Support Vector Machines work by finding the hyperplane separating the instances of the dataset, the support vectors that run alongside the hyperplane, and the margin between the hyperplane and the support vectors. The hyperplane acts as a decision boundary for deciding the class of an instance in classification and if an instance falls on one side it becomes that class and if it falls on the other side it becomes the other class. The hyperplane is found by finding the maximum margin between possible support vectors in the data set, and the support vectors are found by finding the closest instances to the decision boundary from each class. The shape of the hyperplane can be changed in a support Vector Machine based off the type of kernel the machine decides to use, with a linear kernel the hyperplane will be a linear separation between data instance, a polynomial kernel will result in a curved hyperplane, and a radial kernel can result in a circular hyperplane. Support Vector Machines also have hyperparameters that can be tuned to get better results for the model created, like cost which allows for several instances to be classified incorrectly and gamma which controls the bias-variance tradeoff of the model.

The strengths of Support Vector Machines are that they can be very robust when faced with outliers in the data due to the margins they have being reliant on support vectors. The ability for a Support Vector Machine to be used for both regression and for classification is useful, and the ability to use different kernel types and tune hyperparameters to optimize your model allows Support Vector Machines to be extremely flexible as an algorithm. Some of the weaknesses of Support Vector Machines are that they can take more time to execute than other types of machine learning models. When faced with a large data set with lots of instances and parameters the computation time needed to build the model increases by a lot. Additionally, more time and instances are needed to tune the Support Vector Machine to find the optimal hyperparameters, needing to split a dataset into train, test, and validate instead of just train and test.

Random forest works like how the Decision Tree algorithm works but with a few differences. At every branch in the tree being made by the algorithm the random forest algorithm will select a subset of the available predictors randomly and use only those predictors to generate the branch split. The xgboost algorithm is different from a decision tree algorithm in that it requires for an input a matrix for the parameters of the training data, and a numerical label vector to hold the classification as 0 or 1. This means that given a data field you must first separate and organize that data field into a matrix and label. When given these parameters you set several iterations for the algorithm to iterate over and during each iteration the algorithm will use a different tree, every new tree builds off the results of the older trees gradually reducing errors and finding the optimal tree. The adaboost model trains several learners equal to the number of observations in the data set. Initially each of the learners will have an equal amount of weight in the model but the algorithm will go through several iterations where accurate trees are given more weight and inaccurate trees are given less weight. If the learners are decision trees the trees will start out small and generalized, but over several iterations of running the algorithm adaboost will build the new trees off the old trees.

The strength of random forest is that if you have a really strong predictor value for your dataset the tree generated won't solely use that value because every branch uses a randomly selected subset of predictors. The strength of xgboost is that it is very fast at executing by making use of C++ multi-threading the xgboost algorithm is able to generate a model quickly, additionally missing values in the given data are handled internally by the algorithm so you

don’t have to remove them. The strength of the adaboost algorithm is that it does not given an equal weight to every learner it has causing it to overfit less than other algorithms. The weakness of random forest is that it is slower than the traditional decision tree algorithm because it needs to build multiple trees. The weakness of xgboost is that it can end up overfitting the given data if the number of tree branches are not pruned. The weakness of adaboost is that it does not preformed well when given a dataset that has lots of outliers, and it takes a lot of time to run being very slow when compared to other algorithms (fast adaboost helps speed up computation but limits classification to binary classification).