**ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ**

**ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ**

**ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ**

**ΤΟΜΕΑΣ ΒΙΟΜΗΧΑΝΙΚΗ ΔΙΟΙΚΗΣΗΣ**

**Βαρνάς Γεώργιος**

**ΑΕΜ: 6132**

**3η εργασία στο μάθημα «Ανάλυση Δεδομένων» #MENG228 2023-2024**

**Εισαγωγή**

Η παρούσα εργασία αποτελεί συνέχεια της προηγούμενης(2ης) εργασίας στο μάθημα, όπου τέθηκε ένα πρόβλημα ταξινόμησης(classification). Έτσι και τώρα θα εξετάσουμε κάποιες περαιτέρω μεθόδους ταξινόμησης πάνω στο ίδιο dataset που χρησιμοποιήθηκε στις προηγούμενες εργασίες, στο οποίο επιλέγουμε να ταξινομήσουμε τα δεδομένα μας σε υψηλό(1) ή χαμηλό(0) life expectancy(προσδόκιμο όριο ζωής)/εξαρτημένη μεταβλητή χρησιμοποιώντας αρκετές ανεξάρτητες μεταβλητές(predictors). Για τις μεθόδους ταξινόμησης που θα εφαρμοσθούν θα ελέγξουμε την ακρίβεια που επιτυγχάνεται στα δεδομένα ελέγχου(test accuracy), καθώς και πως μεταβάλλεται το test error όταν αλλάζουμε ορισμένες από τις παραμέτρους του κάθε μοντέλου. Σε όλες τις μεθόδους γίνεται διαχωρισμός των δεδομένων σε δεδομένα εκπαίδευσης-ελέγχου(train-test split) με το 80% να είναι training data και το 20% να είναι test data.

1. **Δέντρο απόφασης(Decision Tree)**

Τα δέντρα απόφασης αποτελούν έναν αρκετά διαδεδομένο αλγόριθμο μηχανικής μάθησης που χρησιμοποιείται τόσο για προβλήματα παλινδρόμησης όσο και για προβλήματα ταξινόμησης, όπως στην περίπτωσή μας. Ένα δέντρο απόφασης έχει μια μορφή αντίστοιχη με ένα διάγραμμα ροής, όπου κάθε εσωτερικός κόμβος υποδηλώνει μία από τις ανεξάρτητες μεταβλητές βάση της οποίας χωρίζονται τα δεδομένα σε δύο ή περισσότερες κλάσεις. Ένας κόμβος μπορεί μετά το διαχωρισμό των δεδομένων να οδηγεί σε μία από τις 2 ή περισσότερες κλάσεις του προβλήματός μας(leaf/terminal node) ή να οδηγεί σε έναν νέο κόμβο που περιέχει κάποιο άλλο feature/ανεξάρτητη μεταβλητή για εκ νέου διαχωρισμό(decision/internal node). Ο αλγόριθμος επιλέγει το πιο σημαντικό για τον διαχωρισμό των δεδομένων feature, χρησιμοποιώντας διάφορα πιθανά κριτήριο(π.χ. Entropy, Gini impurity). Η διαδικασία ένταξης νέων κόμβων συνεχίζεται μέχρι κάποια όρια(π.χ. ορισμένο maximum tree depth, ορισμένος minimum αριθμός δειγμάτων σε κάποιο κόμβο). Η αδιάκοπη προσθήκη κόμβων που δεν συνεισφέρουν ουσιαστικά στην επιθυμητή ταξινόμηση οδηγεί σε overfitting. Μάλιστα για να μην έχουμε overfitting, αρκετές φορές οδηγούμαστε στην αφαίρεση κόμβων από το δέντρο μας, το γνωστό κλάδεμα(tree pruning).

Δημιουργούμε ένα δέντρο απόφασης με μέγιστο βάθος(max depth) ίσο με 3, το κάνουμε fit στα δεδομένα εκπαίδευσής μας και εκτελούμε προβλέψεις για το πρόβλημα ταξινόμησής μας και υπολογίζουμε την ακρίβεια στα δεδομένα ελέγχου, αλλά και το classification report :

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

1. **Bagging**

Επόμενη τεχνική που θα εφαρμόσουμε για την ταξινόμηση είναι το bagging(bootstrap aggregating). Η μέθοδος μηχανικής μάθησης αυτή ανήκει στην οικογένεια των μεθόδων εκμάθησης συνόλου(ensemble learning) και χρησιμοποιείται τόσο για παλινδρόμηση όσο και για ταξινόμηση. Συγκεκριμένα για προβλήματα ταξινόμησης, ο αλγόριθμος bagging περιλαμβάνει τα εξής βήματα:

* Bootstrap sampling: δημιουργούνται n υποσύνολα του σετ δεδομένων με τυχαία δειγματοληψία με αντικατάσταση.
* Base model training: επιλέγουμε έναν αλγόριθμο μηχανικής μάθησης για ταξινόμηση(π.χ. decision trees) και τον εφαρμόζουμε για εκπαίδευση ξεχωριστά σε καθένα από όλα τα διαφορετικά υποσύνολα που έχουν δημιουργηθεί
* Aggregation(majority voting): αφού εκπαιδευτούν όλα τα μοντέλα, για να ταξινομηθεί ένα νέο data point, επιλέγεται η κλάση βάση πλειοψηφίας από το σύνολο των base models

Στο πρόβλημά μας χρησιμοποιούμε εκτιμήσεις από 200 δειγματοληψίες(n\_estimators=200). Τα αποτελέσματα(test accuracy και classification report) που προκύπτουν είναι τα εξής :

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

1. **Random Forest**

Επόμενη μέθοδος ταξινόμησης που εφαρμόζεται είναι η Random Forest. Η μέθοδος ανήκει και αυτή στην κατηγορία των ensemble learning μεθόδων και αποτελεί προέκταση της μεθόδου bagging, καθώς κατά την εκπαίδευση των μοντέλων για κάθε υποσύνολο του σετ δεδομένων δημιουργούνται πολλαπλά δέντρα απόφασης, καθένα από τα οποία περιλαμβάνει τυχαία υποσύνολα των features που συμμετέχουν στους κόμβους καθενός δέντρου. Για την πρόβλεψη της κλάσης στην οποία θα ταξινομηθεί ένα νέο data point, λαμβάνεται η κλάση που έχει την πλειοψηφία από τα δέντρα απόφασης που συμμετέχουν. Η αυξημένη πολυπλοκότητα του αλγορίθμου τον καθιστά αρκετά αποτελεσματικό για προβλήματα ταξινόμησης(πιο βελτιωμένη μέθοδος σε σύγκριση με τα δέντρα απόφασης και τη μέθοδο bagging), ενώ η τυχαιότητα που αφορά τα υποσύνολα των δεδομένων που λαμβάνονται, καθώς και τα features που μετέχουν στα δέντρα που περιέχονται σε ένα δάσος μειώνουν το ρίσκο του overfitting.

Δημιουργούμε ένα τυχαίο δάσος(random forest) με εκτιμήσεις από 200 δειγματοληψίες(n\_estimators=200) και τυχαίο πλήθος υποψήφιων χαρακτηριστικών m=, δηλαδή θέτουμε max\_features=’sqrt’(max\_features = sqrt(n\_features).

Λαμβάνουμε τα εξής αποτελέσματα:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

1. **Boosting**

Τελευταία μέθοδος ταξινόμησης που εφαρμόζεται είναι η μέθοδος boosting. Η μέθοδος αυτή ανήκει επίσης στις μεθόδους τύπου ensemble learning και βασική της ιδέα είναι η δημιουργία προβλέψεων μέσω της σταδιακής βελτίωσης από ένα αδύναμο μοντέλο σε ένα ελαφρώς ισχυρότερο και αντιστοίχως σε ένα τελικό μοντέλο ικανό για ισχυρές προβλέψεις. Σε κάθε επανάληψη(iteration) εντοπίζονται και δίνεται περισσότερη βαρύτητα στα data points που ταξινομήθηκαν εσφαλμένα στην ακριβώς προηγούμενη επανάληψη και έτσι ένα αρχικώς αφελές μοντέλο(weak learner) όπως π.χ. ένα πολύ απλό δέντρο απόφασης ισχυροποιείται σταδιακά. Υπάρχουν αρκετοί διαφορετικοί αλγόριθμοι boosting. Ορισμένοι από τους πιο συνηθισμένους είναι οι AdaBoost, Gradient Boosting Machine(GBM), XGBoost κ.α. Γενικά αρκετές φορές οι αλγόριθμοι boosting είναι επιρρεπείς σε overfitting και γι’ αυτό είναι απαραίτητο να βελτιστοποιήσουμε τις υπερπαραμέτρους τους.

Οι μέθοδοι bagging και boosting έχουν το κοινό ότι δεν είναι συγκεκριμένοι αλγόριθμοι ταξινόμησης αλλά ακολουθούν μια πιο γενικευμένη μεθοδολογία σε συνδυασμό με άλλους γνωστούς προορισμένους αλγορίθμους για να κάνουν ισχυρές προβλέψεις. Βασική τους διαφορά είναι ότι η μέθοδος bagging επιδιώκει να μειώσει την διακύμανση, ενώ η μέθοδος boosting να μειώσει τη μεροληψία.

Για το πρόβλημά μας χρησιμοποιούμε τον αλγόριθμο AdaBoost με εκτιμήσεις από 200 μοντέλα(n\_estimators=200), συντελεστή μάθησης(learning rate) ίσο με 1 και μέγιστο βάθος ίσο με 1.

Υπολογίζουμε ακρίβεια στα δεδομένα ελέγχου και το classification report:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

1. **Γραφική παράσταση του test error συναρτήσει του βάθους για το δέντρο ταξινόμησης**

Θα αναπαραστήσουμε το ζητούμενο γράφημα για τιμές βάθους από 1 έως 30. Το γράφημα που προκύπτει είναι το εξής:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμμή, διάγραμμα

Περιγραφή που δημιουργήθηκε αυτόματα

Όπως παρατηρείται στο διάγραμμα, αρχικά και για χαμηλές τιμές του εξεταζόμενου βάθους, όσο αυξάνεται το βάθος, μειώνεται το test error, κάτι που είναι απολύτως λογικό, διότι το μοντέλο γίνεται πιο σύνθετο(μπορεί να εντάξει περισσότερα features) και πιο ικανό να εντοπίζει πολύπλοκα patterns στα δεδομένα εκπαίδευσης, στα οποία προσαρμόζεται καλύτερα. Ωστόσο, φτάνοντας σε depth=8 βλέπουμε το ελάχιστο test error στο διάγραμμα και έπειτα για depth=9 μια αύξηση στο test error που συνοδεύεται από σταθεροποίησή του για depth>=10. Αυτό οφείλεται σε overfitting. Έπειτα η σταθεροποίηση του test error σημαίνει προφανώς ότι για max depth>8 δεν αλλάζει το δέντρο που προκύπτει οπότε ουσιαστικά έχω το ίδιο ακριβώς μοντέλο και άρα το ίδιο test error.

1. **Γραφική παράσταση του test error συναρτήσει του αριθμού δειγματοληψιών(n estimators) για τα μοντέλα bagging, Random Forest, boosting**

Ορίζουμε τον αριθμό των δειγματοληψιών που θα χρησιμοποιήσει ο κάθε αλγόριθμος για τις εκτιμήσεις σε ένα εύρος από 10 έως 500 με βήμα 10.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμμή, γράφημα

Περιγραφή που δημιουργήθηκε αυτόματα

Για τον αλγόριθμο bagging, βλέπουμε καταρχάς μια αρχική μείωση του test error όσο προσθέτουμε δειγματοληψίες προς χρήση από τον αλγόριθμο. Το ελάχιστο test error επιτυγχάνεται για αρκετούς πιθανούς αριθμούς δειγματοληψιών(n estimators = 50,60,70,90) και έπειτα για περαιτέρω αύξηση των estimators, το test error αυξάνεται λόγω overfitting. Επιπλέον για αρκετά μεγάλο αριθμό δειγματοληψιών, περαιτέρω αύξηση των estimators δεν αλλάζει καθόλου το test error, καθώς οι δειγματοληψίες είναι ήδη αρκετές και το μοντέλο δεν παρουσιάζει βελτίωση.

Εικόνα που περιέχει κείμενο, γραμμή, στιγμιότυπο οθόνης, διάγραμμα

Περιγραφή που δημιουργήθηκε αυτόματα

Για τον αλγόριθμο Random Forest, για τους ίδιους λόγους που επεξηγήθηκαν και παραπάνω, το test error παρουσιάζει μια αρχική μείωση και έπειτα σταθεροποιείται και παρουσιάζει μια flat συμπεριφορά που καταδεικνύει ότι έχει ενταχθεί επαρκής αριθμός δέντρων(estimators). Το test error δεν αυξάνεται ξανά, γεγονός που μας δείχνει ότι ο αλγόριθμος δεν κάνει overfitting, ενώ για ορισμένες τιμές του αριθμού δέντρων μειώνεται λίγο ακόμα το test error.

Εικόνα που περιέχει κείμενο, γραμμή, στιγμιότυπο οθόνης, διάγραμμα

Περιγραφή που δημιουργήθηκε αυτόματα

Για τον αλγόρθμο boosting, επίσης παρατηρούμε μια αρχικά απότομη μείωση του test error και όσο αυξάνεται ο αριθμός των estimators(δηλαδή των μέγιστων επαναλήψεων/βελτιώσεων που εκτελεί ο αλγόριθμος), η μείωση του test error γίνεται με όλο και μικρότερο ρυθμό. Παρότι η μέθοδος boosting τήνει αρκετά συχνά να κάνει overfitting, εδώ δεν φαίνεται κάτι τέτοιο, καθώς το test error δεν εμφανίζει αλλαγή τάσης για μεγάλο αριθμό estimators.

**Σύγκριση μεθόδων**

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Η μέθοδος Random Forest φαίνεται να είναι η βέλτιστη από τις 4 για την ταξινόμηση των δεδομένων μας, αρκετά κοντά με τη μέθοδο boosting, και εμφανώς καλύτερα από τις άλλες 2 μεθόδους(decision trees, bagging). Όλες οι μέθοδοι πάντως έχουν αρκετά καλή ακρίβεια. Το δέντρο ταξινόμησης, ως η πιο απλοϊκή από τις 4 μεθόδους αναμέναμε να έχει την χαμηλότερη ακρίβεια. Η μέθοδος bagging γενικά μειώνει τη διακύμανση, αλλά συχνά δεν εντοπίζει κάποιες αρκετά περίπλοκες σχέσεις στα δεδομένα.

Python Code:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import AdaBoostClassifier

# Read our data, create the dataframe

df1 = pd.read\_excel(r"c:\Users\Γιωργος\Desktop\MECHANICAL ENGINEERING\4ο ΕΤΟΣ\ΑΝΑΛΥΣΗ ΔΕΔΟΜΕΝΩΝ\1η εργασια\Data Analysis\_2024 1st Case\_Data.xlsx")

print(df1.head())

print(df1.describe())

print(df1.info())

# Check for NaN values

print(df1.isnull().values.any()) #prepei na vgazei False, an den exw NaN values se olo to dataframe

# Remove years 2007,2013 from the dataframe

i = df1[(df1['Year']==2013) | (df1['Year']==2007)].index

print(i)

df1.drop(index=i,inplace=True)

print(df1.info()) #gia epalitheysh, se sygrish me to prohgoumeno info

# Convert Life expectancy into a dummy variable

avg\_life\_expectancy = df1['Life expectancy '].mean()

print(avg\_life\_expectancy)

life\_expectancy\_dummies = pd.get\_dummies(df1['Life expectancy '] > avg\_life\_expectancy ,

                                        prefix='Life expectancy ', drop\_first=True)

life\_expectancy\_dummies = life\_expectancy\_dummies.astype(int)

print(life\_expectancy\_dummies)

print(life\_expectancy\_dummies.value\_counts()) #epalitheysh

df2 = pd.concat([df1, life\_expectancy\_dummies], axis=1)

df2.drop(['Life expectancy ','Year','Country', 'Status'], axis=1, inplace=True)

print(df2.head())

print(df2.info())

print(df2.columns)

y = df2['Life expectancy \_True']

X = df2[['Adult Mortality', 'Alcohol', 'percentage expenditure', 'Hepatitis B',

       'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',

       'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population', ' thinness 5-9 years',

       'Income composition of resources', 'Schooling',]]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=101)

# Decision Tree

m1 = DecisionTreeClassifier(max\_depth=3)

m1.fit(X\_train,y\_train)

m1\_pred = m1.predict(X\_test)

m1\_acc = accuracy\_score(y\_test, m1\_pred)

print("Decision Tree test accuracy is:", m1\_acc)

print("\n")

print(classification\_report(y\_test,m1\_pred))

# Bagging

m2 = BaggingClassifier(estimator=DecisionTreeClassifier(max\_depth=3), n\_estimators=200, random\_state=101)

m2.fit(X\_train,y\_train)

m2\_pred = m2.predict(X\_test)

m2\_acc = accuracy\_score(y\_test, m2\_pred)

print("Bagging test accuracy is:", m2\_acc)

print("\n")

print(classification\_report(y\_test,m2\_pred))

# Random Forest

m3 =RandomForestClassifier(n\_estimators=200, max\_features='sqrt', random\_state=101)

m3.fit(X\_train,y\_train)

m3\_pred = m3.predict(X\_test)

m3\_acc = accuracy\_score(y\_test,m3\_pred)

print("Random Forest accuracy is:", m3\_acc)

print("\n")

print(classification\_report(y\_test,m3\_pred))

# Boosting

m4 = AdaBoostClassifier(n\_estimators=200, learning\_rate=1, algorithm="SAMME", random\_state=101)

m4.fit(X\_train,y\_train)

m4\_pred = m4.predict(X\_test)

m4\_acc = accuracy\_score(y\_test,m4\_pred)

print("Boosting accuracy is:", m4\_acc)

print("\n")

print(classification\_report(y\_test,m4\_pred))

# test error vs depth for decision tree

depths = range(1,31)

test\_errors1 = []

for md in depths:

       tree\_model = DecisionTreeClassifier(max\_depth=md, random\_state=101)

       tree\_model.fit(X\_train,y\_train)

       tree\_pred = tree\_model.predict(X\_test)

       tree\_test\_error = 1 - accuracy\_score(y\_test,tree\_pred)

       test\_errors1.append(tree\_test\_error)

plt.figure(figsize=(10,7))

plt.plot(depths, test\_errors1, marker='o', markerfacecolor='red')

plt.title('Test Error vs Depth of Decision Tree')

plt.xlabel('Depth of Decision Tree')

plt.ylabel('Test Error')

plt.show()

# test error vs n\_estimators for bagging

test\_errors2 = []

n\_estim = range(10,500,10)

for n in n\_estim:

       bagg = BaggingClassifier(estimator=DecisionTreeClassifier(max\_depth=3), n\_estimators=n, random\_state=101)

       bagg.fit(X\_train,y\_train)

       bagg\_pred = bagg.predict(X\_test)

       bagg\_test\_error = 1 - accuracy\_score(y\_test, bagg\_pred)

       test\_errors2.append(bagg\_test\_error)

plt.figure(figsize=(10,7))

plt.plot(n\_estim, test\_errors2, marker='o', markerfacecolor='red')

plt.title('Test Error vs Number of base estimators for Bagging')

plt.xlabel('Number of base estimators')

plt.ylabel('Test Error')

plt.show()

# test error vs n\_estimators for Random Forest

test\_errors3 = []

for n in n\_estim:

       rf =RandomForestClassifier(n\_estimators=n, max\_features='sqrt', random\_state=101)

       rf.fit(X\_train,y\_train)

       rf\_pred = rf.predict(X\_test)

       rf\_test\_error = 1 - accuracy\_score(y\_test, rf\_pred)

       test\_errors3.append(rf\_test\_error)

plt.figure(figsize=(10,7))

plt.plot(n\_estim, test\_errors3, marker='o', markerfacecolor='red')

plt.title('Test Error vs Number of base estimators for Random Forest')

plt.xlabel('Number of base estimators')

plt.ylabel('Test Error')

plt.show()

# test error vs n\_estimators for boosting

test\_errors4 = []

for n in n\_estim:

       boo = AdaBoostClassifier(n\_estimators=n, learning\_rate=1, algorithm="SAMME", random\_state=101)

       boo.fit(X\_train, y\_train)

       boo\_pred = boo.predict(X\_test)

       boo\_test\_error = 1 - accuracy\_score(y\_test, boo\_pred)

       test\_errors4.append(boo\_test\_error)

plt.figure(figsize=(10,7))

plt.plot(n\_estim, test\_errors4, marker='o', markerfacecolor='red')

plt.title('Test Error vs Number of base estimators for Boosting')

plt.xlabel('Number of base estimators')

plt.ylabel('Test Error')

plt.show()

all\_test\_acc = [m1\_acc, m2\_acc, m3\_acc, m4\_acc]

methods = ['Decision Tree', 'Bagging', 'Random Forest', 'Boosting']

results = pd.DataFrame(data=all\_test\_acc, index=methods, columns=['test accuracy'])

print(results)