**Random Forest**

* Neajunsurile algoritmului Decision Tree: aceeași caracteristică drept rădăcină, unele caracteristic pot fi excluse și pericolul overfitting
* Selectarea aliatoare a subseturilor de caracteristici si elaborarea mai multor arbori
* Rezultatul predicției la clasificare: clasa ce obține majoritatea la parcurgerea tuturor arborilor
* Rezultatul predicției la regresie: valoarea media obținută la parcurgerea tuturor arborilor
* Hiper-parametrul Number of Estimators: numărul de arbori elaborați - valori recomandate între 64 -128
* Hiper-parametrul Features in Subsets: numarul de caracteristici incluse intr-un subset selectat aliator – valori recomandate sqrt(n) pentru clasificare și n/3 pentru regresie
* Hiper-parametrul Bootstrapping – valoarea implicită True: permite la elaborarea arborelui să se utilizeze doar unele linii cu date (inclusiv cu repetare) și nu întregul set de date
* Hiper-parametrul Out-ofBag Error – valoarea implicită False: permite calcularea erori dintre valoarea adevarată și valoarea predicției cu liniile de date care nu au fost incluse in Bootstrapping pentru un arbore
* Algoritmul de elaborare a modelului RandomForestClassifier în Scikit-Learn:
* importul datelor
* vizualizarea informației despre date ***df.info()***
* verificarea datelor lipsa ***df.isnull().sum()***
* excluderea datelor lipsa ***df.dropna()***
* transformarea caracteristicilor categoriale string in categoriale numerice ***pd.get\_dummies(df,drop\_first=True)***
* crearea setului X si y ***X=df.drop(col\_label, axis=1), y=df[col\_label]***
* crearea seturilor de train si de test ***X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=9)***
* importul algoritmului RandomForestClassifier ***from sklearn.ensemble import RandomForestClassifier***
* crearea modelului RandomForestClassifier de bază **model = RandomForestClassifier(random\_state=9)**
* 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 fiecarei caracteristici ***model.feature\_importances\_***
* Influența diferitor hiper-parametri a modelului RandomForestClassifier în Scikit-Learn:
* importul datelor
* vizualizarea informației despre date ***df.info()***
* vizuazlizarea dependentelor dintre caracteristici in functie de coloana label ***sns.pairplot(df,hue='col\_label')***
* crearea setului X si y ***X=df.drop(col\_label, axis=1), y=df[col\_label]***
* crearea seturilor de train si de test ***X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=9)***
* importul algoritmului RandomForestClassifier ***from sklearn.ensemble import RandomForestClassifier***
* crearea modelului RandomForestClassifier de bază **model = RandomForestClassifier(random\_state=9)**
* Crearea dicționarului cu diferite valori a hiper-parametrilor analizați ***param\_grid***
* Importul modelului GhidSearchCV ***from sklearn.model\_selection import GridSearchCV***
* Crearea modelului GhidSearchCV ***grid = GridSearchCV(model,param\_grid)***
* ajustarea modelului pe datele de training ***grid.fit(X\_train, y\_train)***
* vizualizarea parametrilor cei mai buni ***grid.best\_params\_***
* vizualizarea parametrului oob\_score\_ ***grid***.***oob\_score\_***
* realizarea predicției ***grid.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))***
* Crearea unei bucle de determinare a dependentei hiper-parametrului n\_estimators de eroare
* Vizualizarea depdendentei hiper-parametrului n\_estimators de eroare
* Modelul RandomForestRegressor versus alte modele în Scikit-Learn:
* importul datelor
* vizualizarea dependentei dintre coloane sns.scatterplot(x=col1,y=col2,data=df)
* crearea setului X si y ***X=df.drop(col\_label, axis=1), y=df[col\_label]***
* scalara valorilor lui X (o singura coloana) in gama -1...1 ***X = X.values.reshape(-1,1)***
* crearea seturilor de train si de test ***X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=9)***
* importul algoritmului LinearRegression ***from sklearn.linear\_model import LinearRegression***
* crearea modelului de baza RandomForestClassifier de bază **model0 = LinearRegression()**
* ajustarea modelului pe datele de training ***model0.fit(X\_train, y\_train)***
* realizarea predicției ***lr\_preds=model0.predict(X\_test)***
* determinarea rădăcinii erorii medii pătratice ***np.sqrt(mean\_squared\_error(y\_test,lr\_preds))***
* crearea unui grafic cu afișarea dependenței valorilor label de valorile de intrare si dependenței valorilor prezise de valorile proprii de intrare
* crearea unei funcții de ajustare a modelului, realizarea a predicției, determinare a erorii și afișare a graficului comparativ ***executie\_model***
* importul modulului pipeline pentru modelul liniar regression polinomial ***from sklearn.pipeline import make\_pipeline***
* importul modulului PolynomialFeatures pentru setarea gradului polinomului in pipeline ***from sklearn.preprocessing import PolynomialFeatures***
* crearea modelului LinearRegression polinomial cu ajutorul pipeline ***model1 = make\_pipeline(PolynomialFeatures(2),LinearRegression())***
* aplicarea functiei executie\_model asupra modelului LinearRegression polinomial
* importul algoritmului KNeighborsRegressor ***from sklearn.neighbors import KNeighborsRegressor***
* crearea modelului KNeighborsRegressor pentru diferite valori a parametrului n\_neighbors ***model3 = KNeighborsRegressor(n\_neighbors=n)***
* aplicarea functiei executie\_model asupra modelului KNeighborsRegressor
* importul algoritmului DecisionTreeRegressor ***from sklearn.tree import DecisionTreeRegressor***
* crearea modelului DecisionTreeRegressor ***model4= DecisionTreeRegressor()***
* aplicarea functiei executie\_model asupra modelului DecisionTreeRegressor
* importul algoritmului SVR ***from sklearn.svm import SVR***
* crearea modelului SVR ***svr = SVR()***
* importul modulului GhidSearchCV f***rom sklearn.model\_selection import GridSearchCV***
* crearea dictionarului cu parametri pentru GhidSearchCV ***param\_grid***
* crearea modelului GhidSearchCV pe baza modelului SVR ***model5 = GridSearchCV(svr,param\_grid)***
* aplicarea functiei executie\_model asupra modelului GhidSearchCV
* importul algoritmului RandomForestRegressor ***from sklearn.ensemble import RandomForestRegressor***
* crearea modelului RandomForestRegressor pentru diferite valori a parametrului n\_estimators ***model6 = RandomForestRegressor(n\_estimators=n)***
* aplicarea functiei executie\_model asupra modelului RandomForestRegressor
* importul algoritmului GradientBoostingRegressor ***from sklearn.ensemble import GradientBoostingRegressor***
* crearea modelului GradientBoostingRegressor implicit ***model7=*** ***GradientBoostingRegressor()***
* aplicarea functiei executie\_model asupra modelului GradientBoostingRegressor
* importul algoritmului AdaBoostRegressor ***from sklearn.ensemble import AdaBoostRegressor***
* crearea modelului AdaBoostRegressor implicit ***model8=*** ***AdaBoostRegressor()***
* aplicarea functiei executie\_model asupra modelului AdaBoostRegressor