV
```

In [2]:
```python
lst = []
subj_indices = dict()
tmp = 0
count = 0
for i in range(0,9):
    subj_end = (len(pd.read_csv("Processed_Data/data{}.csv".format(i))))
    subj_indices[i] = (tmp, tmp + subj_end )
    tmp += subj_end
    lst.append(pd.read_csv("Processed_Data/data{}.csv".format(i)))
df_train = reduce(lambda a,b : pd.concat([a,b], ignore_index = True, sort = False),lst)
```

In [3]:
```python
labels = np.array(df_train["label"])
df_train = df_train.drop(columns = ["Unnamed: 0"])
```

In [4]:
```python
df_train
```

Out[4]:

	pc1_skewness	pc2_skewness	svm_skewness	svm_signal_range	pc1_acv_IQR	pc2_acv_IQR	svm_acv_IQR	pc1_mean_cross_rate	pc2_mean_cross_rate	svm_mean_cross_rate	...	pc1_spectral_arclength	pc
0	0.217169	0.427684	-0.267858	2.288246	0.047415	1.006595	0.086815	0.293333	0.186667	0.373333	...	-7.740445	
1	0.334265	0.323507	-0.401467	2.291532	0.132898	1.313510	0.157584	0.213333	0.186667	0.360000	...	-9.202736	
2	0.183891	0.444156	-0.412860	2.272023	0.150718	1.140971	0.093732	0.280000	0.173333	0.373333	...	-9.435125	
3	-0.127863	0.634129	-0.715171	2.432092	0.084887	1.228101	0.061760	0.280000	0.173333	0.346667	...	-9.227076	
4	-0.195138	0.516711	-0.693328	2.410430	0.105364	1.597534	0.130668	0.213333	0.186667	0.346667	...	-8.267170	
...	...	...	...	...	...	...	...	...	...	...	...	...	
3824	-2.752360	2.716700	2.636396	0.675728	0.000371	0.003032	0.000547	0.040000	0.026667	0.066667	...	-5.713489	
3825	0.198934	-0.375018	2.003070	2.501907	0.089026	0.062560	0.010948	0.093333	0.066667	0.160000	...	-9.793343	
3826	0.049758	-0.263374	2.342345	2.501907	0.059870	0.025049	0.007552	0.080000	0.040000	0.173333	...	-8.536678	
3827	-4.099096	0.746744	0.169340	0.501452	0.000102	0.000209	0.000083	0.186667	0.093333	0.200000	...	-6.210468	
3828	0.041662	-1.069579	-0.577920	0.038737	0.000008	0.000006	0.000003	0.160000	0.066667	0.186667	...	-6.863067	

3829 rows × 26 columns

In [5]:
```python
subj_indices
```

Out[5]:
```
{0: (0, 450),
 1: (450, 775),
 2: (775, 1172),
 3: (1172, 1638),
 4: (1638, 2223),
 5: (2223, 2708),
 6: (2708, 3127),
 7: (3127, 3521),
 8: (3521, 3829)}
```

In [6]:
```python
np.unique(labels, return_counts = True)
```

Out[6]:
```
(array([0, 1]), array([2363, 1466]))
```

In [7]:
```python
testing_set = df_train.iloc[2708:3127].reset_index().drop(columns = "index")
training_set = df_train.drop([i for i in range(2708,3127)]).reset_index().drop(columns = "index")

training_set.to_csv("training_set.csv")
testing_set.to_csv("testing_set.csv")
```

In [31]:
```python
training_set
```

Out[31]:

	pc1_skewness	pc2_skewness	svm_skewness	svm_signal_range	pc1_acv_IQR	pc2_acv_IQR	svm_acv_IQR	pc1_mean_cross_rate	pc2_mean_cross_rate	svm_mean_cross_rate	...	pc1_spectral_arclength	pc
0	0.217169	0.427684	-0.267858	2.288246	0.047415	1.006595	0.086815	0.293333	0.186667	0.373333	...	-7.740445	
1	0.334265	0.323507	-0.401467	2.291532	0.132898	1.313510	0.157584	0.213333	0.186667	0.360000	...	-9.202736	
2	0.183891	0.444156	-0.412860	2.272023	0.150718	1.140971	0.093732	0.280000	0.173333	0.373333	...	-9.435125	
3	-0.127863	0.634129	-0.715171	2.432092	0.084887	1.228101	0.061760	0.280000	0.173333	0.346667	...	-9.227076	
4	-0.195138	0.516711	-0.693328	2.410430	0.105364	1.597534	0.130668	0.213333	0.186667	0.346667	...	-8.267170	
...	...	...	...	...	...	...	...	...	...	...	...	...	
3405	-2.752360	2.716700	2.636396	0.675728	0.000371	0.003032	0.000547	0.040000	0.026667	0.066667	...	-5.713489	
3406	0.198934	-0.375018	2.003070	2.501907	0.089026	0.062560	0.010948	0.093333	0.066667	0.160000	...	-9.793343	
3407	0.049758	-0.263374	2.342345	2.501907	0.059870	0.025049	0.007552	0.080000	0.040000	0.173333	...	-8.536678	
3408	-4.099096	0.746744	0.169340	0.501452	0.000102	0.000209	0.000083	0.186667	0.093333	0.200000	...	-6.210468	
3409	0.041662	-1.069579	-0.577920	0.038737	0.000008	0.000006	0.000003	0.160000	0.066667	0.186667	...	-6.863067	

3410 rows × 26 columns

In [32]:
```python
y_train = np.array(training_set["label"])
X_train = training_set.drop(columns = "label")

y_test = np.array(testing_set["label"])
X_test = testing_set.drop(columns = "label")
```

Grid Search For hyperparameters

Randomized Grid search for best hyperparameters.

In [19]:
```python
param_grid = {'bootstrap': [True, False],
              'max_depth': [10, 20, 40, 80, None],
              'max_features': [None, 'sqrt'],
              'criterion' : ["gini", "entropy"],
              'min_samples_leaf': [1, 2, 4],
              'min_samples_split': [2, 5, 10],
              'n_estimators': [200,400,800,2000]}
```

In [ ]:
```python
rf = RandomForestClassifier()
```

In [ ]:
```python
rf_search = HalvingRandomSearchCV(rf,
                                  param_grid,
                                  resource='n_samples',
                                  max_resources='auto',
                                  factor = 10,
                                  verbose = 20,
                                  random_state = 28).fit(X_train, y_train)

#rf_randomsearch = RandomizedSearchCV(estimator = rf, param_distributions = param_grid, n_iter = 100, cv = 3, verbose=20, random_state= None, scoring = "roc_auc")
#rf_randomsearch.fit(df_train, labels)
```

In [33]:
```python
clf = RandomForestClassifier(bootstrap = True, criterion='gini', max_depth=80, max_features="sqrt",
                             min_samples_leaf=2, min_samples_split = 2, n_estimators=2000)
clf.fit(X_train, y_train)
```

Out[33]:

▾ RandomForestClassifier

RandomForestClassifier([redacted])

In [34]:
```python
y_test
```

Out[34]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0])
```

In [36]:
```python
preds = clf.predict(X_test)
print("Confusion Matrix:")
tn, fp, fn, tp = metrics.confusion_matrix(y_test, preds).ravel()
print(f'True Positives: {tp}')
print(f'False Positives: {fp}')
print(f'True Negatives: {tn}')
print(f'False Negatives: {fn}')

print("\nClassification Report:")

print(metrics.classification_report(y_test, preds))

y_pred_proba = clf.predict_proba(X_test)[:,1]
print("\n\n\nTesting Set Performance Unseen Set:\n")
print("AUROC: {:.2f}".format( metrics.roc_auc_score(y_test, y_pred_proba)))
print("Accuracy Score: {:.2f}".format( metrics.accuracy_score(y_test, clf.predict(X_test))))
print("F1 Score: {:.2f}".format(metrics.f1_score(y_test, clf.predict(X_test))))
print("Sensitivity: {:.2f}".format(tp/(tp+fn)))
print("Specificity: {:.2f}".format(tn/(tn + fp)) )
print("Positive Predictive Value: {:.2f}".format(tp/(tp+fp)))
print("Negative Predictive Value: {:.2f}".format(tn/(tn+fn)))
```

```
Confusion Matrix:
True Positives: 153
False Positives: 31
True Negatives: 202
False Negatives: 33

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.87      0.86       233
           1       0.83      0.82      0.83       186

    accuracy                           0.85       419
   macro avg       0.85      0.84      0.85       419
weighted avg       0.85      0.85      0.85       419




Testing Set Performance Unseen Set:

AUROC: 0.90
Accuracy Score: 0.85
F1 Score: 0.83
Sensitivity: 0.82
Specificity: 0.87
Positive Predictive Value: 0.83
Negative Predictive Value: 0.86
```

In [ ]: