\section{Conceptual questions}

\subsection{Explain how do we control the data-fit complexity in regression trees.}

The data-fit complexity in a regression tree can be controlled by limiting the size of the tree because too large of a tree can overfit the data and too small of a tree can underfit the data. This can be achieved through a number of methods and possibly a combination of said methods. Some lower-level data-fit complexity controlling methods include: controlling the depth of a tree, controlling the number of leaf nodes of a tree, and controlling the number of training data points that a node or leaf node can contain. On a higher level, one can use tree pruning and examine the data-fit complexity tradeoff by growing a large tree, stopping when a minimum node size has reached and then performing pruning by minimizing the cost function. The goal here is to measure the data fitting error (using a specific measure) of each node and keeps nodes with high predictive power. A typical approach is to test if the deletion of a node causes no or little change in error and if so, delete that node. A similar approach can be taken but this time by examining the explained variance after deleting the node.

\subsection{What’s the main difference between boosting and bagging?}

Although there are many similarities between boosting and bagging, the main difference between the two lies in the fact that boosting uses weak learners on weighted data sets, i.e. data points have different weights such that newer models do well where the previous ones did not. Additionally the weak learners are combined linearly via a weighted average with more wight on the models that performed best on training data. Bagging on the other hand runs weak learners using bootstrapping and non-weighted data and get combined in the end by simply averaging the weak learners (same weight for each).

\subsection{Explain how OOB errors are constructed and how to use it to understand a good choice for the number of trees in random forest. Is OOB error test or training error and why?

OOB errors are constructed by examining the classification of each point in a set using only the trees that were trained using bootstrap samples that did not include said point. Then, OOB error is obtained by comparing all predictions in the set to ground truth labels as usual. By further examining the OOB error when using a different number of trees, the OOB error can be used to determine a good choice for the number of trees by seeing and using the number of trees when the OOB error saturates or stabilizes (which often differs compared to using test error). OOB error is a test error because the classification of OOB sample points is completed by using trees which have not seen the points and therefore the OOB samples are all considered unseen data.

\subsection{Explain what is ``kernel trick” and why it is used?}

The kernel trick involves considering the inner product of features maps instead of expanding features explicitly. This inner product of feature maps is represented through valid kernel functions (many choices for kernel function so long as the Gram matrix is positive semi-definite). Therefore the feature map is not directly constructed. The trick is particularly useful when the data has many features and transformations of this data can include polynomial combinations of the features which leads to an extremely large number of terms and large computational costs. So the kernel trick is used such that the objective function uses inner products of the transformed feature vectors i.e. the kernel function. This trick now dramatically decreases the computational complexity (O(n)) and cost which is ultimately why it is used. Additionally, the kernel trick allows for a linear decision boundary in the transformed non-linear feature space i.e. a non-linear decision boundary in the original feature space. Often this means higher dimensional transformations can assist with prediction.

\section{Random forest and on-class SVM for email spam classifier}

In this problem I use the Spambase dataset and evaluate the effectiveness of a decision tree, a random forest, and a one-class SVM (novelty detector) classifier in the classic problem of email spam filtering. This dataset consists of 4601 emails labeled as spam (1) or not (0) and the collection of non-spam emails came from filed work and personal emails and hence the occurrence of the word, ``George”, and the area code, ``650”, as indicators in the features of the dataset. This is common for constructing a personalized email spam filter. Per the UCI Machine Learning Repository, there are 57 attributes in this dataset, 48 of which are concerned with the frequency of a specific word, 6 of which are concerned with the frequency of a character, and 3 that measure the length of sequences of consecutive capital letters in different ways.

\subsection{CART model)

I first construct and visualize a classification tree using the data by taking advantage of the sklearn library in Python. For simplicity here, I use all of data for constructing the tree. Using entirely default parameters, the tree grows quite large reaching a depth of 34 with 299 leaves. Figure 1 shows the visualization of this default-type tree. In this tree, the minimum number of samples required to be at a leaf node is only one. When using a dataset of this size, this can greatly increase the chance of overfitting to the training data, so I use a general rule of thumb to better construct a new decision tree. The rule of thumb is that no leaf node contains less than 5\% of the training data. After applying this simplification, the constructed tree becomes much smaller having a depth of 10 and 16 leaves. This ``improved” or more general tree is shown in Figure 2.

\subsection{Random forest model versus classification tree}

Now I build and train a random forest model to compare the effectiveness of a random forest compared to a decision tree for this task. I shuffle and split the dataset for training (using 80/% of the data) and testing (using 20/% of the data). Because the ultimate goal is to make predictions on unseen data and to compare models, I retrain the decision tree using just the training set and do the same for the random forest model. As I have learned, the number of trees used in the forest can greatly impact the test error and model effectiveness so I tune the number of trees used in the final model by first analyzing the test error using a different number of trees. The goal is to see when the test error saturates and use the number of trees where it does so, this keeps the final model simple and general, yet effective. Figure 3 shows this analysis; specifically it shows the test error (total misclassification error rate) versus the number of trees for both the random forest and the re-trained decision tree. As expected, the test error for the decision tree is constant when varying the number of trees. From this analysis, I see that the test error for the random forest model is minimized when the number of trees used is 500. Table 1 summarizes the final results for comparing these two models when using the test set. It is clear from Figure 3 and Table 1 that the random forest model with a tuned number of trees achieves a lower test error and better results. Generally, it can be seen that the random forest model achieves better results when there is a sufficient number of trees.

TALK ABOUT A COMPARISON BETWEEN THE FINAL TEST ERRORS HERE.

\subsection{One-class SVM for spam filtering}

Lastly,