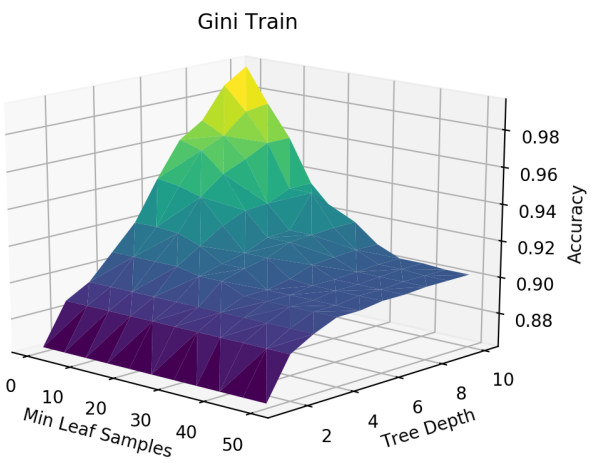
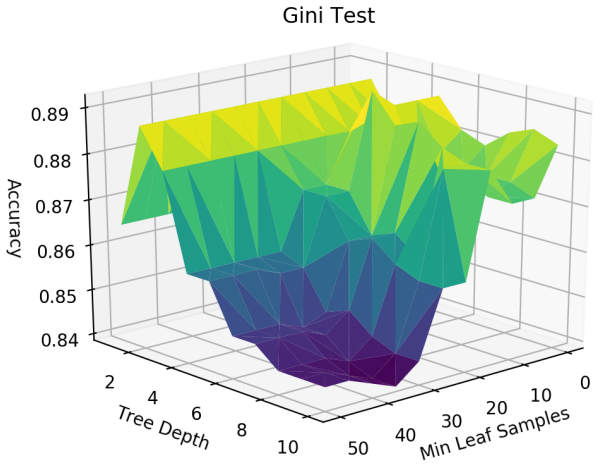
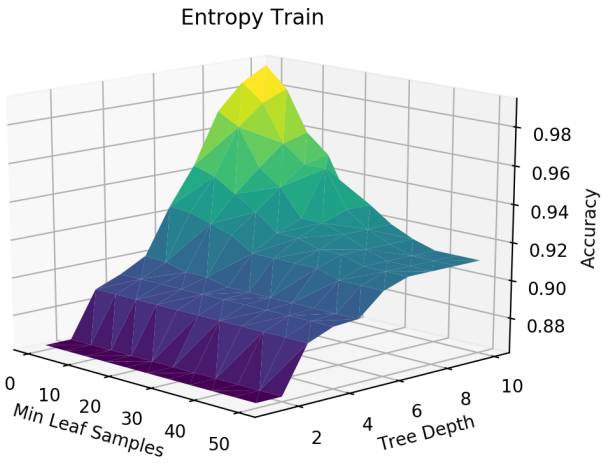
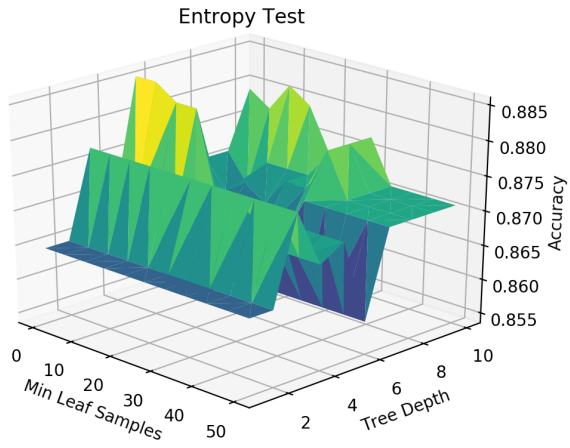
QUESTION 1C









QUESTION 1D

<u>Train Function Time Complexity</u>

- The `train` function calls the `build_tree` function, which is recursive function forms a binary recursive tree which contains 2^p nodes.
- At each recursive / tree level we process a maximum of *n* data points since the data is either passed on to a node at the next level or stops at a leaf.
- For each node, we need to find the best feature and the best split value using the `find_best_feature` function.
- In the build_tree function, for each node we add, we call the `find_best_feature` function passing through on average n / 2^p of the data.

- Since there are 2^p nodes at each level, with each node processing n / 2^p of the data each level processes at most *n* data points.
- Since the same amount of total data is processed over each level using the work done at each level can be assumed to be the work done by the `find_best_feature` function
- Therefore, the time complexity for the train function is

  $O(p) \times O(\texttt{find\_best\_feature})$

  where *p* is the depth of the decision tree.
- In the `find_best_feature` function, we iterate through all features where for each feature we find the best split value using the `find_best_split` function
- Therefore, the time complexity of the train function is:

  $O(p) \times O(d) \times O(\texttt{find\_best\_split})$

  where *d* is the number of features in the input data.
- In the `find_best_split` function, we iterate through all possible split values of dataset. This value can vary based on the number of flips in the dataset when it's ordered by the respective feature. The worst this value can be is if there are label flips every alternate datapoint which will result in the number of split values to be *n*
- For each split value, we split the dataset at that value using the `get_split` function and calculate the entropy / gini index of the split using the `calculate_entropy` or `calculate_gini` function.
- Therefore, the time complexity of the train function is:

$O(p) \times O(d) \times O(n) \times (O(\texttt{calculate\_entropy}$ or $\texttt{calculate\_gini}) + O(\texttt{get\_split}))$

  where *d* is the number of features in the input data.
- Calculating the entropy and gini index both take $O(n)$ to count the number of points falling into each class
- Splitting the data also takes $O(n)$ as it iterates through each data point to put it into the respective bucket based on how its feature value compares to the split value

Therefore, the total time complexity of the train function is

$= O(p) \times O(d) \times O(n) \times (O(n) + O(n))$

$= O(p) \times O(d) \times O(n) \times O(n)$

$= \boldsymbol{O(pdn^2)}$

Predict Function Time Complexity

The predict function iterates over all *n* datapoints. For each datapoint, it runs it through the decision tree until it reaches a leaf. The most nodes each datapoint will pass through until it gets classified is equal to depth of the tree *p*.

Therefore, the time complexity of the predict function is ***O(np)***

QUESTION 2D

	Strategy	TrainAUC	ValAUC	Time
0	Holdout	0.960111	0.778959	0.011925
1	2-fold	0.955713	0.778253	0.010519
2	5-fold	0.952962	0.790454	0.033221
3	10-fold	0.954095	0.788652	0.068606
4	MCCV w/ 5	0.946533	0.759180	0.030486
5	MCCV w/ 10	0.944186	0.765115	0.073768
6	True Test	0.954458	0.834606	0.000000

With regards to the AUC, the holdout strategy seems to overestimate the performance of the model for TrainAUC and underestimate the performance based on the ValAUC. Since it runs on only a single train-test split it is possible for it to give a biased accuracy based on the proportion of outliers in the training vs testing data.

The k-fold and Monte Carlo strategies seem to provide a more accurate version of the accuracy since the accuracy has been found from the average of the accuracy over multiple trails. The greater the number of trials, the more the biases cancel each other out.

The k-fold strategy provides the accuracy values closest to the true accuracy of the model as compares to other strategies

The runtime of all strategies are more or less the same, with the runtime of k-fold and mc depending on the value of k and s respectively

QUESTION 3D

Optimum hyperparameters for K Nearest Neighbors:
k = 15

Optimum hyperparameters for Decision Tree:
max_depth = 6
min_samples_leaf = 48

```
K Nearest Neighbors ---------------
  Proportion   TestAUC  Accuracy
0       100%  0.524745  0.862500
1        99%  0.524745  0.862500
2        95%  0.533642  0.866667
3        90%  0.525950  0.864583

Decision Tree ---------------
  Proportion   TestAUC  Accuracy
0       100%  0.655978  0.887500
1        99%  0.655978  0.887500
2        95%  0.655978  0.887500
3        90%  0.640593  0.883333
```

Both the TestAUC values and Accuracy values are higher for the Decision Tree as compared to K Nearest Neighbors.

In terms of the sensitivity, neither KNN nor DT were affected by losing 1% of their data since their TestAUC and Accuracy remained the same for 100% of the data and 99% of the data.

On losing 5% of their data, the TestAUC and Accuracy of KNN oddly improved, however this may have been due to the elimination of outliers. DT was again unaffected by the removal of 5% of data.

On losing 10% of the data however, the TestAUC and accuracy of both KNN and DT were affected with the TestAUC and Accuracy of KNN dropped slightly for 90% of the data as compared to 95%, but was still higher than all 100% of the data. DT, on the other hand, dropped in TestAUC and Accuracy.