Machine learning goals and challenges: Classification is a machine learning problem where the goal is to learn a discriminative model (function and its parameters) in a supervised setting: Given a set of samples having multi-dimensional features, for which outcome measurements exist, predict the outcome for a new sample. The development of robust and efficient machine learning models is a two-fold data science approach: (1) data exploration, (2) model selection and evaluation. As a first step, exploratory analysis of the data is essential to gain insights into the problem as well as into the data and to identify the underlying modeling challenges. Moreover, data exploration also helps to formulate the appropriate modeling assumptions and experimental design while considering the resources at hand. As a second step, selection of the optimal model is crucial such that it generalizes to the new data. The optimal model can be selected based on the performance metric obtained through rigorous evaluation of the modeling assumptions with the principled experimental design.

What are the modeling challenges(?): In the proposed problem, exploratory analysis of the data identified two fundamental modeling challenges: (1) imbalanced class distributions and, (2) extremely high dimensionality. Furthermore, the analysis also revealed non-linear structure from the data. Class imbalances are an inherent characteristic of many real-world machine learning problems and require particular attention while developing models. Likewise, in this data, the proportion of majority class was $\sim$90%, and the minority class was $\sim$10%. On the other hand, increasing data dimensionality poses difficulties inferring model and parameters correctly.

What is my solution(?): To solve the challenges mentioned above, my solution combined three machine learning principles: (1) dimensionality reduction, (2) non-linear classification and (3) sub-sampling (to account for class imbalances).

In the realm of high dimensionality, big data samples are critical to infer model parameters accurately. But this is not practical in many cases, and more advanced machine learning models can be developed that favor sparse solutions (imposing regularization that shuts down the unnecessary features). Alternatively, dimensionality reduction is a more straightforward and ready-to-implement solution. To this end, matrix factorization has evolved as the state-of-art tool for dimensionality reduction and data visualization. Essentially Principal Component Analysis (PCA) is a matrix factorization approach that decomposes the observed high dimensional data into multiple low-dimensional latent factors (also known as "components"). The underlying assumption is that the combination of multiple latent factors has generated the high-dimensional data, corrupted with some noise. However, each combination has generated some parts of the data. The machine learning aim is then to learn these components thereby capturing the strongest variation patterns for individual features in a principled way. In this way, the latent factors can then be seen as "noise-free" low-dimensional representation of the data and the replacement for high-dimensional noisy observations.

From the modeling perspective, linear methods are a natural choice, since they are interpretable, easy-to-understand and provide direct information on the relationship between the features and outcome. However, they neglect the relevant non-linear structure in the data. As a result, the nonlinear-methods provide better predictions by modeling complex interactions, of course, compromising the interpretability. As the primary focus of the problem was purely a prediction task (no interpretability was desired), I resort using the state-of-the-art non-linear machine learning models.

Finally, to tackle class imbalances, I chose a "down" sampling approach. Mainly, this type of sampling randomly picks subset of samples from the majority class to match the sample size of the minority class. In the current set-up, this choice is plausible since there are available much more samples from a majority class and picking their subsets at random would not only reduce the noise but also favor learning a balanced classifier.

How did I implement the solution(?): Since my knowledge about the given problem

was limited, therefore it was difficult to formulate the correct assumptions to model non-linear relationships. I, therefore, compared various machine learning models and chose the model that provided best classification accuracy and is better generalizable. To put into context, I evaluated Random Forest (RF), Decision Trees (DT), K Nearest Neighbor (KNN) and Linear Logistic Regression (LR) on the task of classifying new samples into two classes. Linear regression was compared as a baseline approach to validate the non-linear assumptions.

As an experiment design process, I partitioned the given labeled data into training (90%) and internal held-out set (10% to be used for validation purpose), randomly. Since data exploration already revealed class imbalances, I created stratified ten folds of the training data, such that the ratios of majority vs. minority classes were kept consistent across all the folds. The held-out data had the same consistency of the class distributions as observed from the labeled data. In case of class imbalances, stratified K-folds is central to the experimental design since it can significantly help to avoid overfitting on the training data.

In practice, I performed 10-fold cross-validation (CV) on the training data, wherein each fold of the CV run, 1/10th of the data was held-out as a test set at random, while the model was trained on the other 9/10 data. Since each model has specific parameters to be tuned, a nested CV procedure was adopted for that purpose. Again, for creating the nested CV folds, the class imbalances were dealt with stratification. For each fold of CV runs, PCA was performed on the training data to estimate the components and test data was then transformed into the PCA space by using components loadings. Any normalization/preprocessing/dimensionality reduction that is solely done on training data is crucial to avoid over-fitting that could be caused by the leakage of test data into the training phase (also known as double-dipping). Lastly, for each modeling approach, predictive performances were computed across the predictions obtained from a complete round of CV experiment. Mainly, the performance metrics that are important to estimate the effect of class imbalances were used to evaluate the efficient model.

How good is the solution(?): Table 1 shows the balanced accuracies, i.e., (Sensitivity + Specificity)/2 obtained from a CV run for each of the methods. The higher the value, the better is the model. The RF, DT, and KNN provided improved classifications than LR, validating the non-linear modeling assumptions. KNN is slightly better than the other two models. But the difference is quite small and may not be statistically significant. Since, the random forest is a kind of ensemble based learning approach and is expected to maintain a reasonable tradeoff between the bias and variance, thus better generalizable to new data, as compared to KNN. I chose random forest for learning the model of complete labeled data and generated predictions for the test set. Meanwhile, predictions on the internal held-out set (validation set) gave an accuracy of 0.97 when tested with random forest, thereby validating the proposed solution.

Table 1: Accuracies from a cross-validation experiment

RF	DT	KNN	LR
0.95	0.94	0.96	0.52

What does that mean practically(?): But the important question is what does this classification accuracy imply in practice. To find answer lets look into the confusion matrix shown in Table 2. 155 samples (∼5%) have been wrongly classified as -ve class, 15 samples (only ∼4%) have been misclassified as +ve labels. Now, let's assume this solution has been deployed as

Table 2: Confusion Matrix: Random Forest Model

		Observation	
Prediction		1	-1
	1	2878	15
	-1	155	328

cybersecurity AI assistant to detect outliers/fraudulent/intruders or threats. In this case, the intruders are denoted by the minority class (i.e., -ve labels). Naturally, the goal here would be to detect the intruders with higher accuracy as much as possible. In other words, the optimal solution should minimize the error on the minority class. Otherwise, it would raise serious concerns about the credibility of the AI assistant. On the contrary, classifying +ve labels as threats will be initially quarantined and later on will be cleared by the malware/anti-virus software without increasing any overhead on resources. In summary, 96% of the times my solution will be correct for identifying the intruders while minimal resources would be needed to handle only 4% of the cases.

How can the solution be improved (?): If there are available more resources (regarding time and computing power) then perhaps I would get estimates of accuracies from the 10-fold cross-validation procedure that is repeated multiple times with different random cross-validation stratified folds. Also to minimize over-fitting at the zero-ish level, I will adopt the nested CV so that the normalization/preprocessing is done separately for each of the nested folds too. There exist various other sampling approaches such as "up", "smote" and "rose" to tackle class imbalances, it is worth testing them. Non-linear dimensionality reduction techniques, for instance, t-distributed stochastic neighbor embedding (t-SNE) could be explored when non-linearities are inherent in the data. Moreover, a bunch of probabilistic machine learning models can also be tested, as the Bayesian approach offers systematic solutions to handle uncertainties in the data that arise due to high dimensionality and noisy observations, in addition to providing confidence intervals on the classification estimates. Lastly, a new machine learning model or customized solution (tailor-made cybersecurity AI assistant) can be further developed to offer personalized protection, given detailed knowledge about the data, the problem and operational use case.