**Gradient Boosting**

* Gradient Boosting vs AdaBoost: utilizarea arborilor mari, coeficientul egal pentru toți arborii și utilizarea erorii reziduale a arborelui anterior pentru elaborarea arborelui curent
* Algoritmul de implementare teoretica a Gradient Boosting pentru sarcini de regresie:
* Crearea modelului de baza ce reprezintă un arbore cu o frunza a cărei valoare yp este media datelor de la iesire y ***yp=mean(y)***
* Calcularea erorii reziduale ***e = y-yp***
* Crearea unui nou model ce va reprezenta un arbore iar drept date de ieșire se va considera eroarea reziduala e
* Determinarea predicție acestui model f1
* Înmulțirea predicției f1 cu un coeficient a (learning rate) și sumarea cu predicția anterioara ***F1=yp+a\*f1***
* Calcularea noii erorii reziduale ***e1 = y-F1***
* Repetarea procedurii de creare a modelului considerându-se ieșirea drept noua eroare reziduala e1, de determinare a predicției acestui model f2 și de actualizat predicția totala F2
* Algoritmul de implementare teoretica a Gradient Boosting pentru sarcini de clasificare:
* Crearea modelului de baza ce reprezintă un arbore cu o frunza a cărei valoare yp este logaritmul natural al cotei (odds) ***yp=ln(n\_clas1/n\_clas2)***
* Transformarea valorilor predicție yp in probabilitate de predictie ***p\_yp=exp(yp)/(1+exp(yp)***
* Calcularea erorii reziduale considerându-se valorile unei clase ***y=1*** si a celeilalte ***y=0*** și relația ***e = y-p\_yp***
* Crearea unui nou model ce va reprezenta un arbore iar drept date de ieșire se va considera eroarea reziduala e
* Determinarea predicție acestui model f1
* Transformarea valorii predicției f1 in unități ale logaritmului cotei ***F\_f1=f1/(p\_yp\*(1-p\_yp))***
* Înmulțirea predicției F\_f1 cu un coeficient a (learning rate) și sumarea cu predicția anterioara formându-se predicția totala ***F1=yp+a\*F\_f1***
* Transformarea valorilor predictiei totale F1 in probabilitate de predictie ***p\_F1=exp(F1)/(1+exp(F1)***
* Calcularea erorii reziduale considerându-se valorile unei clase ***y=1*** si a celeilalte ***y=0*** și relația ***e1 = y-p\_F1***
* Repetarea procedurii de creare a modelului considerându-se ieșirea drept noua eroare reziduala e1, de determinare a predicției acestui model f2 și de actualizat predicția totala F2
* Algoritmul de elaborare a modelului GradientBoostingClassifier în Scikit-Learn:
* importul datelor
* vizualizarea informației despre date ***df.info()***
* verificarea prezentei lipsurilor ***df.isna().sum()***
* Vizualizarea informatiei statistice despre date ***df.describe().transpose()***
* determinarea numărului de valori pentru clasele label ***sns.countplot(data=df,x=col\_label)***
* afișarea grafica a numărului de valori unice pe categorie ***sns.barplot(data=df.describe().transpose().reset\_index().sort\_values('unique'),x='index',y='unique')***
* crearea setului X si y ***X=df.drop(col\_label, axis=1), y=df[col\_label]***
* transformarea datelor categoriale in numerice in setul X ***X = pd.get\_dummies(X,drop\_first=True)***
* crearea seturilor de train si de test ***X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.15, random\_state=9)***
* importul algoritmului GradientBoostingClassifier ***from sklearn.ensemble import GradientBoostingClassifier***
* crearea modelului GradientBoostingClassifier implicit ***model = GradientBoostingClassifier ()***
* ajustarea modelului pe datele de training ***model.fit(X\_train, y\_train)***
* realizarea predicției ***model.predict(X\_test)***
* determinarea acurateti modelului ***accuracy\_score(y\_test,y\_pred)***
* determinarea confusion matrix a modelului ***confusion\_matrix(y\_test,y\_pred)***
* vizualizarea grafica a confusion matrix a modelului ***plot\_confusion\_matrix(model,X\_test,y\_test)***
* elaborarea classificasion report a modelului ***print(classification\_report(y\_test,y\_pred))***
* determinarea importantei fiecărei caracteristici ***model.feature\_importances\_***
* determinarea denumirii caracteristicii celei mai importante ***X.columns[model.feature\_importances\_.argmax()]***
* vizualizarea numărului de valori ale clasei label in funcție de caracteristică ce mai importanta ***sns.countplot(data=df,x=col\_importanta,hue=col\_label)***
* Influența unor hiper-parametri asupra modelului GradientBoostingClassifier în Scikit-Learn:
* crearea modelului GradientBoostingClassifier de baza ***model = GradientBoostingClassifier()***
* importul modulului GhidSearchCV f***rom sklearn.model\_selection import GridSearchCV***
* crearea dicționarului cu parametri pentru GhidSearchCV ***param\_grid***
* crearea modelului GhidSearchCV pe baza modelului GradientBoostingClassifier ***model\_grid = GridSearchCV(model,param\_grid)***
* ajustarea modelului pe datele de training ***model\_grid.fit(X\_train, y\_train)***
* realizarea predicției ***model\_grid.predict(X\_test)***
* vizualizarea celor mai buni hiperparametri ***model\_grid.best\_params\_***
* determinarea acurateței modelului ***accuracy\_score(y\_test,y\_pred)***
* determinarea confusion matrix a modelului ***confusion\_matrix(y\_test,y\_pred)***
* vizualizarea grafica a confusion matrix a modelului ***plot\_confusion\_matrix(model\_grid,X\_test,y\_test)***
* elaborarea classificasion report a modelului ***print(classification\_report(y\_test,y\_pred))***
* determinarea importantei fiecărei caracteristici ***car =model\_grid.*** ***best\_estimator\_.feature\_importances\_***
* structurarea importantei caracteristicilor intr-un df  ***caracteristici = pd.DataFrame(index=X.columns,data=car,columns=['Importanta'])***
* sortarea caracteristicilor după importanta ***caracteristici.sort\_values("Importanta",ascending=False))***
* selectarea doar a celor caracteristici cu importanta mai mare de 0.001 **caracteristici[caracteristici['Importanta']>0.001]**
* vizualizarea grafica a importantei caracteristicilor ***sns.barplot(data=caracteristici,x=caracteristici.index,y='Importanta')***