Homework 4

Matthew Carey

Problem 1:

a)
```
The binary tree structure has 15 nodes.
- depth   0 has    1 nodes, of which    0 are leaves
- depth   1 has    2 nodes, of which    0 are leaves
- depth   2 has    4 nodes, of which    0 are leaves
- depth   3 has    8 nodes, of which    8 are leaves
The decision tree:  (Note: Y = 'yes' to above question; N = 'no')
Decision: X['great'] <= 0.50?
  Y Decision: X['waste'] <= 0.50?
    Y Decision: X['disappoint'] <= 0.50?
      Y Leaf: p(y=1 | this leaf) = 0.504 (496 total training examples)
      N Leaf: p(y=1 | this leaf) = 0.130 (46 total training examples)
    N Decision: X['so-cal'] <= 0.50?
      Y Leaf: p(y=1 | this leaf) = 0.000 (36 total training examples)
      N Leaf: p(y=1 | this leaf) = 1.000 (1 total training examples)
  N Decision: X['bad'] <= 0.50?
    Y Decision: X['terribl'] <= 0.50?
      Y Leaf: p(y=1 | this leaf) = 0.777 (188 total training examples)
      N Leaf: p(y=1 | this leaf) = 0.125 (8 total training examples)
    N Decision: X['bought'] <= 0.50?
      Y Leaf: p(y=1 | this leaf) = 0.000 (13 total training examples)
      N Leaf: p(y=1 | this leaf) = 0.800 (5 total training examples)
```

b) Using GridSearchCV, we found that the best values for `max_depth` and `min_samples_leaf` are 128 and 9, respectively. To reduce overfitting in a model, typically you would increase the min_samples_leaf hyperparameter. This is because increasing this value will constrain the tree to have more samples in each leaf, preventing it from growing too deep and overfitting to unimportant noise in the training data.

c)
```
The binary tree structure has 13 nodes.
- depth   0 has    1 nodes, of which    0 are leaves
- depth   1 has    2 nodes, of which    0 are leaves
- depth   2 has    4 nodes, of which    1 are leaves
- depth   3 has    6 nodes, of which    6 are leaves
The decision tree:  (Note: Y = 'yes' to above question; N = 'no')
Decision: X['great'] <= 0.50?
  Y Decision: X['excel'] <= 0.50?
    Y Decision: X['disappoint'] <= 0.50?
      Y Leaf: p(y=1 | this leaf) = 0.430 (4041 total training examples)
      N Leaf: p(y=1 | this leaf) = 0.114 (368 total training examples)
    N Decision: X['it_was'] <= 0.50?
      Y Leaf: p(y=1 | this leaf) = 0.905 (263 total training examples)
      N Leaf: p(y=1 | this leaf) = 0.643 (28 total training examples)
  N Decision: X['return'] <= 0.50?
    Y Decision: X['bad'] <= 0.50?
      Y Leaf: p(y=1 | this leaf) = 0.745 (1413 total training examples)
      N Leaf: p(y=1 | this leaf) = 0.415 (142 total training examples)
    N Leaf: p(y=1 | this leaf) = 0.275 (91 total training examples)
```

d) In the first decision tree, at depth 3, there are two instances where an internal node has two child leaf nodes predicting the same sentiment class. Specifically, the internal node where the decision is based on whether 'waste' is less than or equal to 0.50 and whether 'so-cal' is less than or equal to 0.50, both result in child leaf nodes predicting positive sentiment (y=1). Similarly, another internal node at the same depth, where the decision is based on whether 'terribl' is less than or equal to 0.50 and whether 'bought' is less than or equal to 0.50, leads to child leaf nodes also predicting negative sentiment (y=0).This occurrence makes sense because it reflects how decision trees partition the feature space. At each internal node, the algorithm evaluates a feature and splits the data based on a certain threshold. If the resulting subsets contain mostly samples of the same class, the algorithm may continue splitting until further splits don't significantly improve the purity. Thus, if both resulting subsets at a node mostly belong to the same sentiment class, it's reasonable for the algorithm to create child leaf nodes that predict that class. This behavior aligns with the concept of Gini impurity, where the goal is to minimize impurity at each node by segregating samples into homogeneous groups. Therefore, having two children predict the same class indicates that the model is confident about the predictions for both subsets represented by the children.

Problem 2:

a) Below is the most and least important words determined by the random forest:

	Top Words	Bottom Words
1	easy_to	first_few
2	worst	years_after
3	poor	always_be
4	love	me_about
5	excel	domin
6	great	several_other
7	disappoint	the_night
8	bad	you_had
9	return	toaster_oven
10	waste	right_to

Figure 2a: Bottom Words were the lowest of 791 that were below the threshold of 0.00001

b) The best hyperparameters found for the forest are as follows: `'max_depth': 32, 'max_features': 33, 'min_samples_leaf': 1, 'n_estimators': 100.` The maximum value for max_features is the size of the vocabulary that we are using, which is 7729. It is important to tune the value of max_features rather than setting it to this max value, because this can prevent overfitting. If every vocab word were considered at each split, in addition to being computationally expensive, this would make each tree much more susceptible to capturing unwanted noise that exists in the data, which leads to overfitting.

c) The `n_estimators` hyperparameter in random forests balances model complexity and generalization performance. Increasing `n_estimators` typically decreases the variance of the model's predictions while potentially increasing bias, and vice versa. Setting `n_estimators` excessively large will not cause overfitting to the degree of other hyperparameters, and it can even reduce the likelihood of overfitting due to the ensemble nature of random forests. However it is possible to cause overfitting if it memorizes noise in the data, though it is far less likely than most other hyperparameters. After playing with the value of `n_estimators` in our best forest, we can see that setting it excessively large does not have particularly adverse effects on the validation set, which is typical in situations of overfitting.

Problem 3:

a) The best hyperparameters for the gbdt classifier were learning_rate=0.01, max_depth=32, max_features=3, and n_estimators=300. Similar to random forests, because gradient boosting is quite robust to overfitting, a large value will typically result in a better performance, and there is not much concern for overfitting. For shallow trees, Gradient Boosting can still work, because it is an ensemble learning method that combines multiple weak learners to form a strong learner. Even though individual shallow trees may have limited power and can't capture complex patterns in the data, the ensemble as a whole can compensate for this limitation by iteratively focusing on the areas where the previous weak learners fell short. In fact, GBDT is typically done with shallower trees compared to individual tree classifiers.

b) The use of friedman_mse as the criteria is not weird, because although mse may imply a regression based task, friedman_mse is a variant that is specifically designed for decision trees within the context of boosting.

c) The following is the top and bottom most important words as determined by the GBDT classifier:

	Top Words	Bottom Words
0	poor	con
1	a_great	ending_is
2	bore	domin
3	the_best	in_just
4	bad	one_point
5	easy_to	sunday
6	disappoint	host
7	waste	the_hero
8	excel	i_strongly
9	great	hold_it

Words with importance below 0.00001 threshold: 69

Figure 3c: most and least important words as determined by the GBDT classifier

Problem 4:

a)

Method	Max Depth	Num Trees	Train BAcc	Validation BAcc	Test BAcc
Decision Tree	128	1	0.8308	0.7296	0.7127
Random Forest	32	100	0.9690	0.8373	0.8342
Gradient Boosted Trees	32	300	0.9943	0.8487	0.8522

Figure 4a: overview of hyperparameters and performance for all the classifiers of this report