**Iris Random Forest Classification**

**Overview:**

In this lab, you will learn how to classify the famous iris data set with the random forest algorithm.

Fisher’s [iris data set](https://en.wikipedia.org/wiki/Iris_flower_data_set) is one of most famous data sets used in pattern recognition literature. It contains three classes (or ‘target labels’) with 50 instances of each class. Each class represents one of the species of the iris plant (Iris Setosa, Iris Versicolour, and Iris Virginica). Each instance is defined by 4 attributes: sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm.

First, you will do some preliminary exploration of the data set, then the data will be preprocessed (i.e., prepared for analysis), the model will be trained, and, finally, we will use our model to make predictions about the species of unknown iris plants.

**Objectives:**

* To become familiarized with data structures and data transformations used in a simple classification problem
* List specific data structures learned in this lab
* To gain experience classifying data with the random forest algorithm
* To become more comfortable creating data visualizations

**What you’ll need:**

* Jupyter Notebooks
* Python 3
* Basic programming knowledge

**![Diagram

Description automatically generated]()**

**Introduction**

First of all, what is the random forest algorithm? The random forest algorithm is considered to be an ‘ensemble’ method because it combines multiple decision trees and then the different predictions are aggregated to identify the most popular result among different trees, thus improving accuracy. Ensemble methods are often employed to help reduce variance when datasets contain a lot of noise.

Like all ML algorithms, the RF algorithm comes with its own set of pros and cons. The RF algorithm helps reduce the risk of overfitting (a common problem for simple decision trees) by using a robust number of trees and averaging out the best result. It’s also a very flexible method that allows for easy evaluation for feature importance. Unfortunately, with larger data sets it can be a slow process to make all the necessary computations, but this problem can be mitigated by implementing parallel solutions to optimize model training. RF results are also a bit more complex than a simple decision tree, which comes with the drawback of decreased interpretability.

For more information on implementing parallelization when working with pandas, see this [link](https://modin.readthedocs.io/en/latest/using_modin.html) to learn about using Modin for distributing data and computation to accelerate workflows involving pandas.

**Part 1: Data Preprocessing**

*Alright, let’s get started.*

We’re going to be working in a Python 3 environment with Jupyter Notebooks.

The first thing you’re going to need to do is make sure you have all of your libraries installed:

The libraries that will be required are:

* *NumPy* – a library adding support for large, multi-dimensional arrays and matrices, along with high-level mathematical functions to operate on said arrays.
  + [NumPy documentation](https://numpy.org/doc/stable/)
* *Pandas* – a library for data analysis, cleaning, and pre-processing.
  + [Pandas documentation](https://pandas.pydata.org/docs/)
* *Scikit-learn* – a machine learning library that features various classification, regression, and clustering algorithms, including random forests, the algorithm we will be using in this lab.
  + [Scikit-learn documentation](https://scikit-learn.org/stable/tutorial/index.html)
* *Matplotlib* – a library for creating static, animated, and interactive visualizations in Python.
  + [Matplotlib documentation](https://matplotlib.org/)
* *Seaborn* – another data visualization library that provides a high-level interface for creating aesthetic and informative statistical graphics.
  + [Seaborn documentation](https://seaborn.pydata.org/)
* *Dtreeviz* – a library for decision tree visualization and model interpretation.
  + [Dtreeviz library](https://github.com/parrt/dtreeviz)

If you don’t already have a virtual environment set up with the requisite packages, you can use the ‘%pip install’ command in your first notebook kernel.

# install requisite libraries

%pip install numpy

%pip install pandas

%pip install sklearn

%pip install matplotlib

%pip install seaborn

Now let’s import our libraries:

# Importing required libraries  
import numpy as np  
import pandas as pd  
import sklearn  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import confusion\_matrix  
from sklearn.datasets import load\_iris  
import sklearn.metrics as metrics  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn import tree  
from dtreeviz.trees import dtreeviz # will be used for tree visualization

Now that all of the necessary packages / librarires are imported, we can start exploring the data. The iris data set comes included with the sklearn library, so it can be assigned to a variable simply by calling the load\_iris() function. The data includes 150 4-column arrays containing measurements for each feature, and another 150 1-column arrays with either a 0, 1, or 2 indicating what flower corresponds with each feature array. For more information on the array data structure, follow this [link](https://www.geeksforgeeks.org/introduction-to-arrays/).

For this particular lab, the dataset is small and the computation is relatively low, so using parallel techniques would yield little aside from experience for the learner, but if you’d like to learn more about ways to implement parallelization when working with pandas, see this [link](https://www.ibm.com/cloud/learn/random-forest) to learn more about Modin, an early state DataFrame library that wraps Pandas and transparently distributes data and computation to accelerate workflows that stand to benefit from distributed computing.

# Loading datasets  
iris = load\_iris()  
  
# visualize the data  
df = pd.DataFrame(iris.data, columns=iris.feature\_names)  
df['species'] = np.array([iris.target\_names[i] for i in iris.target])  
sns.pairplot(df, hue='species')

A picture containing diagram

Description automatically generated

We can see that setosa’s features are fairly dissimilar from the other two flowers, but versicolor and virginica appear to be a little less distinct.

*What do you notice about the data set?*

*What data structure(s) have been used so far? Why?*

Next, we will convert our data into a Pandas DataFrame:

# Convert to pandas dataframe  
iris\_data = pd.DataFrame({  
 'sepal length':iris.data[:,0],  
 'sepal width':iris.data[:,1],  
 'petal length':iris.data[:,2],  
 'petal width':iris.data[:,3],  
 'species':iris.target  
})  
iris\_data.head()

*Output:*

Table

Description automatically generated

Now that the data has been converted to a Pandas DataFrame, we will establish the independent (X) and dependent (Y) variables. Pandas DataFrames are mutable, allow for labeled axes, and are relatively simple to perform mathematical operations and transformations on, this makes them ideal for tackling this sort of classification problem.

## Some of the advantages of using Pandas for data preprocessing ([source](https://pandas.pydata.org/about/)):

* A fast and efficient **DataFrame** object for data manipulation with integrated indexing;
* Tools for **reading and writing data** between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
* Intelligent **data alignment** and integrated handling of **missing data**: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form;
* Flexible **reshaping** and pivoting of data sets;
* Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets;
* Columns can be inserted and deleted from data structures for **size mutability**;
* Aggregating or transforming data with a powerful **group by** engine allowing split-apply-combine operations on data sets;
* High performance **merging and joining** of data sets;
* **Hierarchical axis indexing** provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure;
* **Time series**-functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
* Highly **optimized for performance**, with critical code paths written in [Python](http://www.cython.org/) or C.
* Python with pandas is in use in a wide variety of **academic and commercial** domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

# setting independent (X) and dependent (Y) variables  
X = iris\_data[['sepal length', 'sepal width', 'petal length', 'petal width']] # Features  
Y = iris\_data['species'] # Labels  
  
# printing feature data  
print('\nFeature data: \n')  
print(X[0:5])  
print('--------------------------------------------------------')  
# printing dependent variable values (0 = setosa, 1 = versicolor, 3 = virginica)  
print('\nDependent variable values:\n')  
print(Y)

Here’s what our data looks like now:

Table

Description automatically generated

Notice we have a feature matrix comprised of the 4 different measurements and then we have a target vector that represents the dependent variable. 150 rows with either a 0, 1, or 2, depending on the flower. You may have noticed that a vector is simply a one-dimensional array, and tensors are just a generalization of n-dimensional arrays.

A picture containing text, crossword puzzle

Description automatically generated

Next, we’ll split the data into training and test sets, allocating 30% of the data for testing the model.

# splitting into train and test sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size = 0.3, random\_state = 100)

**Part 2: Model Construction**

With all of the data preprocessing complete, we will define the random forest classifier, and then fit the training data to the model. Be sure to set a random state value to ensure reproducibility. With the model trained, we can make predictions for our test set to check our model’s accuracy.

# defining random forest classifier  
clfr = RandomForestClassifier(random\_state = 100)  
clfr.fit(X\_train, y\_train)  
  
# making prediction  
Y\_pred = clfr.predict(X\_test)

Now that our model is trained and we’ve tested it against our test set, we can take a look at how it performed. We’ll generate an accuracy score that tells us what percentage of our predictions were correct, and we’ll visualize the prediction distribution with a confusion matrix.

# checking model accuracy  
print("Accuracy:", metrics.accuracy\_score(y\_test, Y\_pred))  
cm = pd.DataFrame(confusion\_matrix(y\_test, Y\_pred), columns=iris.target\_names, index=iris.target\_names)  
sns.heatmap(cm, annot=True)

Chart

Description automatically generated

Now, what this was really all about: making predictions. We are going to make species predictions with the dimensions of a new, unknown flower.

# making predictions on new data  
species\_id = clfr.predict([[5.1, 3.5, 1.4, 0.2]])  
iris.target\_names[species\_id]  
print(iris.target\_names[species\_id])



Pretty cool!

Now we’re going to generate a feature importance score to determine which features were the most relevant to the model for making predictions.

# determining feature importance (e.g. model participation)  
feature\_imp = pd.Series(clfr.feature\_importances\_,index=iris.feature\_names).sort\_values(ascending=False)  
print(feature\_imp)  
# Creating a bar plot to visualize feature participation in model  
sns.barplot(x=feature\_imp, y=feature\_imp.index)  
# use '%matplotlib inline' to plot inline in jupyter notebooks  
# Add labels to your graph  
plt.xlabel('Feature Importance Score')  
plt.ylabel('Features')  
plt.title("Visualizing Important Features")  
plt.legend()  
plt.show()

Chart, bar chart

Description automatically generated

*What does the confusion matrix tell us?*

*Is our accuracy rate adequate, or should a different method be used? Why or why not?*

*What could we do to improve the classification rate?*

The Gini coefficient measures the inequality among values of a frequency distribution. For example, levels of income. A Gini coefficient of zero expresses perfect equality, where all values are the same, while a coefficient of 1 indicates all of the values are concentrated in one area. The Gini Impurity is the default criterion for classification when using the RF algorithm per the [sklearn documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html). More information about Gini Impurity can be found [here](https://victorzhou.com/blog/gini-impurity/).

To wrap up the lab, we will use the dtreeviz library to generate a decision tree diagram that visualizes the nodal splits that define the classification parameters.

plt.figure(figsize=(20,20))  
\_ = tree.plot\_tree(clfr.estimators\_[0], feature\_names=X.columns, filled=True)

Diagram

Description automatically generated with medium confidence

*Why would it be useful to visualize our classification model this way?*

*What data structure is this?*

**Links Discussing AI/Python Security**

* <https://www.linkedin.com/pulse/ai-infers-unseen-security-vulnerabilities-boris-paskalev/>
* <https://dev.to/leahfb/python-security-top-5-best-practices-2of3>
* <https://www.genieai.co/blog/using-python-type-checking-to-improve-security>

**Post-lab Questions**

1. What was the classification rate?
2. What is a Gini coefficient? Why does it approach 0 as the tree moves away from the initial node.
3. What are a few advantages of the RF algorithm that make it suitable for this sort of classification problem?
4. What are the data structures used in this lab?
5. Did we train our algorithm with a supervised or unsupervised method?
6. What is one way we could we implement parallel computing when using Pandas?

References

*About pandas*. pandas. (n.d.). https://pandas.pydata.org/about/.

Lewinson, E. (n.d.). *Beautiful decision tree visualizations with dtreeviz*. KDnuggets. https://www.kdnuggets.com/2021/03/beautiful-decision-tree-visualizations-dtreeviz.html.

Nisha. (2020, October 21). *Random Forest Algorithm on Iris Datasets: Machine Learning*. Goeduhub Technologies. https://www.goeduhub.com/11098/random-forest-algorithm-on-iris-datasets-machine-learning.

Singh, S. (2019, May 30). *Random forest algorithm on iris dataset!!!* Kaggle. https://www.kaggle.com/sonalisingh1411/random-forest-algorithm-on-iris-dataset.

*Using Modin*. Using Modin - Modin documentation. (n.d.). https://modin.readthedocs.io/en/latest/using\_modin.html.

*What is Random Forest?* IBM. (n.d.). https://www.ibm.com/cloud/learn/random-forest.