**General Approach:**

* First, both the data files were imported and stored into a pandas dataframe.
* Second, missing training/testing data were visualized using seaborn’s heatmap function.
* Third, ‘YES', 'NO' values were replaced with '1' and '0' in training/testing data and the missing values were filled with the corresponding data mean values.
* Fourth, One-hot Encoding was used to convert the categorical features to dummy variables using pandas get\_dummies function.
* Fifth, redundant feature columns were dropped after One-Hot encoding
* Finally, we had the Training Data Shape: (245725, 46) and

Testing Data Shape: (105312, 46) for training and testing respectively.

**Model Selection (Model 1 Random Forest with RandomizedSearchCV, training time = 22.01Hrs, score = 0.87097):**

* Random Forest was selected as the model for the given classification problem over SVM because of its speed of computation on large datasets. This is because, If we have data with nn points and mm features, an intermediate step in SVM is constructing an n×nn×n matrix (think about memory requirements for storage) by calculating n2n2 dot products (computational complexity). Therefore, as a rule of thumb, SVM is hardly scalable beyond 10^5 points. Large number of features (homogeneous features with meaningful distance, pixel of image would be a perfect example) is generally not a problem.
* For a classification problem Random Forest gives us probability of belonging to class. SVM gives us distance to the boundary, we still need to convert it to probability somehow when we need probability (like in this case).
* Grid search was used to determine the optimal values for hyperparameters; we wanted to identify a model structure that performs best on records it has not been trained on. For this a variety of hyperparameter values were considered. For example, below are some candidate hyperparameters.
  + **For number of trees**: n\_estimators = [100,150,200,250,300]
  + **For max number of features to consider at every split**: max\_features = ['auto','sqrt','log2']
  + **For max number of levels in a tree**: max\_depth = [10,20,30,40,50]
  + **For minimum number of samples required to split a node**: min\_samples\_split = [2,5,10,15,20]
  + **For minimum number of samples required at each leaf node**: min\_samples\_leaf = [1,2,5,10,15]
  + **(Best Parameters obtained in our case)**:

n\_estimators = 200

max\_features = sqrt

max\_depth = None

min\_samples\_split = 20

min\_samples\_leaf = 15

Random forest models typically perform well with default hyperparameter values, however, to achieve maximum accuracy, optimization techniques can be worthwhile.

**Model Selection (Model 2 XGBoost with RandomizedSearchCV, score = 0.87236) (Final Model):**

* **Speed and Performance** - XGBoost come equipped with several speed enhancements, making it possible to train a well-performing model in a short amount of time. XGBoost and similar libraries can also be trained on GPUs (some assembly required), making training even faster on larger datasets.
* **Gradient Boosting (GB)** methods are powerful classifiers/regressors that typically perform very well on structured data, and the XGBoost library is an excellent implementation of this algorithm.
* **It is an ensemble learning algorithm**, which combines the predictions of multiple base learners (usually, each one being a fairly weak performer on its own) to generate one overall prediction for each input/example. This allows it to learn more complex relationships between the features and targets/labels in the training set.
* **Core algorithm is parallelizable**
* **Consistently outperforms single-algorithm methods**
* **Best Parameters obtained in our case –**

Learning Rate: 0.1

Gamma: 2

Max Depth: 7

Subsample: 0.7

Max Features at Split: 0.6

Alpha: 0.5

Lambda: 1

Minimum Sum of the Instance Weight Hessian to Make a child: 3

Number of Trees: 100