Tema 2

Analiză de regresie pentru oferte de vânzări auto

Mineritul datelor și analiza datelor (MDAD)

Raport tehnic

**1. Citirea și încărcarea datelor din fișierul la dispoziție**

Datele sunt citite din fisierul CSV folosind biblioteca python “pandas”: pd.read\_csv(file, low\_memory=False). Mai multe detalii despre functia implementata sunt in Figura 6 din sectiunea Anexe.

**2. Transformarea datelor**

a. Din format CSV în formatul necesar pentru restul pipeline-ului

Formatul care faciliteaza manipularea si analiza datelor este cel oferit de biblioteca “pandas” a python. In cadrul temei am generat un dataframe pe baza datelor citite in format CSV: df = read\_data('auto\_train.csv') cu care am prelucrat datele mai departe.

b. Descoperirea și corectarea erorilor care au apărut din procedura de colectare

Pentru identificarea problemelor aparute in procedura de colectare am afisat diferite informatii ale dataset-ului, precum: dimensiunile dataset-ului, tipurile de date, min/max pentru anumite coloane numerice, valorile unice ale coloanelor, suma valorilor nule pe fiecare coloana. Astfel am descoperit:

* + - Sunt 19526 intrari si 60 de coloane
    - Tipuri de date si numarul coloanelor de acest tip:
      * object – 47
      * float64 – 11
      * int64 – 2
    - Majoritatea coloanelor contin valori NaN
    - Coloana “VIN (serie sasiu)” nu contine date concludente, ci doar un alt string “Vezi VIN-ul (seria de sasiu)” care ascundea VIN-ul pe platforma sursa
    - Anumite coloane erau de tip float desi era mai normal sa fie de tip int: Numar locuri, Capacitate cilindrica
    - Exista valori outliers pentru coloane precum “Pret”: >300.000 si “Putere”: < 40 si > 700
    - Mai mult de 15 coloane aveau peste 16000 de valori NaN dintr-un total de 19526, motiv pentru care le-am eliminat din set, le-am considerat irelevante pentru scopul temei.

In ceea ce priveste corectarea erorilor descoperite, am eliminat valorile outliers printr-o metoda de forma:

* df = df[(df['pret'] > 2000) & (df['pret'] < 300000)]

pastrand doar valorile din intervalul dorit, am continuat prin a inlocui valorile de tip NaN cu valori convenabile pentru tipul respectiv de date. Au fost patru cazuri specifice: inlocuirea NaN cu media valorilor coloanei, inlocuirea cu valori random alese din aceeasi coloana, inclocuirea cu o valoare anume si inlocuirea cu valori random dintr-un interval specificat. Pentru mai multe detalii, gasiti Figura 7 din secitunea Anexe.

* df[col].fillna(df[col].mean(), inplace=True)
* df[col] = df[col].apply(replace\_nan\_with, args=("value",))
* replace\_nan\_with\_random(df, col)
* replace\_nan\_with\_random\_da\_nu\_values(df, col)

Functia “replace\_nan\_with” este o metoda definita de mine care primeste un element si verifica daca acesta este NaN sau nu. Pentru datele noastre, am considerat ca aceasta asignare reflecta cat mai corect realitatea. Functia replace\_nan\_with\_random(df, col) primeste ca parametrii un dataframe si o coloana, obtine datele unice de pe coloana, iar apoi inlocuieste NaN cu valori random din cele de pe coloana respective. Metoda replace\_nan\_with\_random\_da\_nu\_values(df, col) este folosita pentru un caz punctual si anume pentru cazul cand inlocuim NaN cu valori random de Da / Nu. Mai multe detalii legate de implementarea acestor functii se gasesc in Figura 8 din sectiunea Anexe.

Pentru coloanele care au necesitat o conversie de tip, am folosit metoda astype(newType) expusa de dataframe-ul folosit sa stochez datele. Exemplu:

* df['Numar locuri'] = df['Numar locuri'].astype(int)

c. Adăugarea sau eliminarea de coloane (acolo unde este cazul, de exemplu prin transformarea celor existente)

La aceasta sectiune am eliminat din dataset in principiu acele coloane ce contineau un numar foarte mic de valori ce nu ar fi ajutat la obtinerea de informatie relevanta. Exemplu:

* df.drop("VIN (serie sasiu)", axis=1, inplace=True)

**3. Antrenarea si evaluarea modelelor de regresie**

In cadrul acestei teme, am antrenat si evaluat urmatoarele modelel de regresie: LinearRegression, RidgeRegression, DecisionTreeRegressor, RandomForestRegressor, ExtraTreesRegressor, AdaBoostRegressor si GradientBoostingRegressor. Algoritmii au fost obtinuti si rulati folosind biblioteca sklearn. In cele ce urmeaza voi explica detaliile tehnice ale functiilor pe care le-am implementat pentru a obtine rezultatele dorite, urmand ca in final sa vorbim despre rezultatele efective ale modelelor.

Pentru a incepe procesul de antrenare, trebuie sa alegem acele caracteristici ale setului care sunt cele mai relevante pentru predictia pretului, motiv pentru care am inceput prin a ilustra graficele Pearson si Spearman. Aceste grafice releva gradul de corelatie mai mare de 0.4 dintre caracteristici si pret, lucru ce se poate observa in Figura 1.

A close-up of a diagram

Description automatically generatedFigura 1: graficele Pearson si Spearman

In functie de rezultatele obtinute in urma trasarii graficelor anterioare, am extras caracteristicile si target-ul in doua variabile x respectiv y:

* x = df[['Anul fabricației', 'Putere', 'Capacitate cilindrica']]
* y = np.log1p(df['pret'])

Pentru y am aplicat o logaritmare care imbunatateste mult perfomantele deoarece reduce semnificativ diferenta dintre valorile extreme ale variabilei, avand in vedere ca vorbim despre un interval de valori [2000, 300.000], dupa normalizare.

Urmeaza impartirea setului de date in date de antrenare si date de testare. Am considerat 80% pentru antrenare si 20% pentru testare. Am folosit o functie oferita de biblioteca sklearn.model\_selection pentru aceasta:

* x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

Anumiti algoritmi de regresie contin hiperparametri care, prin ajustarea lor, aduc rezultate mai bune regresiei, motiv pentru care am definit parametrii initiali pentru fiecare algoritm ce permite o astfel de configurare (vezi Figura 9 din sectiunea Anexe). Parametrii initiali au fost definiti conform recomandarilor de specialitate pentru o configurare generic valabila.

Pentru obtinerea parametrilor optimi pentru dataset-ul meu, am folosit Grid Search. Astfel, am definit o functie care primeste un model de regresie, parametrii initiali si datele de antrenare, instantiaza GridSearchCV oferit de sklearn.model\_selection, il antreneaza pe datele de input si intoarce cei mai buni parametrii pentru modelul respectiv. Mai multe detalii legate de implementare gasiti in Figura 10 din sectiunea Anexe. Rezulta o lista cu cei mai buni hiperparametri pentru fiecare model de regresie (vezi Figura 11 din sectiunea Anexe).

Asadar, pentru fiecare model ce contine hiperparametrii, pasii pe care i-am realizat pentru evaluarea datelor sunt ca in exemplul urmator:

1. ridge\_model = RidgeCV()
2. best\_ridge\_model = get\_best\_hyperparams(ridge\_model, ridge\_initial\_hyperparams, x\_train, y\_train)
3. solution\_rm, score\_rm = run\_regression(best\_ridge\_model, x\_train, y\_train, x\_test, y\_test, df, "Ridge regression")

Se instantiaza modelul, se obtin cei mai buni hiperparametrii, iar apoi se ruleaza regresia. Pentru LinearRegression lipseste parte de obtinere a celor mai buni parametrii.

Am definit functia run\_regression care primeste ca intrare un model, datasetul initial si cele de antrenare si testare si intoarce datele prezise si score-ul regresiei. Flow-ul este destul de simplu, se face antrenarea modelului:

* model.fit(x\_train, y\_train)

apoi se fac predictiile:

* x\_train\_pred = model.predict(x\_train)
* x\_test\_pred = model.predict(x\_test)

Am folosit si metoda de cross-validation care returneaza eroare medie patratica sub radical pentru cele n fold-uri, in cazul meu am ales 3 deoarece dimensiunea setului de date permite o impartire cat mai exacta la 3:

* n\_folds = 3
* print('\nTrain RMSE: {}'.format(crossval\_rmse(model, n\_folds, df, x\_train, y\_train).mean()))
* print('Test RMSE: {}'.format(crossval\_rmse(model, n\_folds, df, x\_test, y\_test).mean()))

Metoda crossval\_rmse creeaza 3 fold-uri de cross-validation in mod aleator folosind KFold din biblioteca sklearn.model\_selection si intoarce eroare medie patratica sub radical. Pentru mai multe detalii legate de implementarea acestei functii, vezi Figura 12 din sectiunea Anexe.

Dupa afisarea RMSE pentru train si test, trasam graficul predictiei modelului si evaluam score-ul acestuia:

* score = model.score(x\_test, y\_test)

De asemenea, pentru a vedea implementarea efectiva a functiei run\_regression, gasiti Figura 13 din sectiunea Anexe.

In cele ce urmeaza, voi prezenta graficele corespunzatoare fiecarui model de regresie in care se reprezinta valorile reale, valorile prezise si linia de regresie.

Graficul din Figura 2 ilustreaza rezultatele regresiei liniare fara hiperparametrii (subpunctul a) si cu hiperparametrii (subpunctul b). Rezultatele sunt foarte asemanatoare, un scor mai bun il are regresia cu parametrii, insa prezenta unor anomalii ale datelor altereaza putin regresia.

A graph showing a line of dots

Description automatically generated with medium confidence

A graph showing a line of dots

Description automatically generated with medium confidence  
Figura 2: a) Grafic regresie liniara fara hiperparametrii b) Grafic regresie liniare cu hiperparametrii

Figura 3 ilustreaza doua modele de regresie cu rezultate mult mai bune: DecisionTree si RandomForest. Graficele subliniaza o predictie mult mai buna, anomaliile sunt tratate mai bine ceea ce conduce la un RMSE mai mic, deci la rezulatte mai bune.

A diagram of a graph

Description automatically generated A graph of a graph showing a line of dots

Description automatically generated with medium confidence

Figura 3: a) Grafic regresie DecisionTree b) Grafic regresie RandomForest

Dupa cum se observa in Figura 4, ExtraTrees returneaza un rezultata foarte similar cu cel al RandomForest, desi au principii de implementarea diferite. AdaBoost are o regresie diferita si mult mai slaba.

A diagram of a graph

Description automatically generated A graph of a graph showing a line of dots

Description automatically generated with medium confidence

Figura 4: a) Grafic regresie ExtraTrees b) Grafic regresie AdaBoost

In cele din urma, GradientBoosting pare ca ofera cele mai bune rezultate pentru dataset-ul nostru. Observam in Figura 5 cum datele sunt distribuite destul de normal, iar linia de regresie imparte graficul in doua sectiuni relativ similare.

A graph of a graph showing a line

Description automatically generated with medium confidence

Figura 5: Grafic regresie GradientBoosting

In tabelul de mai jos se regasesc rezultatele modelelor cu hiperparametrii initiali si cu hiperparametrii optimi, calculati cu GridSearchCV. Pentru anumite modele nu am reusit sa obtin performante radical mai bune, insa pentru AdaBoost si GradientBoosting diferenta este sesizabila. De asemenea, am rulat de mai multe ori algoritmul, insa rezulatele erau de cele mai multe ori aceleasi ori se modifica a 12 zecimala, motiv pentru care am considerat ca este in regula sa nu pun 3 – 4 seturi de rezultate cu impact insesizabil.

Modelul care ofera cea mai buna regresie pe setul nostru de date este GradientBoosting, fiind urmat de RandomForest si ExtraTrees.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Rulare cu hiperparametrii initiali** | | **Rulare cu cei mai buni hiperparametrii** | |
| **LinearRegression** | Train RMSE | 0.3293836733666 | Train RMSE | 0.3293836733666 |
| Test RMSE | 0.3369294767006 | Test RMSE | 0.3369294767006 |
| Score | 0.8457231446663 | Score | 0.8457231446663 |
| **RidgeCV** | Train RMSE | 0.3293835139645 | Train RMSE | 0.3293835674325 |
| Test RMSE | 0.3369293128996 | Test RMSE | 0.3369308209136 |
| Score | 0.8457248960880 | Score | 0.8457245152744 |
| **DecisionTree** | Train RMSE | 0.2711931093747 | Train RMSE | 0.2705386707228 |
| Test RMSE | 0.3211692760541 | Test RMSE | 0.3207208619963 |
| Score | 0.8944239434094 | Score | 0.8945357910187 |
| **RandomForest** | Train RMSE | 0.2495183516928 | Train RMSE | 0.2495092990512 |
| Test RMSE | 0.2817282241428 | Test RMSE | 0.2801147354646 |
| Score | 0.9111796802017 | Score | 0.9127852611072 |
| **ExtraTrees** | Train RMSE | 0.2571273373130 | Train RMSE | 0.2570067254778 |
| Test RMSE | 0.2920702417564 | Test RMSE | 0.2921488402866 |
| Score | 0.9103051069156 | Score | 0.9106395440455 |
| **AdaBoost** | Train RMSE | 0.3402028031144 | Train RMSE | 0.3463810872348 |
| Test RMSE | 0.3425029420900 | Test RMSE | 0.3308673732302 |
| Score | 0.8048826770013 | Score | 0.8183135107680 |
| **GradientBoosting** | Train RMSE | 0.2521434435242 | Train RMSE | 0.2401496461050 |
| Test RMSE | 0.2674757653052 | Test RMSE | 0.2644798196494 |
| Score | 0.9092051164760 | Score | 0.9182039906483 |

**4. Anexe**

def read\_data(path):

    return pd.read\_csv(path, low\_memory=False)

Figura 6: citire fisier CSV

df["Are VIN (Serie sasiu)"] = df["Are VIN (Serie sasiu)"].apply(replace\_nan\_with, args=("Nu",))

    replace\_nan\_with\_random(df, "Versiune")

    df["Km"].fillna(df["Km"].mean(), inplace=True)

    df["Putere"].fillna(df["Putere"].mean(), inplace=True)

    df["Capacitate cilindrica"].fillna(df["Capacitate cilindrica"].mean(), inplace=True)

    replace\_nan\_with\_random(df, "Transmisie")

    df["Consum Extraurban"].fillna(df["Consum Extraurban"].mean(), inplace=True)

    replace\_nan\_with\_random(df, "Cutie de viteze")

    df["Consum Urban"].fillna(df["Consum Urban"].mean(), inplace=True)

    df["Emisii CO2"].fillna(df["Emisii CO2"].mean(), inplace=True)

    replace\_nan\_with\_random(df, "Numar de portiere")

    replace\_nan\_with\_random(df, "Numar locuri")

    df["Se emite factura"].fillna("Nu", inplace=True)

    df["Eligibil pentru finantare"].fillna("Nu", inplace=True)

    df["Primul proprietar (de nou)"].fillna("Nu", inplace=True)

    df["Carte de service"].fillna("Nu", inplace=True)

    df["Fara accident in istoric"].fillna("Da", inplace=True)

    replace\_nan\_with\_random(df, "Audio si tehnologie")

    replace\_nan\_with\_random(df, "Confort si echipamente optionale")

    replace\_nan\_with\_random(df, "Electronice si sisteme de asistenta")

    replace\_nan\_with\_random(df, "Siguranta")

    replace\_nan\_with\_random(df, "Generatie")

    replace\_nan\_with\_random(df, "Norma de poluare")

    replace\_nan\_with\_random(df, "Optiuni culoare")

    replace\_nan\_with\_random(df, "Tara de origine")

    replace\_nan\_with\_random(df, "Data primei inmatriculari")

    replace\_nan\_with\_random(df, "Performanta")

    replace\_nan\_with\_random\_da\_nu\_values(df, "Inmatriculat")

Figura 7: inlocuire NaN cu valori concrete

def replace\_nan\_with(current\_el, new\_el):

    if current\_el is np.nan:

        return new\_el

    else:

        return current\_el

def replace\_nan\_with\_random(df, column):

    valori\_nan = df[column].isna()

    numar\_nan = valori\_nan.sum()

    valori\_aleatoare = df[column].dropna().sample(numar\_nan, replace=True)

    df.loc[valori\_nan, column] = valori\_aleatoare.values

def generate\_random\_da\_nu\_values(n):

    return np.random.choice(['Da', 'Nu'], size=n)

def replace\_nan\_with\_random\_da\_nu\_values(df, column):

    valori\_nan = df[column].isna()

    numar\_nan = valori\_nan.sum()

    valori\_aleatoare = generate\_random\_da\_nu\_values(numar\_nan)

    df.loc[valori\_nan, column] = valori\_aleatoare

Figura 8: detalii de implementare a functiilor de inlocuire a valorilor NaN

ridge\_initial\_params = {'alphas': [0.1, 1.0, 10.0]}

decision\_tree\_initial\_params = {'max\_depth': [None, 10, 20]}

random\_forest\_initial\_params = {'n\_estimators': [50, 100, 200]}

extra\_trees\_initial\_params = {'n\_estimators': [50, 100, 200]}

ada\_boost\_initial\_params = {'n\_estimators': [50, 100, 200]}

gradient\_boosting\_initial\_params = {'n\_estimators': [50, 100, 200]}

Figura 9: parametrii initiali

def get\_best\_hyperparams(model, initial\_hyperparams, x\_train, y\_train):

    # folosesc GridSearch pentru a gasi cei mai buni hiperparametrii

    grid\_search = GridSearchCV(model, initial\_hyperparams, cv=3)

    # antrenez modelul

    grid\_search.fit(x\_train, y\_train)

    # afisez cei mai buni hiperparametri obtinuti

    print("\nGridSearch - best hyperparams:", grid\_search.best\_params\_)

    # afisez cel mai bun scor obtinut

    print("GridSearch - best score:", grid\_search.best\_score\_)

    return grid\_search.best\_estimator\_

Figura 10: obtinerea celor mai buni hiperparametrii folosind GridSearch

ridge\_initial\_hyperparams = {'alphas': [0.1, 1.0, 30.0]}

decision\_tree\_initial\_hyperparams = {'max\_depth': [None, 10, 20]}

random\_forest\_initial\_hyperparams = {'n\_estimators': [50, 100, 250]}

extra\_trees\_initial\_hyperparams = {'n\_estimators': [50, 100, 300]}

ada\_boost\_initial\_hyperparams = {'n\_estimators': [50, 100, 200]}

gradient\_boosting\_initial\_hyperparams = {'n\_estimators': [50, 100, 2500]}

Figura 11: parametrii finali

def crossval\_rmse(model, n\_folds, df, x, y):

    # creez n fold-uri de cross-validare in mod aleator

    kf = KFold(n\_folds, shuffle=True).get\_n\_splits(df.values)

    # evaluez eroarea medie patratica sub radical pentru cele n folduri, folosind datele de input (antrenare sau testare)

    rmse = np.sqrt(-1 \* cross\_val\_score(model, x, y, scoring="neg\_mean\_squared\_error", cv=kf))

    return rmse

Figura 12: functia crossval\_rmse

def run\_regression(model, x\_train, y\_train, x\_test, y\_test, df, model\_name):

    print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print("Training and testing of: {}".format(model\_name))

    # antrenez modelul pe datele de antrenare

    model.fit(x\_train, y\_train)

    # calculez predictiile pentru datele de antrenare si testare

    x\_train\_pred = model.predict(x\_train)

    x\_test\_pred = model.predict(x\_test)

    # Definesc numarul de fold-uri pe care le face "crossval\_rmse". Aceasta functie creeaza n fold-uri

    # de cross validare in mod aleatoriu, returnand eroarea medie patratica sub radical pentru cele n fold-uri

    n\_folds = 3

    print('\nTrain RMSE: {}'.format(crossval\_rmse(model, n\_folds, df, x\_train, y\_train).mean()))

    print('Test RMSE: {}'.format(crossval\_rmse(model, n\_folds, df, x\_test, y\_test).mean()))

    # trasez graficul predictiei modelului

    plt.scatter(x\_train\_pred, y\_train, c = "blue",  label = "Training data")

    plt.scatter(x\_test\_pred, y\_test, c = "black",  label = "Validation data")

    plt.title(model\_name)

    plt.xlabel("Predicted values")

    plt.ylabel("Real values")

    plt.legend(loc = "upper left")

    plt.plot([x\_train\_pred.min(), x\_train\_pred.max()], [x\_train\_pred.min(), x\_train\_pred.max()], c = "red")

    plt.show()

    # Evaluez score-ul modelului

    score = model.score(x\_test, y\_test)

    print("\nScore of {}: {}\n".format(model\_name, score))

    return x\_test\_pred, score

Figura 13: functia run\_regression