Μεταπτυχιακό μάθημα: “Εξόρυξη Δεδομένων”

1η Σειρά Ασκήσεων (ΒΑΣΙΛΗΣ ΓΚΟΛΕΣ ΑΜ:410)

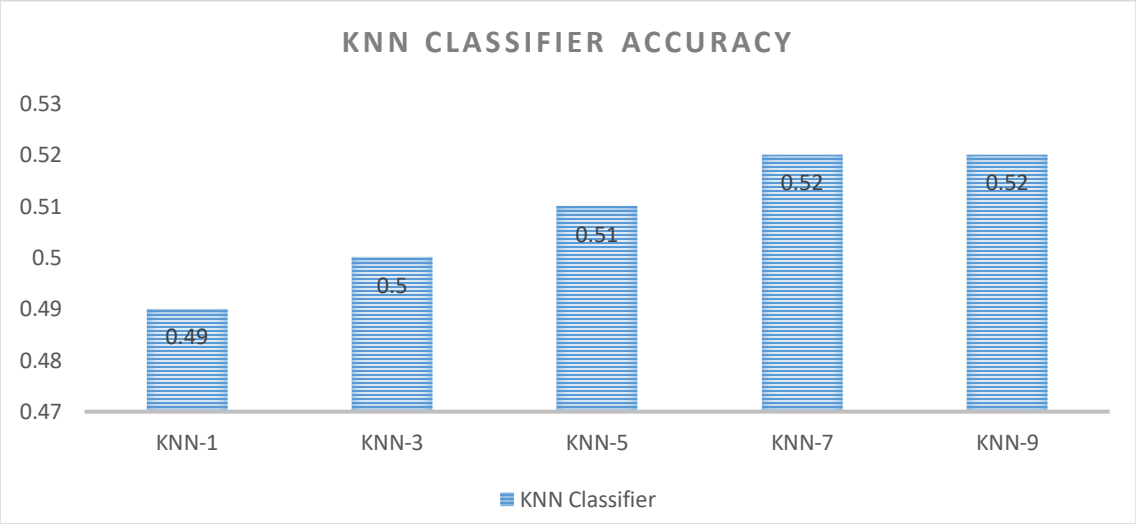
Ταξινόμηση συνόλου δεδομένων.

Για την άσκηση αυτή έγινε χρήση των μεθόδων ταξινόμησης της βιβλιοθήκης μηχανικής μάθησης sklearn της python.

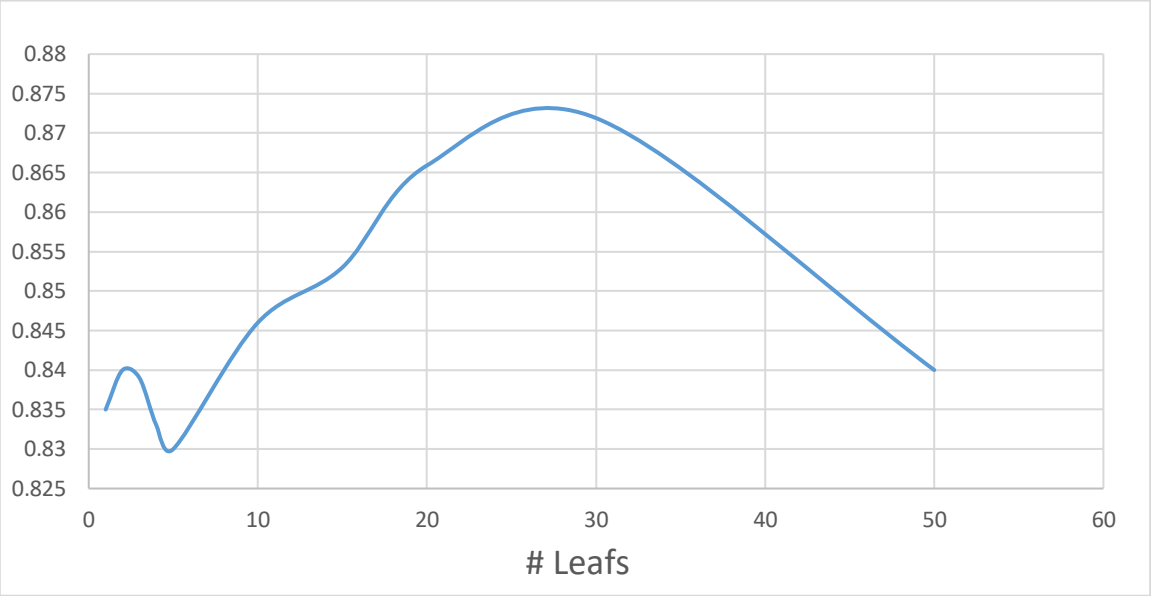
Αρχικά χρειάζεται να φορτώσουμε ένα csv αρχείο με τα δεδομένα μας.Το αρχείο αυτό περιέχει 17 στήλες απο τις οποίες μόνον η τελευταία δηλώνει την κατηγορία των δεδομένων. Οπότε σε μία λίστα που την ονομάζω evidence αποθηκεύω τις 16 πρώτες στήλες και σε μία λίστα labels την τελευτάια στήλη για κάθε γραμμή του αρχείου μας.
Η λίστα labels περιέχει τις κατηγορίες των δεδομένων μας στη μορφή 0 και 1.
Η παραπάνω διαδικασία υλοποιείται στον κώδικα μας απο τη συνάρτηση load_data(filename).

Για την εκπαίδευση του ταξινομητή εκτελούμε 10-fold cross validation πάνω στα σύνολα evidence και labels που κατασκευάσαμε προηγουμένως και έτσι παίρνουμε ένα σύνολο εκπαίδευσης και ένα σύνολο ελέγχου κάθε φορά για 10 φορές όσα είναι και τα folds που ορίσαμε, και με βάση αυτά κάνουμε εκπαίδευση. Στη συνέχεια εξετάζουμε τη γενίκευση του μοντέλου μας σε δεδομένα που δεν έχει εκπαιδευτεί και συγκρίνουμε την αποδοση του με την μετρική accuracy παίρνοντας το μέσο όρο για όλες τις περίπτωσεις. Το μοντέλο μας ορίζεται στον κώδικα σε σχόλια και κάθε φορά μπορούμε να αφαιρέσουμε τα σχόλια και να επίλέξουμε πιο μοντέλο ταξινόμησης θα χρησιμοποιήσουμε.
Η παραπάνω διαδικασίες υλοποιούνται στον κώδικα μας απο τις συνάρτησεις:
evaluate(labels, predictions), train_model(evidence, labels) και main().

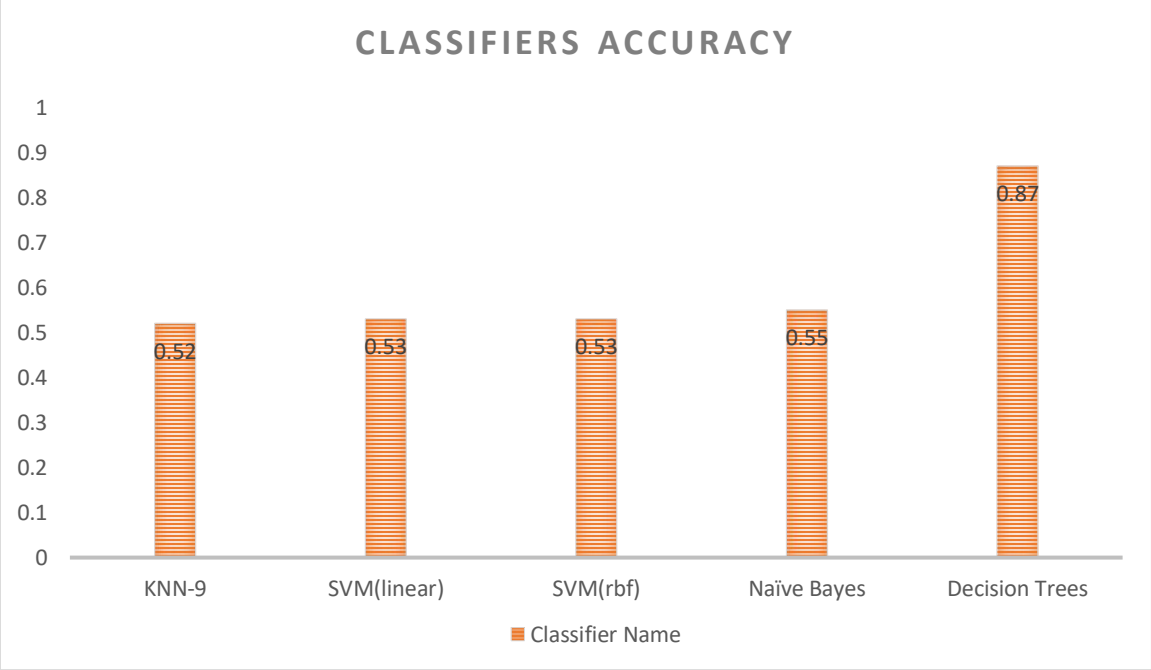
Παρακάτω δίνονται μερικά διαγράμματα εκτίμησης ακριβείας για διάφορους ταξινομητές.



1.1 Ακρίβεια πρόβλεψης για KNN με 1,3,5,7 και 9 γειτόνους αντίστοιχα.



1.2 Ακρίβεια πρόβλεψης Decision Trees με διάφορες τιμές της παραμέτρου leaf size



1.3 Σύγκριση διαφόρων ταξινομητών ως προς την ακρίβεια πρόβλεψης.

Παρατηρούμε ότι ο ταξινομητής Decision Tree έχει την καλύτερη ακρίβεια πρόβλεψης σε σχέση με τους υπόλοιπους.

Παράθεση του κώδικα σε Python.

```
import csv
import sys

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import KFold
from sklearn import tree

def main():

    filename = 'C:/Users/iTTaste/Desktop/CS50Beyond/DM_Project/dataset.csv'

    # Load data and split into train and test set
    evidence, labels = load_data(filename)

    # Estimate accuracy using cross-validation
    kfold = KFold(n_splits = 10, shuffle = True, random_state = 1)
    scores = []

    for train_index, test_index in kfold.split(evidence):
        X_train, X_test, y_train, y_test = np.array(evidence)[train_index], np.ar
ray(evidence)[test_index], \
            np.array(labels)[train_index], np.array(labels)[test_index]
        model = train_model(X_train, y_train)
        predictions = model.predict(X_test)
        sensitivity = evaluate(y_test, predictions)
        scores.append(sensitivity)
    print('Mean score is: ',np.mean(scores))

def load_data(filename):
    """
    Load dataset from a CSV file and convert them into a list of
    evidence lists and a list of labels. Return a tuple (evidence, labels)
    evidence should be a list of lists, labels should be the corresponding
    list of labels.Labels should be 1 or 0.
    """
    evidence = []
    labels = []
    with open(filename) as f:
        reader = csv.reader(f)
        next(reader)

        for row in reader:
```

```
            evidence.append(
                [float(cell) for cell in row[0 : 15]]
            )
            labels.append(1 if row[16] == '1' else 0)
    #print('Evidence',evidence)
    return (evidence, labels)

def train_model(evidence, labels):
    """
    Given a list of evidence lists and a list of labels, return
    the coresponding model each time.
    """

    #model = svm.SVC()
    model = GaussianNB()
    #model = tree.DecisionTreeClassifier()
    #model = KNeighborsClassifier(n_neighbors = 2)
    return model.fit(evidence, labels)

def evaluate(labels, predictions):
    # Compute how well we performed
    total = 0
    correct = 0
    incorrect = 0
    acuracy = 0.0
    for actual, predicted in zip(labels, predictions):
        total += 1
        if actual == predicted:
            correct += 1
        else:
            incorrect += 1
    acuracy = correct / total
    return(acuracy)

if __name__ == "__main__":
    main()
```