Carolyn Heinzer
CMSC 23400
Lab 3 Tasks 2&3
06/09/2024

Task 2: User Identification Classifier

Methods:

Since I don't have an Oculus or other VR headset, I used the same data and therefore the same gesture as my group did when we completed Lab 3 – jumping jacks. They work well because everyone has a different jump height and wingspan, as well as speed.

- The three authenticated users are Carolyn ("CAR"), Quinn ("QUI"), and Urunna ("URU").
- I used the same scenes and scripts provided in previous labs for collecting data, since they already allow recording data as different users. I then changed the labels of the files from activity codes to the user codes (listed above).

Design process:

I based my code for splitting the data into testing and training sets on the code I wrote for that purpose in Lab 2. This time, though, I wanted to make sure equal amounts of each user's data were incorporated into the model, so I chose six random files from each user's data for the Validation set and put the rest into Training.

I started out trying to adapt my code from last week to classify users with a decision tree. However, a problem I kept running into last week was that the classification report kept saying the model ran with 100% accuracy, which is very suspicious. I used the predict_proba( ) function to find the "confidence" scores for each sample in the Validation data set, and every sample scored 1.0 in the category the model classified it as. Again, this was suspicious, but I really realized there was a problem when I ran the model on the Adversary data set and predict_proba( ) returned 1.0 on whatever category the model classified it as, which is obviously impossible.

I realized my model was probably overfitting – the tree was probably too deep and started learning the noise in the data as well – so I used GridSearchCV to find the optimal hyperparameters. However, even when I adjusted depth and other parameters, accuracy was still at 100%, as was predict_proba( ) for every sample in both the Adversary and Validation sets. If I pruned the tree more than recommended (I ended up trying depths of 1), the accuracy would decrease, as would predict_proba( ), but the confidence values it returned for the users it classified a sample as were the same for samples in Validation and Adversary, so it was impossible to distinguish between a valid user and and an invalid one.

I re-worked my entire code multiple times (predict_user.py, ground_up.py, 0-combine_data.py, and 1-diff_features.py, which is my final code), starting from the first

helper function and testing everything written every time I added a small chunk. I changed the Validation/Train split from 30/60 to 20/80, I tried combining all data into a single file first and then splitting the data. I thought maybe the model was considering a sample (which I had preprocessed and put relevant attributes into a dictionary) as a string rather than several separate items. It wasn't. I thought that maybe the way I processed data generalized it too much (I was using number of peaks, mean value, and standard deviation) and initially tried to put the entire trace of relevant attributes into the X set for the decision tree before realizing that I couldn't put lists as elements of X.

Thinking about processing the data, though, made me think about each attribute individually and what would distinguish different people versus different activities. Different numbers of peaks in the velocity of a controller, for example, would vary more widely between activities than between users doing the same activity, I think. I chose a few ways of preprocessing a few attributes and looked at how their values varied between users, then incrementally added more attributes.

Unfortunately, I was still having the same problems with predict_proba( ) and accuracy. I read up on classification in the scikit learn documentation and found a method called decision_function, which "predicts a "soft" score for each sample in relation to each class, rather than the "hard" categorical prediction produced by predict." I hoped this meant that it would give non-100% confidence values, if for some reason, predict_proba( ) was the thing having issues. When trying to better understand this function, I came across a Stack Overflow post talking about the distinction between decision_function and predict as methods of C-support vector classification (SVC). Because I had gone through so much revision and rewriting, I thought that perhaps trying a different method that wasn't a decision tree was worth a shot. I barely had to change the code I had written to replace the DecisionTreeClassifier with SVC( ). After making the switch, the classification report reported 89% accuracy and predict_proba( ) arrays that did not simply consist of 0s and 1s.

My model:

In their paper "Behavioural Biometrics in VR: Identifying People from Body Motion and Relations in Virtual Reality,"[1] Pfeuffer et al. claim that "particular feature combinations" can be derived from the movement and position of body segments (here the head and hands), so I focused on velocity and position of the headset and controllers.

Specifically, the attributes I chose to focus on are:

---

[1] Ken Pfeuffer, Matthias J. Geiger, Sarah Prange, Lukas Mecke, Daniel Buschek, and Florian Alt. 2019. Behavioural Biometrics in VR: Identifying People from Body Motion and Relations in Virtual Reality. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) (*CHI '19*). Association for Computing Machinery, New York, NY, USA, 1âĂŞ12. https://doi.org/10.1145/3290605.3300340

- Maximum headset_vel.y: This should reflect maximum jump speed, which is likely to differ between users.
- Maximum headset_pos.y: This should be the user's maximum jump height. The user's standing height seems to be normalized to headset_pos.y = 0, so this is the closest thing I could think of to something that's unique per user relating to height.
- Min and max controller_left_vel.x, y, and z: The signature gesture is jumping jacks, so the users will be moving their arms quite a lot
- Min and max controller_left_pos.x, y, and z: should give a sense of the shape of the arm movements and will differ depending on wingspan
- Max controller_left_pos.z - min controller_left_pos.z: this should give us a sense of (slightly less than) the user's wingspan

These are the feature columns upon which the SVC is trained, with 'User' as the target. I decided to use an SVC rather experimentally, as it provided confidence scores for classifications that actually differed between valid users and adversaries. I believe the SVC to have suffered less from massive overfitting than the decision tree I initially tried, and it may also be successful because SVMs are more adept with handling high-dimensional data, which, since this data has only one observation (user) and several features (described above), it is.

My code:

to_dict(file_name: str, code: int):

- Open the specified file (either from the Train or Validation folder depending on the code) and set the keys of a dictionary to the fields dictated in the first line. Read through each line, separate it by commas, and append each value to the relevant attribute name in the dictionary, then return the dictionary.

preprocess(file_name: str, code: int)

- This function creates a dictionary and sets it to the result of to_dict called upon its inputs. It then returns a new dictionary of relevant metadata on a set of relevant attributes.

combine_files(dir: str, file_out: str, code: int):

- The files in a directory are listed, then preprocess is called on each of them and the resulting dictionary is appended to a list of dictionaries. A csv file of specified name is created and the dictionary is written to the csv with the header as a specified list of the relevant metadata/attributes.

create_classifier(train_file: str):

- Create a dataframe from the specified csv and map a dictionary of user letter codes to numbers onto the user field of the dataframe. Train a SVC with all the columns in the dataframe except 'user' as feature columns and 'user' as the target column, then return the SVC.

predict_shallow(sensor_data: str) -> str:

- This function creates an SVC by calling create_classifier on the file created in combine_files from the Train folder, then calls preprocess on whatever sensor data it's predicting the user for. The code for preprocess should be 1 since this file is either going to be from Validation or Adversary. It then calls predict on all values except the corresponding value for 'user' and returns the user code for the numeric result.

predict_shallow_folder(data_folder: str, output: str):

- The only thing I changed about this function was finding the actual values for the files in whatever folder this is running on so that I could run classification report after the labels are output. It calls predict_shallow on every file in the specified folder.

Results:

Classification report for files in Validation (test set):

	Precision	Recall	F1-score	Support
CAR	0.91	1.00	0.95	10
QUI	0.80	1.00	0.89	8
URU	1.00	0.67	0.80	9
Accuracy			0.89	27
Macro avg.	0.90	0.89	0.88	27
Weighted avg.	0.91	0.89	0.88	27

Latency: ~6s

Trained with [ ] samples/user	Accuracy
5	0.84
10	0.90
20	0.89

I initially trained the classifier on 20 samples per user for an 80/20 split of training. It's interesting that training it with 10 per user provides a small increase in accuracy. I'm not sure if it's negligible or not.

Discussion:

The difficulty I had initially with getting 1.0 confidence scores for adversary data from predict_proba( ) with the decision tree suggests that the data is extremely noisy and perhaps difficult to distinguish between users. Unless data is de-noised, then, it seems easy for an adversary to log in as a user with this gesture and a decision tree considering any and all recorded attributes. With the SVC, confidence scores are lower for adversarial data but still significant, suggesting that noise is still a problem or the distinguishing features I have are not distinct enough.

As described by Liebers et al. in their paper "Understanding User Identification in Virtual Reality Through Behavioral Biometrics and the Effect of Body Normalization,"[2] body normalization increases user identifiability because it forces the model to focus on idiosyncratic movements by the users rather than just their height and arm length. However, I wasn't even able to get height because the y attribute of position is set to 0 when the user isn't moving vertically. If I were to re-record data for this, I would have the user hold their arms out horizontally and add the positions of each controller, then normalize their height and wingspan.

Task 3: Adversary Identification

Methods:

I looked at the results of predict_proba( ) for valid and adversarial data and saw that for most of the sensor traces, the smallest difference between confidence scores for each user was frequently greater than 0.3, or 30%. Initially, I hoped there would be a clear threshold that confidence scores for a valid user met and an adversary did not, but could not find one that was higher than most adversarial confidence scores and lower than most valid users'.

I implemented my check in predict_shallow( ), creating an array of probabilities for each user class, determining their minimum and maximum, and then subtracting the difference between the maximum and whichever score was neither. Then, if that difference was greater than or equal to 0.3, I called predict( ) on the X values. Otherwise, predict_shallow returns the string "ADV."

Results:

Classification report for files in Validation (valid users):

---

[2] Jonathan Liebers, Mark Abdelaziz, Lukas Mecke, Alia Saad, Jonas Auda, Uwe Gruenefeld, Florian Alt, and Stefan Schneegass. 2021. Understanding User Identification in Virtual Reality Through Behavioral Biometrics and the Effect of Body Normalization. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) (*CHI '21*). Association for Computing Machinery, New York, NY, USA, Article 517, 11 pages. https: //doi.org/10.1145/3411764.3445528

	Precision	Recall	F1-score	Support
CAR	1.00	0.80	0.89	10
QUI	0.88	0.88	0.88	8
URU	1.00	0.67	0.80	9
Accuracy			0.78	27
Macro avg.	0.72	0.59	0.64	27
Weighted avg.	0.96	0.78	0.86	27

Classification report for files in Adversary:

	Precision	Recall	F1-score	Support
ADV	1.00	0.28	0.43	18
Accuracy			0.28	18
Macro avg.	0.33	0.09	0.14	18
Weighted avg.	1.00	0.28	0.43	18

Latency:

- Validation set: 5s
- Adversarial set: 2s

Discussion:

The accuracy is about what I expected from looking at the data and deciding upon this metric. It's not a fantastic method of detecting an adversary, but I was unable to find a better one, at least in terms of analyzing predict_proba( ) and using the same attributes as previously. Valid users aren't consistently more confidently predicted by the model than adversaries, which frequently have confidence scores over 0.5 and occasionally over 0.9. This may be yet another effect of noise. Height is relative, so adversaries artificially making themselves taller or shorter is not a vulnerability. One limitation is that the model relies heavily on metadata rather than the sensor trace itself, so individuals have less unique data profiles.

Task 2 Files:

- Assets/Scripts/OculusSensorCapture.cs (didn't modify)
- Assets/Scripts/ActivityDetector.cs (didn't modify)
- Python/Lab3/create_validation_set.py
    - To run: python3 Python/Lab3/create_validation_set.py, although it has already been run and the data is already split, so you'd have to run it on an older version of the Data directory.
    - Data in folders Data/Lab3/Train and Data/Lab3/Validation.
- Python/Lab3/1-diff_features.py (Task 2)
    - To run: python3 Python/Lab3/1-diff_features.py --label_folder Data/Lab3/[Folder]
        - Validation, Adversary, or AltTest
    - Results in Data/Lab3/Results/1-val-report.txt (classification report included)
- Code for training with different sample sizes per user can be found in Python/Lab3/1-diff_size_training.py and 1-alt_create_validation_set.py
    - Results for training with different sample sizes per user can also be found in Data/Lab3/Results (1-10samples-val-report.txt and 1-5samples-val-report.txt)

Task 3 Files:

- Labeled adversary data in Data/Lab3/Adversary
    - To run: python3 Python/Lab3/2-detect_adversary.py --label_folder Data/Lab3/[Folder]
        - Validation or Adversary
    - Results in 2-adv-report.txt and 2-val-report.txt

Online references:

"1.10. Decision Trees." *Scikit*, scikit-learn.org/stable/modules/tree.html. Accessed 6 June 2024.

"Classification_report." *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html#sklearn.metrics.classification_report. Accessed 6 June 2024.

"Comparing Support Vector Machines and Decision Trees for Text Classification." *GeeksforGeeks*, GeeksforGeeks, 21 Mar. 2024, www.geeksforgeeks.org/comparing-support-vector-machines-and-decision-trees-for-text-classification/.

"GridSearchCV." *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed 8 June 2024.

"How to Use Logistic Regression Predict_proba Method in Scikit-Learn." *Saturn Cloud Blog*, 30 Dec. 2023, saturncloud.io/blog/how-to-use-logistic-regression-predictproba-method-in-scikitlearn/.

Kavlakoglu, Eda, and Karina Kervin. “Classifying Data Using the SVM Algorithm Using Python.” *IBM Developer*, 7 Mar. 2024, developer.ibm.com/tutorials/awb-classifying-data-svm-algorithm-python/.

Lakshmi, Vijaya. “Case Study : SVM vs Decision Tree in Machine Learning.” *LinkedIn*, 29 Aug. 2023, www.linkedin.com/pulse/case-study-svm-vs-decision-tree-machine-learning-vijaya-lakshmi/.

Lakshmi, Vijaya. “Case Study : SVM vs Decision Tree in Machine Learning.” *LinkedIn*, 29 Aug. 2023, www.linkedin.com/pulse/case-study-svm-vs-decision-tree-machine-learning-vijaya-lakshmi/.

Pramod, Om. “Decision Trees.” *Medium*, Medium, 29 Jan. 2023, medium.com/@ompramod9921/decision-trees-8e2391f93fa7.

Scikit-learn. “Scikit-Learn/Sklearn/Svm/_base.Py at 5491dc695dbe2c9bec3452be5f3c409706ff7ee7 · Scikit-Learn/Scikit-Learn.” *GitHub*, 10 Apr. 2023, github.com/scikit-learn/scikit-learn/blob/5491dc695dbe2c9bec3452be5f3c409706ff7ee7/sklearn/svm/_base.py#L135.

“SVC.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.predict_proba. Accessed 8 June 2024.

“W3schools.Com.” *Python Machine Learning Train/Test*, www.w3schools.com/python/python_ml_train_test.asp. Accessed 8 June 2024.