Report-01: Title KNN algorithm(Final)

CSE-0408 Summer 2021

Safayet tanvir shiddiki
ID: ug02-37-14-016
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
email address: safayettanvirshiddiki99@gmail.com

***Abstract*—This paper introduced for KNN problem**
***Index Terms*——Languages: Python and anaconda jupyter**

I. Introduction

KNN is a lazy learning, non-parametric algorithm. It uses data with several classes to predict the classification of the new sample point. KNN is non-parametric since it doesn't make any assumptions on the data being studied, the model is distributed from the data

II. Proposed Methodology

When we classified a data set including large number of unlabeled data, if only utilize the few training examples available, then we can't obtain a high accuracy classifier with inadequate training examples; if we want to obtain a classifier of high performance, then labeling the unlabeled data is necessary, but labeling vast unlabeled data wastes time and consumes strength. In this paper, we propose a novel method which uses SVM cooperated with KNN for classification based on semi supervised learning theory. The general model is depicted as above (See Figure 2). To begin with, we construct a weaker classifier SVM according to the few training examples available, then using the weaker SVM classifies the remaining large number of unlabeled data in the data set, picking out n examples belonging to each class around the decision boundary by calculating Euclidean distance in the feature space, because the examples located around the boundary are easy to be misclassified, but they are likely to be the support vectors, we call them boundary vectors, so picking out these boundary vectors whose labels are fuzzy labeled by the weaker classifier SVM. Secondly we recognize these boundary vectors as testing set while recognize initial training examples as training set, use KNN method to classify them and recognize the results as the labels for boundary vectors. In the end, we put these boundary vectors and their labels into initial training set to enlarge the number of the training examples then retrain a SVM, iteratively until the number of the training examples is m times of the whole data set. The experimental results on three UCI data sets indicate that the final classifier SVM has significant improvement on accuracy

III. ADVANTAGES AND DISADVANTAGE FOR KNN ADVANTAGES:-

ADVANTAGES:-
1. Simple to implement and intuitive to understand
2. 2.Can learn non-linear decision boundaries when used for classification and regression. Can came up with a highly flexible decision boundary adjusting the value of K
DISADVANTAGE:-
Factors such as k value, distance calculation and choice of appropriate predictors all have significant impact on the model performance.

IV. Conclusion and Future Work

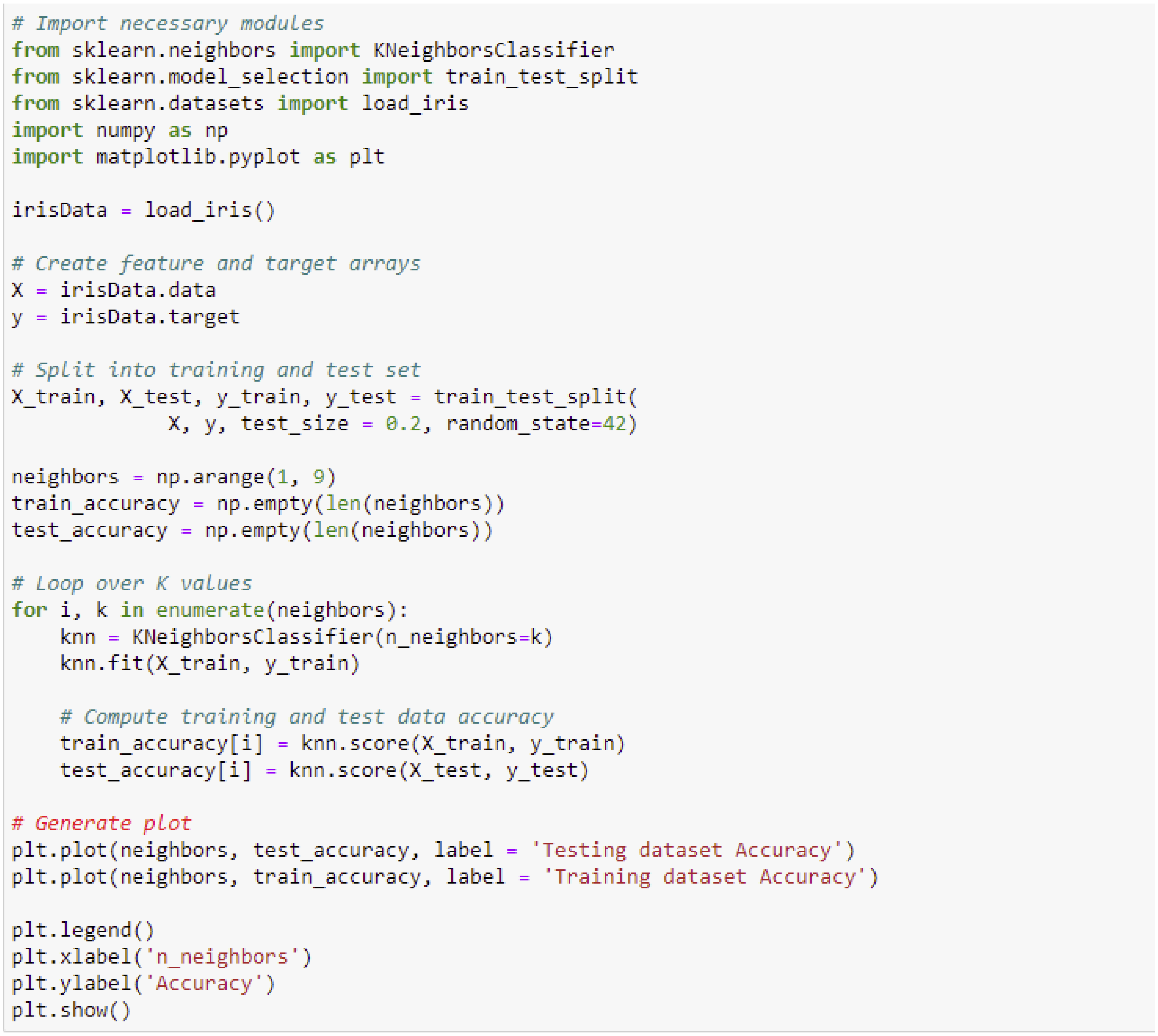
The article introduces some basic ideas underlying the kNN algorithm. The data set should be prepared before running the knn() function in R. After prediction of outcome with kNN algorithm, the diagnostic performance of the model should be checked. Average accuracy is the most widely used statistic to reflect the performance kNN algorithm.

Acknowledgment

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

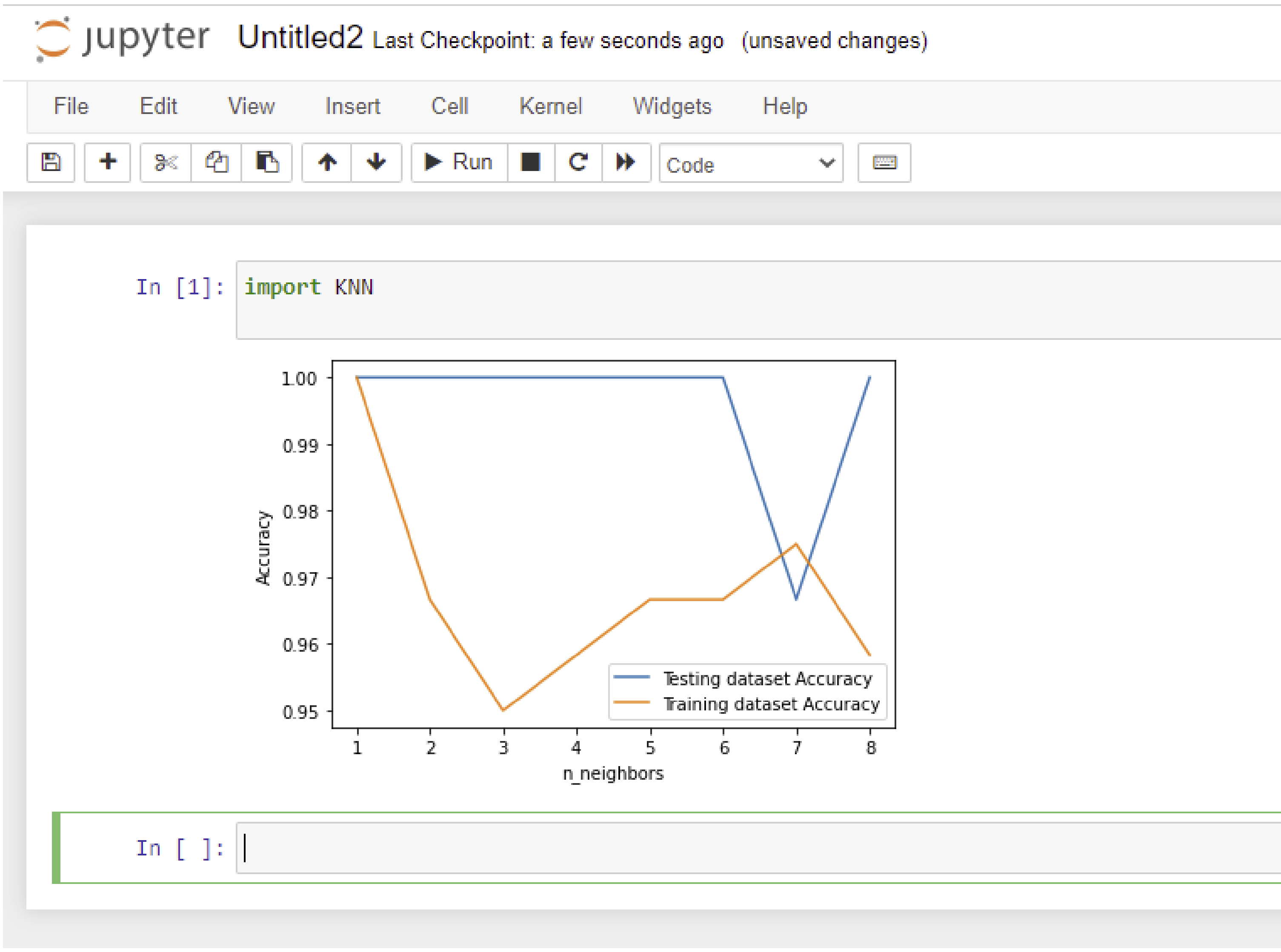
References

[1] [1] Zhou Q, Yang D, Ombrello AK, et al. Early-onset stroke and vasculopathy associated with mutations in ADA2. N Engl J Med. 2014;370:911–20
[2] . [2] Navon Elkan P, Pierce SB, Segel R, et al. Mutant adenosine deaminase 2 in a polyarteritis nodosa vasculopathy. N Engl J Med. 2014;370:921–31.



```python
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt

irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
             X, y, test_size = 0.2, random_state=42)

neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Compute training and test data accuracy
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

Fig. 1. Example of a figure caption.


jupyter
Untitled2 Last Checkpoint: a few seconds ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code

```
In [1]: import KNN
```

```
In [ ]:
```

Fig. 2. Example of a figure caption.

Report-02: Title decision tree algorithm(final)

CSE-0408 Summer 2021

Safayet tanvir shiddiki
ID: ug02-37-14-016
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
email address: safayettanvirshiddiki99@gmail.com

***Abstract*—This paper introduced for Decision Tree problem**

***Index Terms*—Languages: Python and anaconda jupyter**

I. Introduction

In a decision tree, the algorithm starts with a root node of a tree then compares the value of different attributes and follows the next branch until it reaches the end leaf node. Deficiency of adenosine deaminase type 2 (DADA2) is an autosomal recessive systemic autoinflammatory disorder (SAID) described for the first time in 2014 [1, 2]. Both homozygous or compound heterozygous genotypes have been detected [3]. Although one mutation c.139G¿A;p.(Gly47Arg) is frequent, notably in the Georgian population, due to a founder effect, the disease seems to occur ubiquitously; indeed, patients with DADA2 have been identified in several countries [4].

II. Proposed Methodology

Angel Insua, Alberto Monje, Hom-Lay Wang, Marita Inglehart, Patient-Centered Perspectives and Understanding of Peri-Implantitis, Journal of Periodontology.

1.Performance Issue with large data-set: The time required to calculate the distance between the new point and each existing points is huge. Which then degrades the performance of the algorithm.
2.Value of K: It is really crucial to determine what value to assign to k. with different value of K you get different results

III. ADVANTAGES AND DISADVANTAGE FOR DECISION TREE

ADVANTAGES :-
1.When using Decision tree algorithm it is not necessary to normalize the data.
2.Decision tree algorithm implementation can be done without scaling the data as well.
3.When using Decision tree algorithm it is not necessary to impute the missing values.
4.The data pre-processing step for decision trees requires less code and analysis.
5.The data pre-processing step for decision trees requires less time.

DISADVANTAGE:-
1.The mathematical calculation of decision tree mostly require more memory.
2.The mathematical calculation of decision tree mostly require more time.
3.The space and time complexity of decision tree model is relatively higher.
4.Decision tree model training time is relatively more as complexity.

IV. Conclusion and Future Work

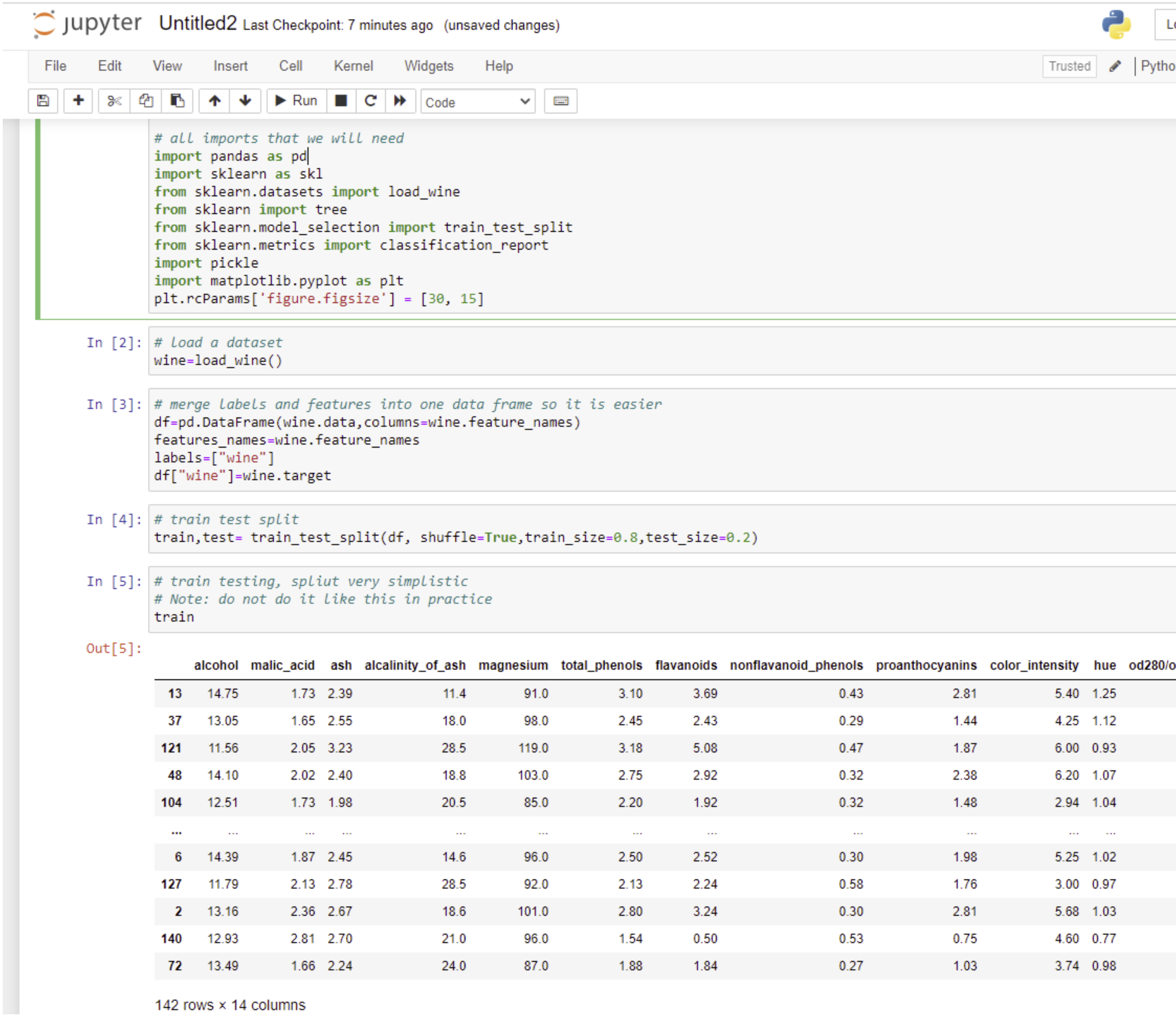
We report a large series of patients referred to us for genetic diagnosis of DADA2. We used information provided by the ordering clinicians to (1) describe the population with suspected DADA2, (2) compare our patients to those previously reported and (3) try to delineate prerequisites for a positive genetic diagnosis. We identified 13 patients carrying recursively inherited mutations in ADA2 that were predicted to be deleterious. Eight patients were compound heterozygous for mutations. Seven mutations were novel (4 missense variants, 2 predicted to affect mRNA splicing and 1 frame shift). Phenotype manifestations included fever, vasculitis and neurological disorders. Prerequisites for quick and low-cost Sanger analysis included one typical cutaneous or neurological sign, one marker of inflammation (fever or elevated CRP level), and recurrent or chronic attacks in adults

Acknowledgment

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] Gonzalez Santiago TM, Zavialov A, Saarela J, et al. Dermatologic features of ADA2 deficiency in cutaneous polyarteritis nodosa. JAMA Dermatol. 2015;151:1230–4. for graphical models. In AAAI, 2007
[2] Short RD, Fukunaga K. The optimal distance measure for nearest neighbor classification. IEEE Transactions on Information Theory 1981;27:622-7. 10.1109/TIT.1981.1056403



```
# all imports that we will need
import pandas as pd
import sklearn as skl
from sklearn.datasets import load_wine
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import pickle
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [30, 15]
```

```
In [2]: # load a dataset
        wine=load_wine()
```

```
In [3]: # merge labels and features into one data frame so it is easier
        df=pd.DataFrame(wine.data,columns=wine.feature_names)
        features_names=wine.feature_names
        labels=["wine"]
        df["wine"]=wine.target
```

```
In [4]: # train test split
        train,test= train_test_split(df, shuffle=True,train_size=0.8,test_size=0.2)
```

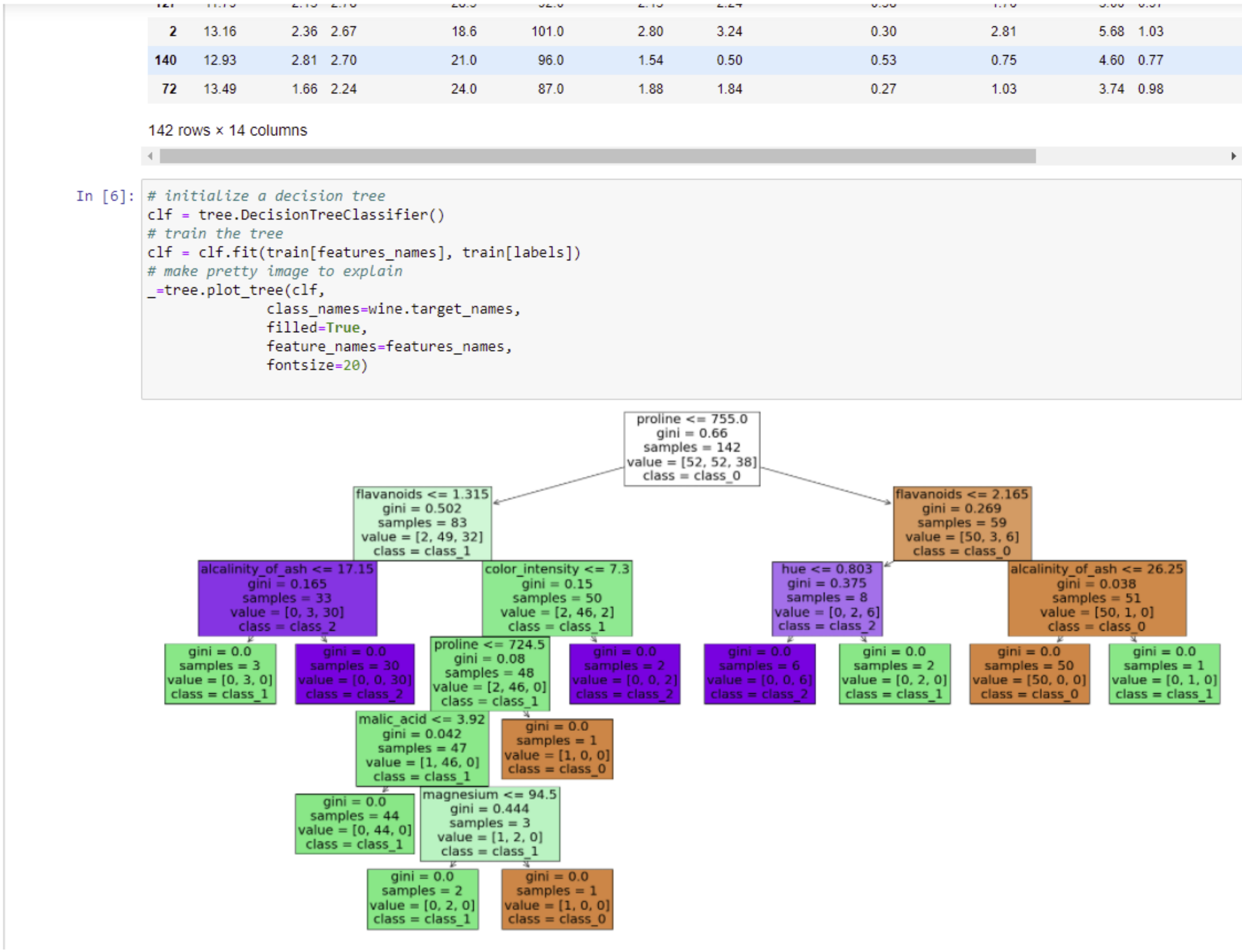
```
In [5]: # train testing, spliut very simplistic
        # Note: do not do it like this in practice
        train
```

Out[5]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/o
13	14.75	1.73	2.39	11.4	91.0	3.10	3.69	0.43	2.81	5.40	1.25	
37	13.05	1.65	2.55	18.0	98.0	2.45	2.43	0.29	1.44	4.25	1.12	
121	11.56	2.05	3.23	28.5	119.0	3.18	5.08	0.47	1.87	6.00	0.93	
48	14.10	2.02	2.40	18.8	103.0	2.75	2.92	0.32	2.38	6.20	1.07	
104	12.51	1.73	1.98	20.5	85.0	2.20	1.92	0.32	1.48	2.94	1.04	
...	...	...	...	...	...	...	...	...	...	...	...	
6	14.39	1.87	2.45	14.6	96.0	2.50	2.52	0.30	1.98	5.25	1.02	
127	11.79	2.13	2.78	28.5	92.0	2.13	2.24	0.58	1.76	3.00	0.97	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	
140	12.93	2.81	2.70	21.0	96.0	1.54	0.50	0.53	0.75	4.60	0.77	
72	13.49	1.66	2.24	24.0	87.0	1.88	1.84	0.27	1.03	3.74	0.98	

142 rows × 14 columns

Fig. 1. Example of a figure caption.

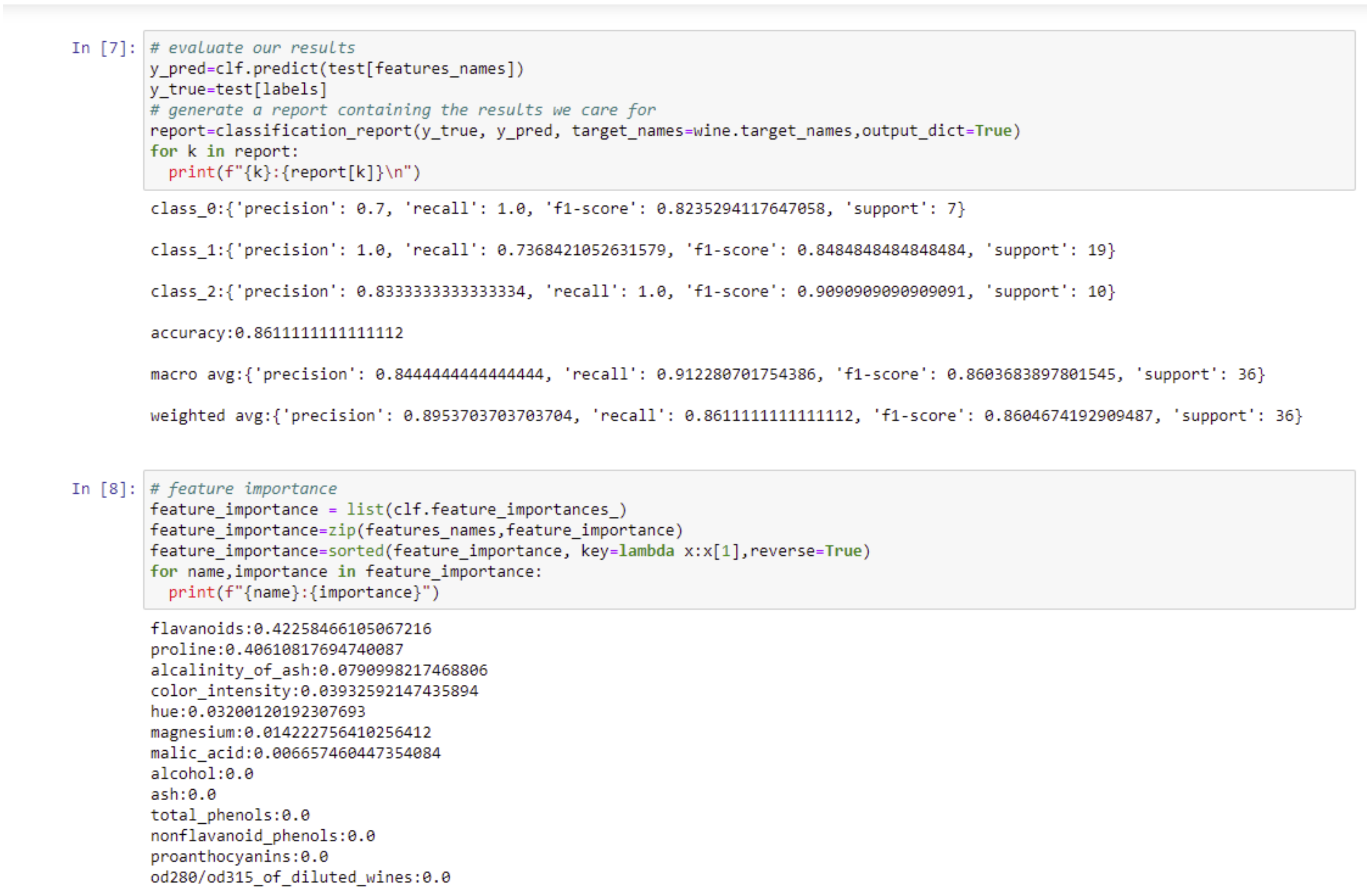


2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03
140	12.93	2.81	2.70	21.0	96.0	1.54	0.50	0.53	0.75	4.60	0.77
72	13.49	1.66	2.24	24.0	87.0	1.88	1.84	0.27	1.03	3.74	0.98

142 rows × 14 columns

```
In [6]: # initialize a decision tree
        clf = tree.DecisionTreeClassifier()
        # train the tree
        clf = clf.fit(train[features_names], train[labels])
        # make pretty image to explain
        _=tree.plot_tree(clf,
                      class_names=wine.target_names,
                      filled=True,
                      feature_names=features_names,
                      fontsize=20)
```

Fig. 2. Example of a figure caption.



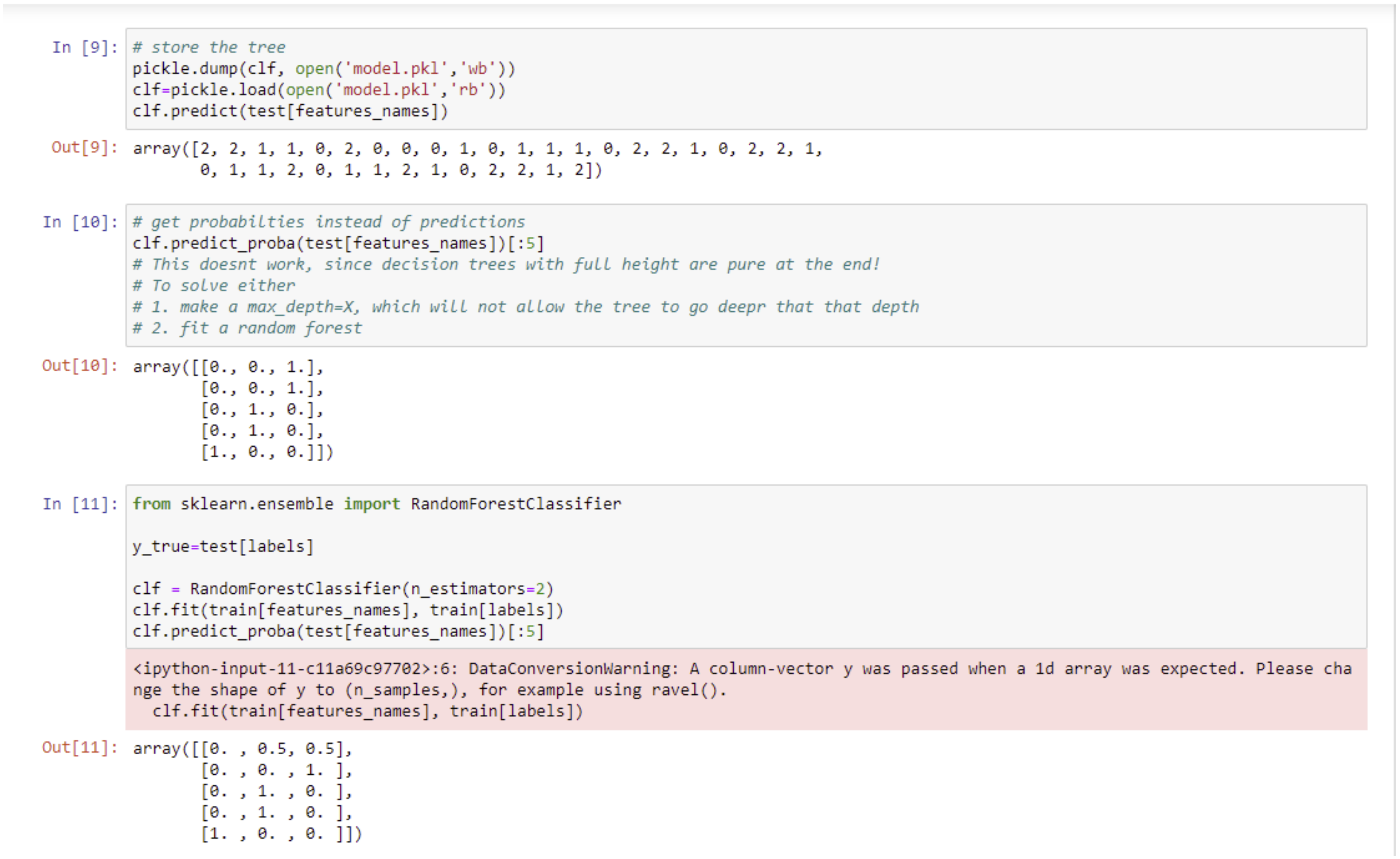
```
In [7]: # evaluate our results
        y_pred=clf.predict(test[features_names])
        y_true=test[labels]
        # generate a report containing the results we care for
        report=classification_report(y_true, y_pred, target_names=wine.target_names,output_dict=True)
        for k in report:
          print(f"{k}:{report[k]}\n")
```

```
class_0:{'precision': 0.7, 'recall': 1.0, 'f1-score': 0.8235294117647058, 'support': 7}

class_1:{'precision': 1.0, 'recall': 0.7368421052631579, 'f1-score': 0.8484848484848484, 'support': 19}

class_2:{'precision': 0.8333333333333334, 'recall': 1.0, 'f1-score': 0.9090909090909091, 'support': 10}

accuracy:0.8611111111111112

macro avg:{'precision': 0.8444444444444444, 'recall': 0.912280701754386, 'f1-score': 0.8603683897801545, 'support': 36}

weighted avg:{'precision': 0.8953703703703704, 'recall': 0.8611111111111112, 'f1-score': 0.8604674192909487, 'support': 36}
```

```
In [8]: # feature importance
        feature_importance = list(clf.feature_importances_)
        feature_importance=zip(features_names,feature_importance)
        feature_importance=sorted(feature_importance, key=lambda x:x[1],reverse=True)
        for name,importance in feature_importance:
          print(f"{name}:{importance}")
```

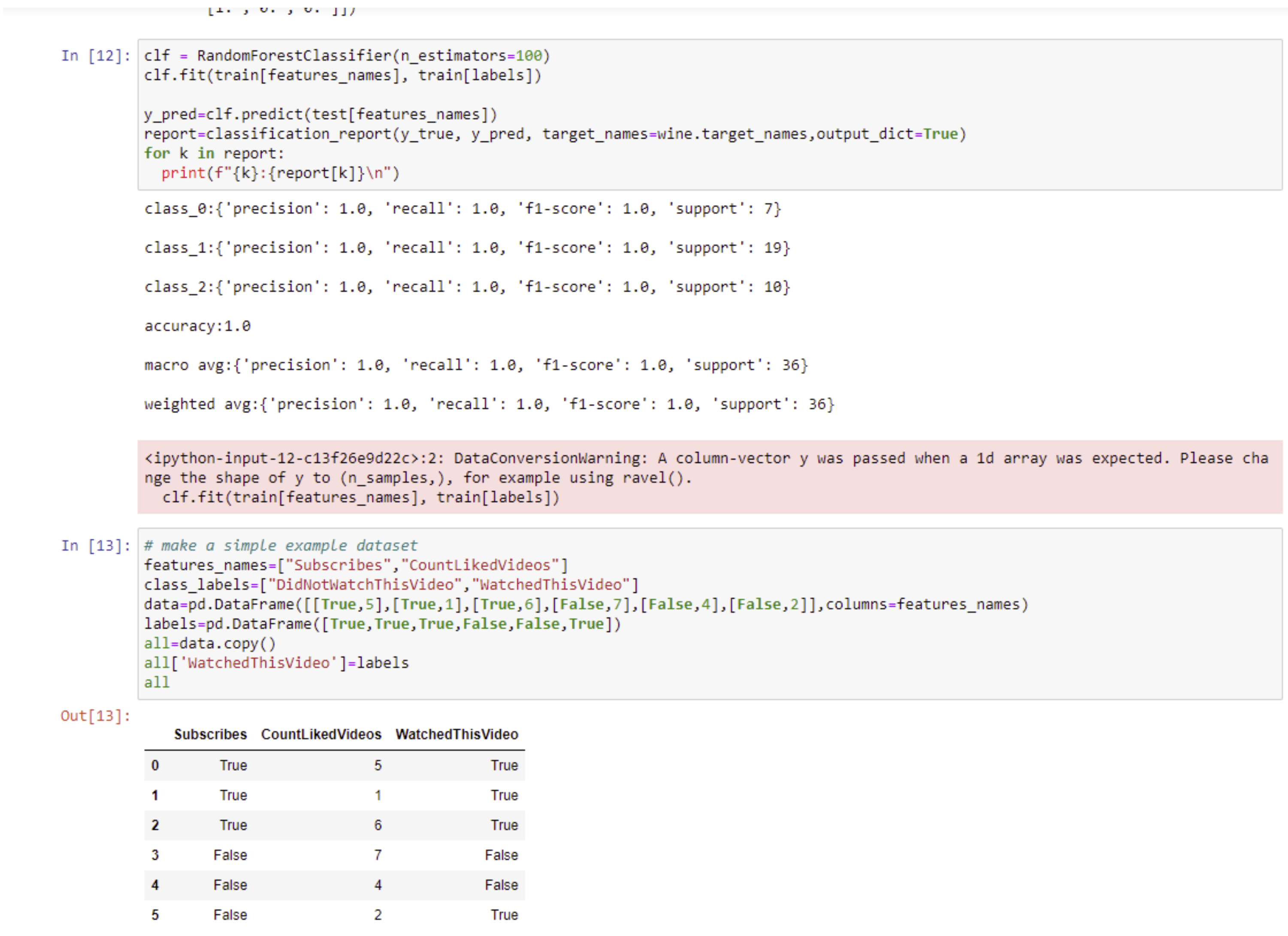
```
flavanoids:0.42258466105067216
proline:0.40610817694740087
alcalinity_of_ash:0.0790998217468806
color_intensity:0.03932592147435894
hue:0.03200120192307693
magnesium:0.014222756410256412
malic_acid:0.006657460447354084
alcohol:0.0
ash:0.0
total_phenols:0.0
nonflavanoid_phenols:0.0
proanthocyanins:0.0
od280/od315_of_diluted_wines:0.0
```

Fig. 3. Example of a figure caption.



```
In [9]:  # store the tree
         pickle.dump(clf, open('model.pkl','wb'))
         clf=pickle.load(open('model.pkl','rb'))
         clf.predict(test[features_names])

Out[9]:  array([2, 2, 1, 1, 0, 2, 0, 0, 0, 1, 0, 1, 1, 1, 0, 2, 2, 1, 0, 2, 2, 1,
                0, 1, 1, 2, 0, 1, 1, 2, 1, 0, 2, 2, 1, 2])

In [10]: # get probabilties instead of predictions
         clf.predict_proba(test[features_names])[:5]
         # This doesnt work, since decision trees with full height are pure at the end!
         # To solve either
         # 1. make a max_depth=X, which will not allow the tree to go deepr that that depth
         # 2. fit a random forest

Out[10]: array([[0., 0., 1.],
                [0., 0., 1.],
                [0., 1., 0.],
                [0., 1., 0.],
                [1., 0., 0.]])

In [11]: from sklearn.ensemble import RandomForestClassifier

         y_true=test[labels]

         clf = RandomForestClassifier(n_estimators=2)
         clf.fit(train[features_names], train[labels])
         clf.predict_proba(test[features_names])[:5]

         <ipython-input-11-c11a69c97702>:6: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
         nge the shape of y to (n_samples,), for example using ravel().
           clf.fit(train[features_names], train[labels])

Out[11]: array([[0. , 0.5, 0.5],
                [0. , 0. , 1. ],
                [0. , 1. , 0. ],
                [0. , 1. , 0. ],
                [1. , 0. , 0. ]])
```

Fig. 4. Example of a figure caption.



```
        [1. , 0. , 0. ]])
```

```
In [12]: clf = RandomForestClassifier(n_estimators=100)
         clf.fit(train[features_names], train[labels])

         y_pred=clf.predict(test[features_names])
         report=classification_report(y_true, y_pred, target_names=wine.target_names,output_dict=True)
         for k in report:
           print(f"{k}:{report[k]}\n")
```

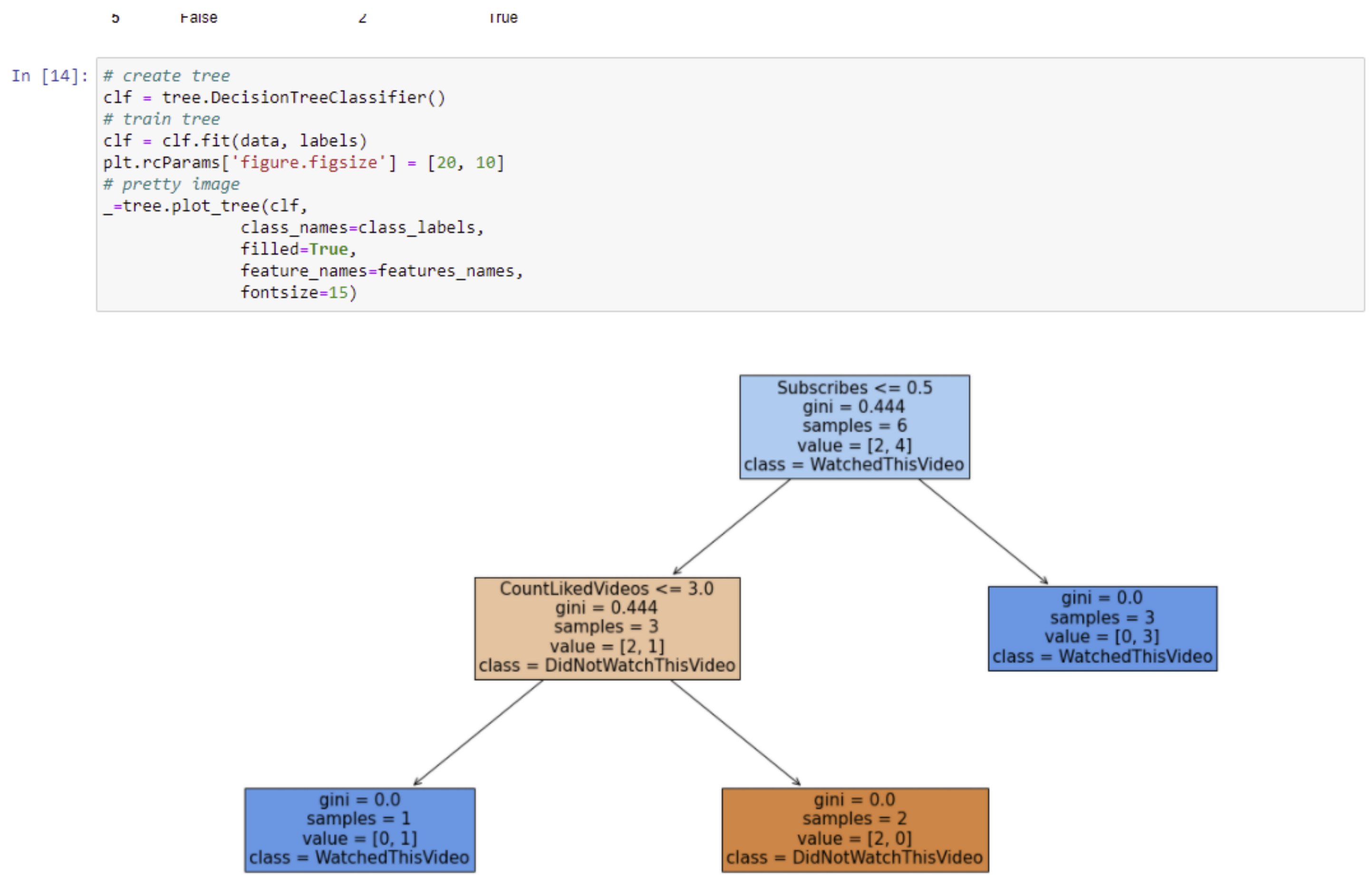
```
class_0:{'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 7}

class_1:{'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 19}

class_2:{'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 10}

accuracy:1.0

macro avg:{'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 36}

weighted avg:{'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 36}

<ipython-input-12-c13f26e9d22c>:2: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
nge the shape of y to (n_samples,), for example using ravel().
  clf.fit(train[features_names], train[labels])
```

```
In [13]: # make a simple example dataset
         features_names=["Subscribes","CountLikedVideos"]
         class_labels=["DidNotWatchThisVideo","WatchedThisVideo"]
         data=pd.DataFrame([[True,5],[True,1],[True,6],[False,7],[False,4],[False,2]],columns=features_names)
         labels=pd.DataFrame([True,True,True,False,False,True])
         all=data.copy()
         all['WatchedThisVideo']=labels
         all
```

Out[13]:

	Subscribes	CountLikedVideos	WatchedThisVideo
0	True	5	True
1	True	1	True
2	True	6	True
3	False	7	False
4	False	4	False
5	False	2	True

Fig. 5. Example of a figure caption.



```
5        False                    2                    True
```

```
In [14]: # create tree
         clf = tree.DecisionTreeClassifier()
         # train tree
         clf = clf.fit(data, labels)
         plt.rcParams['figure.figsize'] = [20, 10]
         # pretty image
         _=tree.plot_tree(clf,
                       class_names=class_labels,
                       filled=True,
                       feature_names=features_names,
                       fontsize=15)
```

Fig. 6. Example of a figure caption.