ISYE 7406 Homework #5

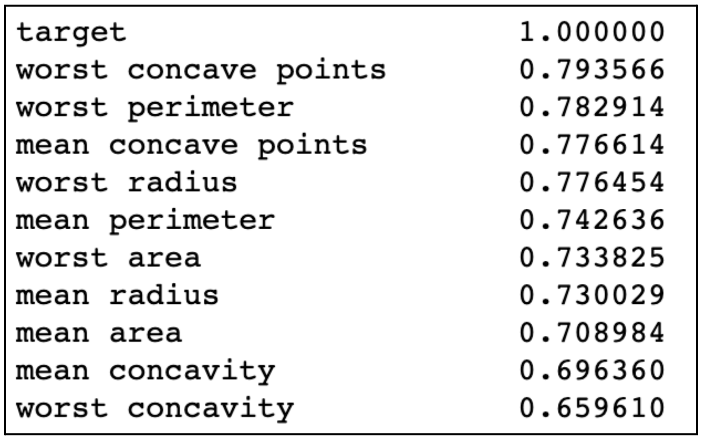
Fall 2022

1. Introduction

In this homework problem, we will be evaluating the performance of two different ensemble methods: random forest and boosting. The dataset is the Wisconsin breast cancer dataset, where we attempt to predict and classify between two classes: malignant and benign. On this dataset, we will also see how the two ensemble techniques perform compared to baseline techniques of Logistic Regression, Support Vector Machine, and Decision Tree Classifier.

2. Data Source

The breast cancer dataset can be downloaded from UCI's ML Repository (https://goo.gl/U2Uwz2). It is also available for download directly from Python's scikit-learn *dataset* module. It contains a total of 569 samples and 30 columns where each row represents a digitized image for breast mass and each feature is computed from the characteristics of the cells within the mass. For each row, a column contains the diagnosis of *malignant* or *benign*, which will serve as the label for our binary classification problem. Before training on our dataset, we can see which features might place the biggest significance in our classification by looking at their correlation with the target column.



```
target                      1.000000
worst concave points        0.793566
worst perimeter             0.782914
mean concave points         0.776614
worst radius                0.776454
mean perimeter              0.742636
worst area                  0.733825
mean radius                 0.730029
mean area                   0.708984
mean concavity              0.696360
worst concavity             0.659610
```

From the above list, we see that the top 10 features with highest correlation to the target relate primarily to *concavity* (four out of ten) so we expect *concavity* to place a high significant factor in our classifiers.

3. Proposed Methodology

First, we will try three simple baseline methods before moving to ensemble methods: Logistic Regression, Decision Tree Classifier, and Support Vector Machine. Logistic Regression and Support Vector Machines would provide a good baseline for comparison because they are well-suited for binary classification problems since the decision functions are typically a linear function (For SVM, it depends on the kernel). As for Decision Tree, it would be useful to see performance of a single decision tree vs. an ensemble tree method. For our two ensemble methods, we will try Random Forest and Adaboost, where in random forest, we consider randomly splitting the dataset by number of features and number of rows and training a single decision tree for the number of estimators set. For Adaboost, we will train many weak learners (single Decision Trees with a small max depth) iteratively to learn to improve upon the train errors in succession of one another. For our performance score, we will be measuring accuracy on a test dataset after training on a separate train dataset (20%). For parameter search, we will retrieve the best parameters off of cross validation on the train dataset.

4. Analysis and Results

For all of our methods, we gather the accuracy score on a test dataset:

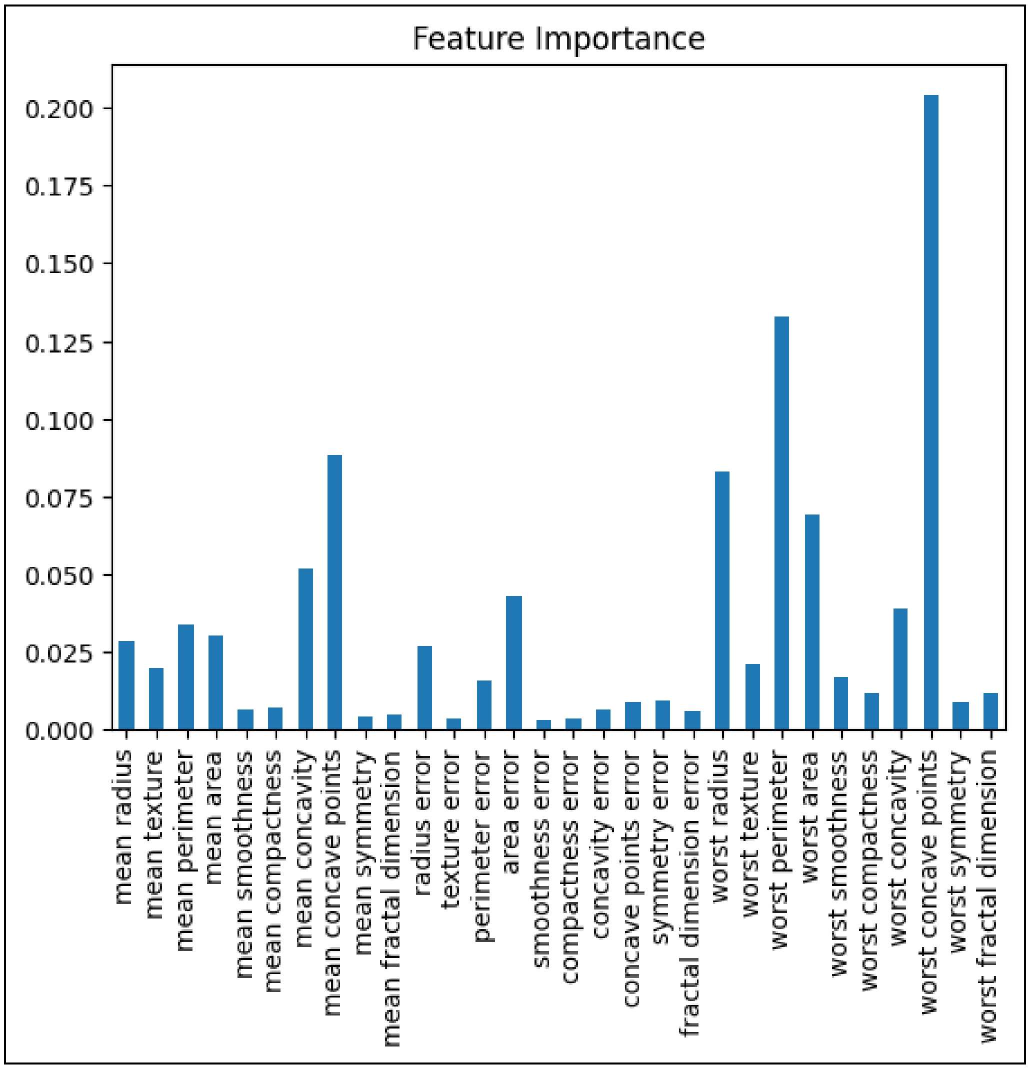
Method	Train Accuracy	Test Accuracy
SVC	0.91%	0.93%
Logistic Regression	0.956%	0.947%
Decision Tree	1.00%	0.912%
Random Forest	1.00%	0.965%
Adaboost	1.00%	0.956%

As we can see, all methods have very high accuracy on the test dataset given the simplistic nature of our dataset, but our ensemble methods have a slightly higher accuracy than the rest. We also notice that the Tree-based classifiers have 100% train accuracy since they typically overfit to the training dataset so ensemble methods are preferred when using tree-based classifiers. They typically reduce the variance to counter the high bias found in tree classifiers.

Next, we find the best parameters in our ensemble methods by performing a grid search and cross validation (*GridSearchCV* in Python). For Random Forest, we evaluated combinations of *entropy* vs *gini* criterion, max depth of trees, how max number of features for trees was computed, minimum sample split, and total number of trees. For Adaboost, we evaluated combinations of the total number of trees and learning rate of the classifier. From here, we find our best parameters and their scores:

Method	Best Parameters	Best CV Score	Test Accuracy
Random Forest	Entropy criterion, Max depth of 6, Max Features (sqrt), Min sample split of 2, 50 total trees	0.960%	**0.974%**
Adaboost	100 total trees, 0.1 learning rate	0.971%	**0.974%**

We find that our two ensemble methods had the same test accuracy score and also outperformed the other simple baseline methods. Even though the improvement was slight because the accuracy of all classifiers was already over 90%, the ensemble methods prove to provide higher accuracy and learn the trends of the data better. To prove this, we can also look at the ranking of feature importance in the random forest classifier with best parameters. In the figure below, we do find that the top first column and top third column are related to *concavity* as we initially predicted in our data analysis before training.



5. Conclusion

In our experiment, we have concluded that ensemble methods have proven to provide higher accuracy on the test dataset in comparison to our three simple baseline methods. However, the performance is tied only to this dataset and could change significantly depending on another dataset used. Since the simple baseline scores were already above 90% accuracy, it is more difficult to judge how well ensemble methods could potentially improve performance. We could also measure by precision, recall, or F1-score given that our dataset is slightly imbalanced. We could also add to our experiment parameter-tuning for the simple baseline methods to see their highest potential performance on the test dataset.

Appendix: Python Code

```
# Import libraries
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.model_selection import cross_val_score, GridSearchCV

# Load dataset
data = load_breast_cancer(as_frame=True)

#Check correlation with target
data.frame.corr()['target'].apply(abs).sort_values(ascending=False)

#Split into train and test
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, 
random_state=0, test_size=0.2)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

clf = SVC(random_state=0)
scores = cross_val_score(clf, data.data, data.target, cv=5)
print("Cross Validation score: {:.3f}".format(scores.mean()))
clf.fit(X_train, y_train)
print("Accuracy on train data: {:.3f}".format(clf.score(X_train, y_train)))
print("Accuracy on test data: {:.3f}".format(clf.score(X_test, y_test)))

clf = LogisticRegression(random_state=0)
scores = cross_val_score(clf, data.data, data.target, cv=5)
print("Cross Validation score: {:.3f}".format(scores.mean()))
clf.fit(X_train, y_train)
print("Accuracy on train data: {:.3f}".format(clf.score(X_train, y_train)))
print("Accuracy on test data: {:.3f}".format(clf.score(X_test, y_test)))

clf = DecisionTreeClassifier(random_state=0)
scores = cross_val_score(clf, data.data, data.target, cv=5)
print("Cross Validation score: {:.3f}".format(scores.mean()))
clf.fit(X_train, y_train)
print("Accuracy on train data: {:.3f}".format(clf.score(X_train, y_train)))
print("Accuracy on test data: {:.3f}".format(clf.score(X_test, y_test)))

clf = RandomForestClassifier(n_estimators=100, random_state=0)
scores = cross_val_score(clf, data.data, data.target, cv=5)
scores.mean()
print("Cross Validation score: {:.3f}".format(scores.mean()))
clf.fit(X_train, y_train)
print("Accuracy on train data: {:.3f}".format(clf.score(X_train, y_train)))
print("Accuracy on test data: {:.3f}".format(clf.score(X_test, y_test)))

clf = AdaBoostClassifier(n_estimators=100)
scores = cross_val_score(clf, data.data, data.target, cv=5)
```

```
scores.mean()
print("Cross Validation score: {:.3f}".format(scores.mean()))
clf.fit(X_train, y_train)
print("Accuracy on train data: {:.3f}".format(clf.score(X_train, y_train)))
print("Accuracy on test data: {:.3f}".format(clf.score(X_test, y_test)))

# Parameter Search for Random Forest
clf = RandomForestClassifier(random_state=0)
param_grid = {
    'n_estimators': [25, 50, 75, 100],
    'max_features': ['sqrt', 'log2', None],
    'max_depth' : [4,5,6,7],
    'criterion' :['gini', 'entropy', 'log_loss'],
    'min_samples_split': [2,4,6]
}
CV_rfc = GridSearchCV(estimator=clf, param_grid=param_grid, cv= 5)
CV_rfc.fit(X_train, y_train)

CV_rfc.best_params_
CV_rfc.best_score_
CV_rfc.score(X_test, y_test)
CV_rfc.best_estimator_.feature_importances_

#Plot importances
rf_importances = pd.Series(CV_rfc.best_estimator_.feature_importances_,
index=data.frame.columns[:30])

fig, ax = plt.subplots()
rf_importances.plot.bar()
ax.set_title("Feature Importance")

#Parameter Search for adaboost
clf = AdaBoostClassifier(random_state=0)
param_grid = {
    'n_estimators': [10, 25, 50, 75, 100],
    'learning_rate': [0.01, 0.05, 0.1, 0.5, 1]
}
CV_adb = GridSearchCV(estimator=clf, param_grid=param_grid, cv= 5)
CV_adb.fit(X_train, y_train)

CV_adb.best_score_
CV_adb.best_params_
CV_adb.score(X_test, y_test)
```