**Find your way out of the Data Forest with Random Forest**

Introduction :

In this blog we will discuss one of the most widely used Ensembling Machine Learning Algorithm called Random Forest. The goal of the blogpost is to get the beginners started with fundamental concepts of a Random Forest and quickly help them to build their first Random Forest model.

**Motive** to create this tutorial is to get you started using the random forest model and some techniques to improve model accuracy. In this article, I've explained the working of random forest and bagging.

Random forest is a tree-based algorithm which involves building several trees (decision trees), then combining their output to improve generalization ability of the model. The method of combining trees is known as an ensemble method.

**Ensemble methods** are supervised learning models which combine the predictions of multiple smaller models to improve predictive power and generalization.

Say, you want to buy a car. But you are uncertain of its quality. You ask 20 people who have previously . 12 of them said " the car is excellent." Since the majority is in favor, you decide to go for it. This is how we use ensemble methods in machine learning too.

The smaller models that combine to make the ensemble model are referred to as **base models**. Ensemble methods often result in considerably higher performance than any of the individual base models could achieve.

## **Two popular families of ensemble methods**

**BAGGING**

Several estimators are built independently on subsets of the data and their predictions are averaged. Typically the combined estimator is usually better than any of the single base estimator.

**Bagging can reduce variance with little to no effect on bias.**

ex: Random Forests

**BOOSTING**

Base estimators are built sequentially. Each subsequent estimator focuses on the weaknesses of the previous estimators. In essence several weak models "team up" to produce a powerful ensemble model. (We will discuss these later this week.)

**Boosting can reduce bias without incurring higher variance.**

ex: Gradient Boosted Trees, AdaBoost

## **Conditions for ensembles to outperform base models**

For an ensemble method to perform better than a base classifier, it must meet these two criteria:

1. **Accuracy:** the combination of base classifiers must outperform random guessing.
2. **Diversity:** base models must not be identical in classification/regression estimates.

## **Bagging**

The ensemble method we will be using today is called **bagging**, which is short for **bootstrap aggregating**.

Bagging builds multiple base models with **resampled training data with replacement.** We train k base classifiers on n different samples of training data. Using random subsets of the data to train base models promotes more differences between the base models.

Random Forests, which "bag" decision trees, can achieve very high classification accuracy.

## **Bagging's magic decrease of model variance**

One of the biggest advantages of Random Forests is that they **decrease variance without increasing bias**. Essentially you can get a better model without having to trade off between bias and variance.

**VARIANCE DECREASE**

Base model estimates are averaged together, so variability of model predictions (across hypothetical samples) is lower.

**NO/LITTLE BIAS INCREASE**

The bias remains the same as the bias of the individual base models. The model is still able to model the "true function" since the base models' complexity is unrestricted (low bias).

Enough of theory now let’s dive into the implementation logistic regression .

We will use implementation provided by the python machine learning framework known as scikit-learn.

**Problem Statement :**

To build a simple Random Forest model for prediction of car quality given other attributes about the car.

**Data details**

|  |
| --- |
| ========================================== 1. Title: Car Evaluation Database  ==========================================  The dataset is available at “<http://archive.ics.uci.edu/ml/datasets/Car+Evaluation>”  2. Sources:  (a) Creator: Marko Bohanec  (b) Donors: Marko Bohanec (marko.bohanec@ijs.si)  Blaz Zupan (blaz.zupan@ijs.si)  (c) Date: June, 1997  3. Past Usage:   The hierarchical decision model, from which this dataset is  derived, was first presented in    M. Bohanec and V. Rajkovic: Knowledge acquisition and explanation for  multi-attribute decision making. In 8th Intl Workshop on Expert  Systems and their Applications, Avignon, France. pages 59-78, 1988.   Within machine-learning, this dataset was used for the evaluation  of HINT (Hierarchy INduction Tool), which was proved to be able to  completely reconstruct the original hierarchical model. This,  together with a comparison with C4.5, is presented in   B. Zupan, M. Bohanec, I. Bratko, J. Demsar: Machine learning by  function decomposition. ICML-97, Nashville, TN. 1997 (to appear)  4. Relevant Information Paragraph:   Car Evaluation Database was derived from a simple hierarchical  decision model originally developed for the demonstration of DEX  (M. Bohanec, V. Rajkovic: Expert system for decision  making. Sistemica 1(1), pp. 145-157, 1990.). The model evaluates  cars according to the following concept structure:   CAR car acceptability  . PRICE overall price  . . buying buying price  . . maint price of the maintenance  . TECH technical characteristics  . . COMFORT comfort  . . . doors number of doors  . . . persons capacity in terms of persons to carry  . . . lug\_boot the size of luggage boot  . . safety estimated safety of the car   Input attributes are printed in lowercase. Besides the target  concept (CAR), the model includes three intermediate concepts:  PRICE, TECH, COMFORT. Every concept is in the original model  related to its lower level descendants by a set of examples (for  these examples sets see<http://www-ai.ijs.si/BlazZupan/car.html).>   The Car Evaluation Database contains examples with the structural  information removed, i.e., directly relates CAR to the six input  attributes: buying, maint, doors, persons, lug\_boot, safety.   Because of known underlying concept structure, this database may be  particularly useful for testing constructive induction and  structure discovery methods.  5. Number of Instances: 1728  (instances completely cover the attribute space)  6. Number of Attributes: 6  7. Attribute Values:   buying v-high, high, med, low  maint v-high, high, med, low  doors 2, 3, 4, 5-more  persons 2, 4, more  lug\_boot small, med, big  safety low, med, high  8. Missing Attribute Values: none  9. Class Distribution (number of instances per class)   class N N[%]  -----------------------------  unacc 1210 (70.023 %)   acc 384 (22.222 %)   good 69 ( 3.993 %)   v-good 65 ( 3.762 %) |

Tools to be used :

Numpy,pandas,scikit-learn

**Python Implementation with code :**

**Import necessary libraries**

Import the necessary modules from specific libraries.

|  |
| --- |
| import os  import numpy as np  import pandas as pd  import numpy as np, pandas as pd  import matplotlib.pyplot as plt  from sklearn import metrics, model\_selection, preprocessing  from sklearn.ensemble import RandomForestClassifier |

**Load the data set**

Use pandas module to read the bike data from the file system. Check few records of the dataset.

|  |
| --- |
| data = pd.read\_csv('data/car\_quality/car.data',names=['buying','maint','doors','persons','lug\_boot','safety','class'])  data.head()  buying maint doors persons lug\_boot safety class  0 vhigh vhigh 2 2 small low unacc  1 vhigh vhigh 2 2 small med unacc  2 vhigh vhigh 2 2 small high unacc  3 vhigh vhigh 2 2 med low unacc  4 vhigh vhigh 2 2 med med unacc |

**Check few information about the data set**

|  |
| --- |
| data.info()  <class 'pandas.core.frame.DataFrame'> RangeIndex: 1728 entries, 0 to 1727 Data columns (total 7 columns): buying 1728 non-null object maint 1728 non-null object doors 1728 non-null object persons 1728 non-null object lug\_boot 1728 non-null object safety 1728 non-null object class 1728 non-null object dtypes: object(7) memory usage: 94.6+ KB |

The train data set has 1728 rows and 7 columns.

There are no missing values in the dataset

**Identify the target variable**

|  |
| --- |
| data['class'],class\_names = pd.factorize(data['class']) |

The target variable is marked as class in the dataframe. The values are present in string format. However the algorithm requires the variables to be coded into its equivalent integer codes. We can convert the string categorical values into a integer code using factorize method of the pandas library.

Let’s check the encoded values now.

|  |
| --- |
| print(class\_names)  print(data['class'].unique())  Index([u'unacc', u'acc', u'vgood', u'good'], dtype='object') [0 1 2 3] |

As we can see the values has been encoded into 4 different numeric labels.

**Identify the predictor variables and encode any string variables to equivalent integer codes**

|  |
| --- |
| data['buying'],\_ = pd.factorize(data['buying'])  data['maint'],\_ = pd.factorize(data['maint'])  data['doors'],\_ = pd.factorize(data['doors'])  data['persons'],\_ = pd.factorize(data['persons'])  data['lug\_boot'],\_ = pd.factorize(data['lug\_boot'])  data['safety'],\_ = pd.factorize(data['safety'])  data.head()  buying maint doors persons lug\_boot safety class  0 0 0 0 0 0 0 0  1 0 0 0 0 0 1 0  2 0 0 0 0 0 2 0  3 0 0 0 0 1 0 0  4 0 0 0 0 1 1 0 |

Check the data types now :

|  |
| --- |
| data.info()  <class 'pandas.core.frame.DataFrame'> RangeIndex: 1728 entries, 0 to 1727 Data columns (total 7 columns): buying 1728 non-null int64 maint 1728 non-null int64 doors 1728 non-null int64 persons 1728 non-null int64 lug\_boot 1728 non-null int64 safety 1728 non-null int64 class 1728 non-null int64 dtypes: int64(7) memory usage: 94.6 KB |

Everything is now converted in integer form.

**Select the predictor feature and select the target variable**

|  |
| --- |
| X = data.iloc[:,:-1]  y = data.iloc[:,-1] |

**Train test split :**

|  |
| --- |
| # split data randomly into 70% training and 30% test  X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, test\_size=0.3, random\_state=0) |

**Training / model fitting**

|  |
| --- |
| # we can achieve the above two tasks using the following codes  # Bagging: using all features  model = RandomForestClassifier(random\_state=1)  model.fit(X\_train, y\_train) |

**Model parameters study :**

|  |
| --- |
| # use the model to make predictions with the test data  y\_pred = model.predict(X\_test)  # how did our model perform?  count\_misclassified = (y\_test != y\_pred).sum()  print('Misclassified samples: {}'.format(count\_misclassified))  accuracy = metrics.accuracy\_score(y\_test, y\_pred)  print('Accuracy: {:.2f}'.format(accuracy))  Misclassified samples: 19 Accuracy: 0.96 |

As you can see the algorithm was able to achieve classification accuracy of 96% on the held out set. Only 19 samples were misclassified.

Algo Advantages :

* Random Forest can be used to solve both kinds of problems ;regression and classification.
* Is capable of handling high dimensional datasets
* Can be used to extract out relevant features
* Handles missing data effectively internally

Algo Disadvantages :

* Difficult to interpret because of various trees involved internally
* It tends to return erratic predictions for observations out of range of training data. For example, the training data contains two variable x and y. The range of x variable is 30 to 70. If the test data has x = 200, random forest would give an unreliable prediction.
* It can take longer time than expectation to grow a large number of trees