Predicting car quality with the help of Neighbors

**Introduction** :

The goal of the blogpost is to get the beginners started with fundamental concepts of the K Nearest Neighbour Classification Algorithm popularly known by the name KNN classifiers. We will mainly focus on learning to build your first KNN model. The data cleaning and preprocessing parts would be covered in detail in an upcoming post.

Classification Machine Learning is a technique of learning where a particular instance is mapped against one of the many labels. The labels are prespecified to train your model . The machine learns the pattern from the data in such a way that the learned representation successfully maps the original dimension to the suggested label/class without any more intervention from a human expert.

## **How does k-Nearest Neighbors Work**

The kNN algorithm is belongs to the family of instance-based, competitive learning and lazy learning algorithms.

Instance-based algorithms are those algorithms that model the problem using data instances (or rows) in order to make predictive decisions. The kNN algorithm is an extreme form of instance-based methods because all training observations are retained as part of the model.

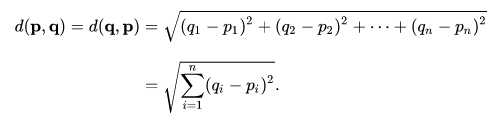
It is a competitive learning algorithm, because it internally uses competition between model elements (data instances) in order to make a predictive decision. The objective similarity measure between data instances causes each data instance to compete to “win” or be most similar to a given unseen data instance and contribute to a prediction.

Lazy learning refers to the fact that the algorithm does not build a model until the time that a prediction is required. It is lazy because it only does work at the last second. This has the benefit of only including data relevant to the unseen data, called a localized model. A disadvantage is that it can be computationally expensive to repeat the same or similar searches over larger training datasets.

Finally, kNN is powerful because it does not assume anything about the data, other than a distance measure can be calculated consistently between any two instances. As such, it is called non-parametric or non-linear as it does not assume a functional form.

## **Euclidean distance**

Euclidean distance is the most commonly used distance measure. Euclidean distance also called as simply distance. The usage of Euclidean distance measure is highly recommended when data is dense or continuous. Euclidean distance is the best proximity measure. The Euclidean distance between two points is the length of the path connecting them.The Pythagorean theorem gives this distance between two points. A generalized term for the Euclidean norm is the L2 norm or L2 distance.



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 KNN classification model for predicting the quality of the car given few of other car attributes.

**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.neighbors import KNeighborsClassifier |

**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**

|  |
| --- |
| # train the decision tree  ## Instantiate the model with 5 neighbors.  model = KNeighborsClassifier(n\_neighbors=5)  ## Fit the model on the training data.  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: 32 Accuracy: 0.94 |

As you can see the algorithm was able to achieve classification accuracy of 94% on the held out set. Only 32 samples were misclassified.Since this is a very simplistic data set with distinctly separable classes. But there you have it. That’s how to implement K-Nearest Neighbors with scikit-learn. Load your favorite data set and give it a try!

## **How to decide the value of n-neighbors**

Choosing a large value of K will lead to greater amount of execution time & underfitting. Selecting the small value of K will lead to overfitting. There is no such guaranteed way to find the best value of K.

|  |
| --- |
| from sklearn.metrics import accuracy\_score  for K in range(25):  K\_value = K+1  neigh = KNeighborsClassifier(n\_neighbors = K\_value)  neigh.fit(X\_train, y\_train)  y\_pred = neigh.predict(X\_test)  print("Accuracy is ", accuracy\_score(y\_test,y\_pred)\*100,"% for K-Value:",K\_value)  ('Accuracy is ', 83.62235067437379, '% for K-Value:', 1) ('Accuracy is ', 80.15414258188824, '% for K-Value:', 2) ('Accuracy is ', 89.21001926782274, '% for K-Value:', 3) ('Accuracy is ', 88.82466281310212, '% for K-Value:', 4) ('Accuracy is ', 93.83429672447014, '% for K-Value:', 5) ('Accuracy is ', 92.8709055876686, '% for K-Value:', 6) ('Accuracy is ', 92.8709055876686, '% for K-Value:', 7) ('Accuracy is ', 89.78805394990366, '% for K-Value:', 8) ('Accuracy is ', 90.94412331406551, '% for K-Value:', 9) ('Accuracy is ', 88.82466281310212, '% for K-Value:', 10) ('Accuracy is ', 89.40269749518305, '% for K-Value:', 11) ('Accuracy is ', 88.6319845857418, '% for K-Value:', 12) ('Accuracy is ', 88.82466281310212, '% for K-Value:', 13) ('Accuracy is ', 89.01734104046243, '% for K-Value:', 14) ('Accuracy is ', 89.78805394990366, '% for K-Value:', 15) ('Accuracy is ', 88.6319845857418, '% for K-Value:', 16) ('Accuracy is ', 88.82466281310212, '% for K-Value:', 17) ('Accuracy is ', 88.4393063583815, '% for K-Value:', 18) ('Accuracy is ', 88.6319845857418, '% for K-Value:', 19) ('Accuracy is ', 88.6319845857418, '% for K-Value:', 20) ('Accuracy is ', 88.2466281310212, '% for K-Value:', 21) ('Accuracy is ', 89.01734104046243, '% for K-Value:', 22) ('Accuracy is ', 89.21001926782274, '% for K-Value:', 23) ('Accuracy is ', 89.01734104046243, '% for K-Value:', 24) ('Accuracy is ', 89.59537572254335, '% for K-Value:', 25) |

It shows that we are getting 93.83% accuracy on K = 5. Hence we are considering K =5 for this tutorial.