5.5: How well does the meta-tree do on the test set? Why does training a decision tree on the combination of original data and model predictions perform so well?

5.6: Suggest one way to improve on the model above

In [17]:
```
# will produce a warning under most versions of SKlearn, but it should be OK to i
# if you get weird errors or the models all stink, let us know

import pickle
with open("data/models.pkl", 'rb') as infile:
    model_dict = pickle.load(infile)
```

Answers:

5.1: Report each model's score on the test set, so that you can compare to these scores later.

In [18]:
```
models = ['Ada', 'KNN', 'Logit', 'QDA', 'RF']

for i in models:
    print('{} Accuracy on test set: {}%'.format(i, \
            round(accuracy_score(y_train, model_dict[i].predict(x_test))*100, 2))
```

```
Ada Accuracy on test set: 50.18%
KNN Accuracy on test set: 51.48%
Logit Accuracy on test set: 51.26%
QDA Accuracy on test set: 51.08%
RF Accuracy on test set: 49.7%
```

5.2: Read in the fresh dataset `data/Higgs_tune.csv`. Similar to 2.1, build `ensemble_tune` and `ensemble_test`, datasets containing each tuned model's prediction of P(this point belongs to class 1) for each of the tuning and test points.

In [19]:
```
higgs_tune = pd.read_csv('data/Higgs_tune.csv')
x_tune = higgs_tune.iloc[:, higgs_tune.columns != 'class']
y_tune = higgs_tune['class']

ensemble_tune = []
ensemble_test = []

# dataset_x should be a pandas dataframe
for i in models:
    ensemble_tune.append(model_dict[i].predict_proba(x_tune)[:,1])
    ensemble_test.append(model_dict[i].predict_proba(x_test)[:,1])

ensemble_tune_prob = np.array(ensemble_tune).reshape(5, len(x_tune)).T
ensemble_test_prob = np.array(ensemble_test).reshape(5, len(x_test)).T

# compute average using prediction probabilities of class 1
ensemble_tune_pred = 1*(ensemble_tune_prob.mean(axis = 1) > 0.5)
ensemble_test_pred = 1*(ensemble_test_prob.mean(axis = 1) > 0.5)

print('Ensemble Accuracy on tune set: {}%'.format(round(accuracy_score(y_tune, en
print('Ensemble Accuracy on test set: {}%'.format(round(accuracy_score(y_test, en
```

```
Ensemble Accuracy on tune set: 65.6%
Ensemble Accuracy on test set: 65.88%
```

5.3: Build a meta-model trained on `ensemble_tune` and predicting the tuning set labels (e.g., a LogisticRegression or RandomForest). Which model does your meta-model consider most important, and how well does your meta-model perform on the test set?

In [20]:
```
# Learn the optimal weights and intercept
fitted_logreg = LogisticRegressionCV().fit(ensemble_tune_prob, y_tune)
print('Model Names: {}'.format(models))
print('Model coeffecients: {} (One weight for each model)'.format(fitted_logreg.c

# use weights and intercept to combine the test data predictions
y_hat = fitted_logreg.predict(ensemble_test_prob)

print('Test accuracy (Classify by LogReg on individual predictions): {}%'.format(
```

```
Model Names: ['Ada', 'KNN', 'Logit', 'QDA', 'RF']
Model coeffecients: [[ 8.22652277 -0.21155845  1.28924669  0.42076795  3.067060
45]] (One weight for each model)
Test accuracy (Classify by LogReg on individual predictions): 69.56%
```

The meta-model considers the Adaboost model most important followed by the RandomForest and Logit models. The linear weights associated with each model can be seen above in the model coefficients output. The overall accuracy when using the meta-model trained with Logistic Regression is even more accurate on the test set than the majority rule meta-model in Q5.2.

5.4: Augment the `ensemble_tune` and `ensemble_test` datasets with the columns from the original tuning and test data to form `augmented_tune` and `augmented_test`. Fit a decision tree model to this new tuning data (max depth 5, no maximum number of features).

In [21]:
```
# Convert ensemble tune/test to dataframes to concatenate
ensemble_tune_df = pd.DataFrame(np.vstack(ensemble_tune).T, columns = models)
ensemble_test_df = pd.DataFrame(np.vstack(ensemble_test).T, columns = models)

# Concatenate x_tune/x_test with ensemble_tune/test
augmented_tune = pd.concat([x_tune, ensemble_tune_df], axis=1, join_axes=[x_tune.
augmented_test = pd.concat([x_test, ensemble_test_df], axis=1, join_axes=[x_test.

augmentedModel_dt = DecisionTreeClassifier(max_depth=5).fit(augmented_tune, y_tun
```

5.5: How well does the meta-tree do on the test set? Why does training a decision tree on the combination of original data and model predictions perform so well?

In [22]:
```
print('Augmented Decision Meta-Tree Classifier:')
print('Classification Accuracy on test set: {}%\n'.format(round(augmentedModel_dt
```

```
Augmented Decision Meta-Tree Classifier:
Classification Accuracy on test set: 70.3%
```

The augmented decision meta-tree classifier with the model predictions of the 5 tuned models in addition to the original test set performs the best of any classifier on the test set thus far. I believe the reason for this is because the decision tree can have different decision paths for the observations that each model predicted correctly so in a way it can have nodes that differentiate the best predictions from each of the 5 tuned models in addition to the features in the test set that have predictive value. This is in a way related to how the Adaboost algorithm weights the classifiers with the highest accuracy most heavily in the final model - the augmented dataset allows the decision tree to identify each classifiers best characterstics.

5.6: Suggest one way to improve on the model above

In [23]:
```
augmented_ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth = 5), learnin
                                   n_estimators = 100).fit(augmented_tune, y_tune

print('Augmented AdaBoost Meta-Tree Classifier:')
print('Classification Accuracy on test set: {}%\n'.format(round(augmented_ada.sco
```

```
Augmented AdaBoost Meta-Tree Classifier:
Classification Accuracy on test set: 78.46%
```

One way to improve the performance of the augmented meta-tree classifier is to use the augmented tuning data to train an AdaBoost model. This improve the test set accuracy by levering the strengths of the AdaBoost model and increasing the weight of an observation if the observation has been misclassified which enables each additional decision tree classifier to concentrate on previously misclassified observations so that the overall accuracy is improved. As shown above the accuracy on the test set increases an additional 9% with the augmented Adaboost classifier.

Question 6 (12 pts): Understanding

This question is an overall test of your knowledge of this homework's material. You may need to refer to lecture notes and other material outside this homework to answer these questions.

Question 6

6.1 How do ensembling, boosting, and bagging all relate: what is common to all three, and what is unique to each of them?

6.2 Which technique, boosting or bagging, is better suited to parallelization, where you could have multiple computers working on a problem at the same time?

6.3 What is the impact of having too many trees/iterations in boosting and in bagging? In which instance is it worse to overshoot?

6.4 Suppose you have 10,000 training observations and have selected (non-polynomial) linear regression as your base model. Which technique will help your model more, boosting or bagging? How does your choice (and boosting/bagging in general) tie to overfitting versus underfitting?

Answers:
6.1 How do ensembling, boosting, and bagging all relate: what is common to all three, and what is unique to each of them?

Boosting and bagging are both ensemble techniques, where a set of weak learners are combined to create a strong learner that obtains better performance than a single one. They all generate several training data sets by random sampling but where Bagging creates N random samples from the original training set, Boosting determines weights for the observations in the training data to correctly predict the most difficult cases. Boosting creates additional models that are trained on the data where previous models did not perform well. Bagging equally weights the average of each model to predict the outcome whereas Boosting adds more weight to those with better performance on the training set. Lastly, both boosting and bagging techniques attempt to reduce variance and provide a more stable prediction model but Boosting tries to reduce the bias while Bagging attempts to improve accuracy on the out-of-sample data set.

6.2 Which technique, boosting or bagging, is better suited to parallelization, where you could have