Lab 5 - Random Forest Classifiers

Kyle Swanson

January 16, 2018

-1 Installation

In order to see some of the visualizations for this lab, you need to install a tool called `graphviz`. Installation instructions can be found here: `https://graphviz.gitlab.io/download/`. If you're having trouble installing graphviz, let me know and I can try to help.

0 Introduction

The goal of today's lab is two-fold:

1. Learn how to train and interpret a random forest classifier.
2. Build experience working with scikit-learn classifiers.

0.1 Random Forest Classifiers

In lecture we learned about a type of ensemble machine learning model called a random forest classifier. As a reminder, a random forest classifier trains multiple decision trees and uses the majority vote of the decision trees to make a prediction. Each decision tree is trained on a subset of the data and builds a tree of rules based on which rules best split the data. Rules are constructed by choosing boundaries for each of the different features (ex. $x_1 < 2, x_1 < 5,\ x_1 < 8,\ x_2 < -1,\ x_2 < 5, \dots$).

0.2 scikit-learn

In the labs so far, I have written most of the scikit-learn code so that you could focus on understanding how the machine learning models work. However, knowing how to write code for scikit-learn models is a useful skill, so today I'm going to leave most of the coding up to you.

If you take a look at `main.py`, you'll see it's pretty empty. I've imported all the functions you're going to need, but the rest is up to you!

0.3 Data

In today's lab we're going to be working with the UCI ML Breast Cancer Wisconsin (Diagnostic) data set, which is available as part of the scikit-learn package (`link`). This data set consists of 569 breast tumor samples, of which 212 are malignant and 357 are benign. For each tumor sample, we are given the following features:

```
area error
compactness error
concave points error
concavity error
fractal dimension error
mean area
mean compactness
mean concave points
mean concavity
mean fractal dimension
mean perimeter
mean radius
mean smoothness
mean symmetry
mean texture
perimeter error
radius error
smoothness error
symmetry error
texture error
worst area
worst compactness
worst concave points
worst concavity
worst fractal dimension
worst perimeter
worst radius
worst smoothness
worst symmetry
worst texture
```

The goal is to build a classifier which can use these features to accurately predict whether the tumor is malignant or benign.

1 Data Loading

In Part 1 of `main.py`, we want to load the data set and split it into a train set and a test set.

First, use the function `load_breast_cancer` to load the data. This will load a Python dictionary containing all of the cancer data.

Next, extract the features and the labels from the data dictionary you just loaded. Use the key `'data'` to extract the features and use the key `'target'` to extract the labels.

Finally, use the scikit-learn function `train_test_split` to split the features and labels into a train set of features and labels and a test set of features and labels. When calling `train_test_split`, you should use `test_size=0.2` and `random_state=42`. Using a test size of 0.2 will split the data into 80% training data and 20% testing data, and using the random state will ensure that your results are reproducible. The documentation for `train_test_split` is available here: `http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

2 Decision Tree Classifier

Before building a random forest, which contains multiple decision trees, we're going to start by building and interpreting just a single decision tree.

2.1 Building and Training a Decision Tree Classifier

In Part 2.1 of `main.py`, you will build and train a decision tree classifier. This is made very easy by scikit-learn.

First, construct a `DecisionTreeClassifier`. When doing so, use `random_state=42` so that your results are reproducible.

Next, call the `fit` method of the classifier you just constructed and pass it the training features and labels from Part 1. This will train the decision tree classifier.

2.2 Evaluating the Decision Tree Classifier

In Part 2.2 of `main.py`, you will compute the training and testing accuracy of the classifier we just trained.

First, call the `predict` method of your classifier and pass it the training features. This will produce predictions for the labels of the training data.

Next, use scikit-learn's `accuracy_score` function to compute the accuracy of the training predictions. This can be done by passing it the training predictions you just computed along with the correct training labels from Part 1.

Now, print out the training accuracy.

Repeat the above steps for the test set to print out the test accuracy.

Now run `main.py`. You should see that the test accuracy is very high, meaning the decision tree classifier does a very good job of using these features to determine which tumors are malignant and which are benign.

2.3 Interpreting the Decision Tree Classifier

Now that we have a trained and tested classifier, we want to understand how the classifier is making its predictions. One of the advantages of using decision tree (and random forest) classifiers is that they are very easy to interpret − we can simply print out the decision tree(s) to see exactly how the predictions are made.

First, print out the features for the tree, sorted by the importance of each feature. A list with the names of the features is available in the data dictionary by using the key `'feature_names'`, and you can get the importance value of each feature in the decision tree by accessing the `feature_importances_` property of the tree (for example, if your tree is named `tree`, then you can type `tree.feature_importances_`). You can either sort the features yourself, or you can use the `sort_features` functions I wrote in `utils.py` (you can call the function with `utils.sort_features`). Once you've sorted the features by importance, print them out in that order along with their importance.

Next, call the function `utils.display_decision_tree` to display a visual representation of the tree. This function takes three parameters: a decision tree, a list of feature names (available in the data dictionary with the key `'feature_names'`), and a list of label names (available in the data dictionary with the key `'target_names'`).

Once your implementation is complete, run `main.py`. You should see the most important features printed to the terminal followed by a visual representation of the decision tree. Notice how the most importance features generally end up closer to the top of the decision tree. This makes sense since we want to split our data based on the most important features early on when making a decision.

3 Random Forest Classifier

Now we will perform the same steps but for a random forest classifier rather than for a decision tree classifier.

3.1 Building and Training a Random Forest Classifier

In Part 3.1 in `main.py`, build a `RandomForestClassifier` with `random_state=42`.

Call `fit` on your classifier with the training data and labels.

3.2 Evaluating the Random Forest Classifier

Just as with the decision tree classifier, here you need to use `predict` and then `accuracy_score` to compute the training and testing accuracy. Print both the training and testing accuracy.

If you get the same results as me, you should see that the random forest classifier has a slightly lower training accuracy but a higher testing accuracy. This is because using multiple decision trees prevents overfitting and improves

the generalizability of the model, meaning the model may not do as well on the training data but it does better on data which it hasn't seen before (the test data), which is what we really care about. This therefore makes it a better model than the decision tree classifier.

3.3 Interpreting the Random Forest Classifier

First, loop through the decision trees in your random forest classifier. You can get the decision trees by accessing the `estimators_` property of your random forest classifier (for example, if your classifier is named `clf`, then you can get the trees with `clf.estimators_`).

For each decision tree:

1. First print out the most important features of the tree in sorted order. You may use `utils.sort_features` or you may sort the features yourself.
2. Then use `utils.display_decision_tree` to display a visual representation of the tree.

Finally, print out the most important features of the entire random forest classifier.

Once your implementation is complete, run `main.py`. By default there are 10 trees in the random forest classifier, so you should see 10 sets of features printed to the terminal followed by 10 displays of decision trees. Note how the randomness of the model makes each decision tree different, thereby making the random forest to a more powerful classifier than any single decision tree. Finally, you should see the most important features of the entire random forest model printed to the terminal.