

Paciento recepto panaudojimo nuspėjimo projektas

Darbą parengė:
Tautvydas Raibikis

Turinys

- Projekto sprendžiama problema ir tikslai
- Supažindinimas su duomenimis
- Projekto eiga ir priimti sprendimai
- Rezultatų matavimo metrikos
- Pasiekti rezultatai
- Demo (deployment)

Problema

Viešai prieinamuose Elektroninės Sveikatos Paslaugų ir Bendradarbiavimo Infrastruktūros Informacinės Sistemos e. recepto duomenyse pastebima, kad apie 15 proc. išduodamų receptų nėra panaudojami.

Vaistai, reikalaujantys recepto, dažniausiai yra reikalingi pacientui dėl rimtų medicininių priežasčių. Galbūt galima nuspėti, pagal recepto parametrus, ar klientas bus linkęs pasinaudoti jam išrašytu receptu.

Tikslai

- Išanalizuoti receptų duomenis
- Sukurti ML modelį, kuris susipažinęs su pateiktais duomenimis gebėtų nuspėti ar receptas bus panaudotas ar ne.
- Sukurtą modelį deploy'inti

Projekto eiga ir priimti sprendimai

Duomenys:

Nuskaityti 2022m. pirmų trijų mėnesių duomenys (naujausi pateikti), iš viso virš 4 mil. įrašų

Išanalizuoti požymiai ir pasirinkti tie, kurie gali padėti numatyti recepto statusą (target)

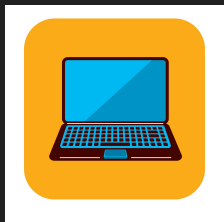
Analizė vykdyta 100 labiausiai pasikartojančių ligų

Duomenys išfiltruoti, kad egzistuotų tik panaudoto recepto arba nepanaudoto ir nebegaliojančio recepto įrašai

Train / test / validation duomenų rinkiniams sukurti pagamintas subalansuotas duomenų rinkinys (50/50 receptas panaudotas arba ne, 600 000 įrašų)

Aplinkos pasirinkimas

Kandidatas nr. 1



Kandidatas nr. 2



Kandidatas nr. 3



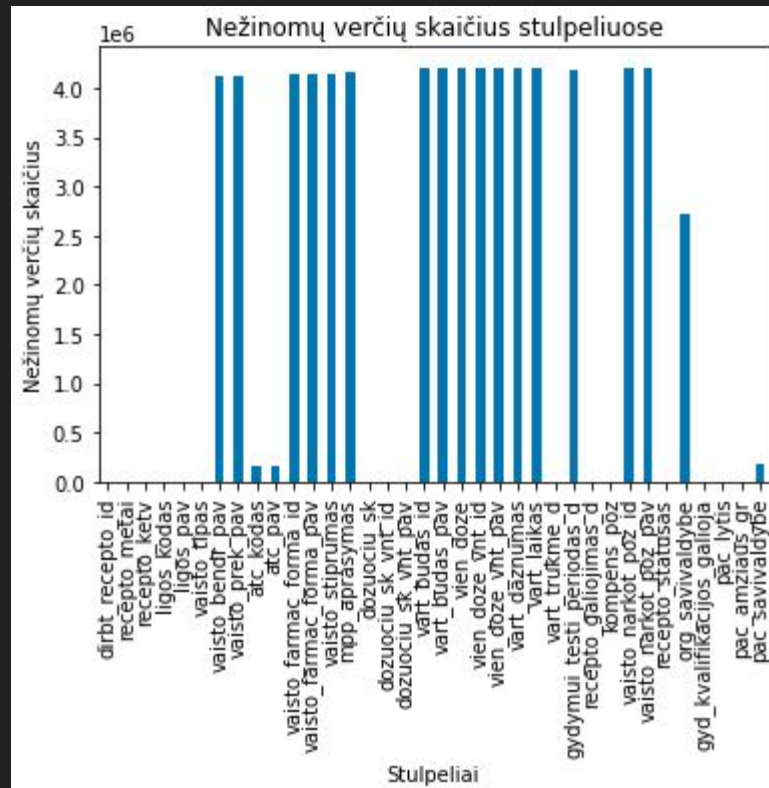
Kandidatas nr. 4



Duomenys:

Pastebėta, jog 17 požymių beveik
visos vertės yra nežinomos

Šių požymių iškart atsisakiau



Duomenys:

Pastebėta, jog yra požymių, kurie atspindi tą patį dalyką ir yra pasikartojantys:

ligos kodas / ligos pavadinimas

vaisto kodas / vaisto pavadinimas

Iš tokių požymių porų pasirinkau po vieną, kad išvengti pasikartojamumo bei papildomo kompleksiško modelyje.

Duomenys:

Taip pat atsisakyta tokių požymių, kaip identifikatoriai, metai, bei ketvirtis.

Galutiniai pasirinkti požymiai:

```
X = df_receptai_balanced[
    [
        "ligos_kodas",
        "atc_kodas",
        "dozuociu_sk",
        "recepto_galiojimas_d",
        "kompens_poz",
        "pac_savivaldybe",
        "vart_trukme_d",
        "pac_lytis",
        "pac_amziaus_gr",
    ]
]

y = df_receptai_balanced["recepto_statusas"]
```

Duomenys:

Nuspėti bandoma reikšmė yra recepto statusas.

Galimi recepto statusai:

completed - receptas panaudotas per jo galiojimo laiką

stopped - receptas nepanaudotas per jo galiojimo laiką

active - receptas nepanaudotas, bet vis dar galioja

on hold - vaistas pagal receptą yra rezervuotas

neatverinama - informacijos apie recepto panaudojimą nėra

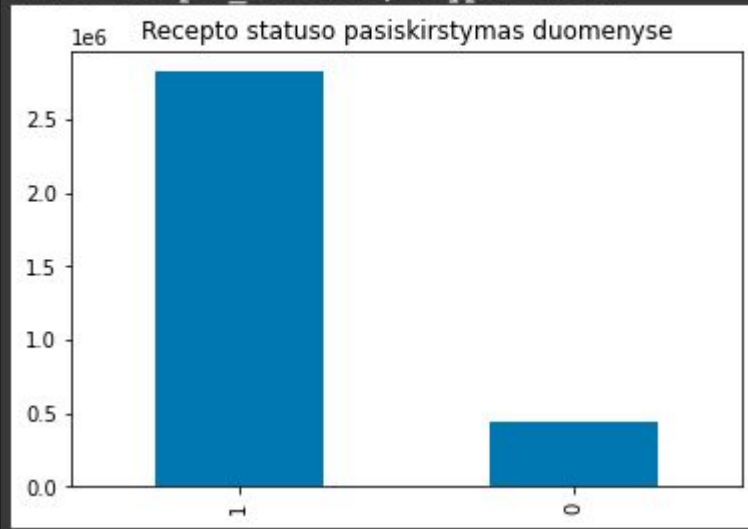
Duomenys išfiltruoti tik pagal completed ir stopped reikšmes

Duomenys:

Duomenų rinkinys po pradinio apdirbimo vis dar yra labai išbalansuotas

Sukurtas subalansuotas, apdirbamo dydžio duomenų rinkinys, turintis po lygiai reikšmių susijusių su panaudotu ir nepanaudotu receptu.

```
1    2825802  
0     446422  
Name: recepto_statusas, dtype: int64
```



```
df_negative = receptai_df.query(" recepto_statusas == 0").sample(300000)  
df_positive = receptai_df.query(" recepto_statusas == 1").sample(300000)
```

```
df_receptai_balanced = pd.concat([df_positive, df_negative], axis=0)
```

Projekto eiga ir priimti sprendimai

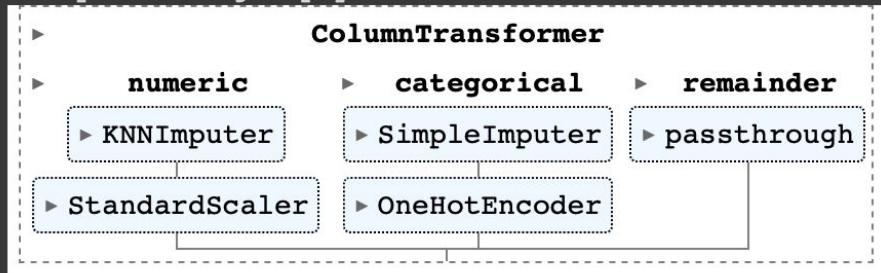
Duomenų apdirbimas:

Požymiai suskirstyti į kategorinius ir skaitinius

Tusčioms reikšmėms užpildyti panaudoti Scikit learn siūlomi imputer'iai

Skaitiniai požymiai scale'inti, kategoriniai OneHotEncode'inti

Preprocessing'o pipeline:



Po preprocessing pipeline duomenų rinkinys turi 946 požymius

Projekto eiga ir priimti sprendimai

Algoritmo pasirinkimas:

Išbandyta keletas algoritmų

Geriausi rezultatai pasiekti su
DecisionTreeClassifier ir
RandomForestClassifier

Šiems algoritmams atliktas
hyperparameter tuning
naudojant
RandomizedSearchCV
algoritmą

```
# apsibrėžiu hyperparametrus, kuriuos noriu tuninti RandomForestClassifier algoritmui
params_rfc = {
    "max_depth": list(np.arange(10, 100, step=10)) + [None],
    "n_estimators": np.arange(10, 500, step=50),
    "max_features": randint(1, 7),
    "min_samples_leaf": randint(1, 4),
    "min_samples_split": np.arange(2, 10, step=2),
}

# susikuriu random forest classifier objektą
rfc = RandomForestClassifier(n_jobs=-1)

# susikuriu randomized search objektą
rfc_random = RandomizedSearchCV(
    rfc, params_rfc, n_iter=10, scoring="accuracy", n_jobs=-1, cv=3
)

# fit'inu randomized search objektą su duomenimis, pritaikau preprocesavimo pipeline
model_rfc_random = rfc_random.fit(
    pd.DataFrame(preprocessor_pipeline.fit_transform(X_val)), y_val
)

# gražinu geriausius hyperparametrų nustatymus ir geriausią pasiektą tikslumą
print("Best hyperparameters are: " + str(model_rfc_random.best_params_))
print("Best score is: " + str(model_rfc_random.best_score_))
```

Best hyperparameters are:

```
{'max_depth': 60,
 'max_features': 4,
 'min_samples_leaf': 1,
 'min_samples_split': 8,
 'n_estimators': 260}
```

Best score is: 0.7637604166666666

Projekto eiga ir priimti sprendimai

Požymių pasirinkimas:

Tikslumui pagerinti bei modelio
kompleksiškumui sumažinti
naudojau RFE algoritmą (Recursive
Feature Elimination)

Geriausias rezultatas pasiektas su
350 požymių

```
# apsibrėžiu RFE objektą (recursive feature eliminator), kurį nadosiu final pipeline'e  
rfe_estimator = LinearRegression(n_jobs=-1)  
  
selector = RFE(rfe_estimator, n_features_to_select=350, step=50)
```

Projekto eiga ir priimti sprendimai

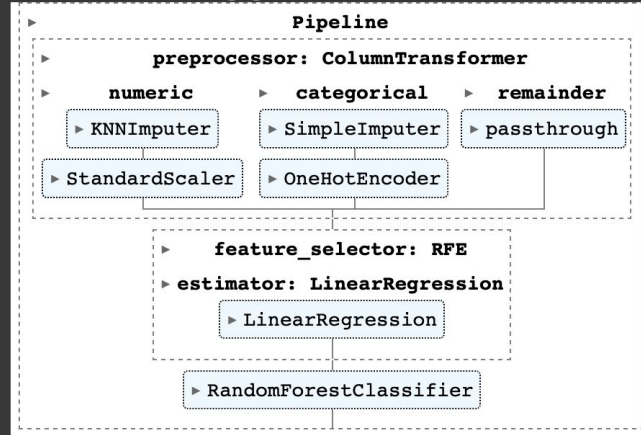
Modelio treniravimas:

Galiausiai, pasirinkus algoritmą, geriausią rezultatą teikiančius hyperparameters ir požymių kiekį, treniruojamas galutinis modelis

```
[82] # galutinis pipeline'as
classification_pipeline = Pipeline(
    steps=[
        ("preprocessor", preprocessor_pipeline),
        ("feature_selector", selector),
        ("classifier", forest_clf),
    ]
)

print("This is the final pipeline:")
classification_pipeline
```


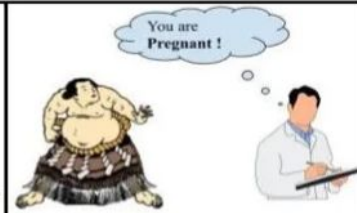
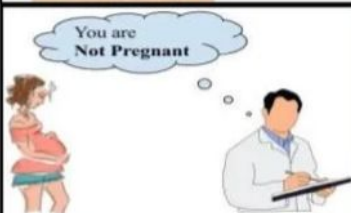

This is the final pipeline:



Rezultato matavimo metrikos

Mano problema yra Binary Classification, tokioje problemoje turime 4 rūšių spėjimus:

		PREDICTED LABEL	
		NEGATIVE	POSITIVE
TRUE LABEL	NEGATIVE	TRUE NEGATIVE	FALSE POSITIVE
	POSITIVE	FALSE NEGATIVE	TRUE POSITIVE

		PREDICTED LABEL	
		NEGATIVE	POSITIVE
TRUE LABEL	NEGATIVE	 TRUE NEGATIVE	 FALSE POSITIVE
	POSITIVE	 FALSE NEGATIVE	 TRUE POSITIVE

Rezultato matavimo metrikos

Precision - klasifikatoriaus gebėjimas nežymėti reikšmės teigiama, kai ji išties yra neigiama:

$$Precision = \frac{TP}{TP + FP}$$

Precision - klasifikatoriaus gebėjimas sėkmingai atrasti visas teigiamas reikšmes:

$$Recall = \frac{TP}{TP + FN}$$

Rezultato matavimo metrikos

F-1 score - harmoninis precision ir recall vidurkis:

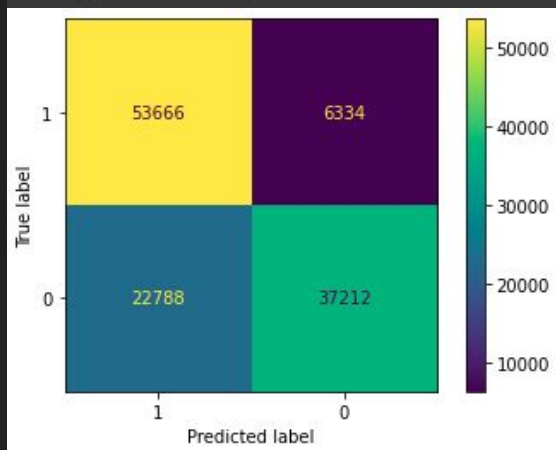
$$F1 \text{ Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Accuracy - teisingai atspėtų reikšmių bei visų spėjimams naudotų reikšmių santykis:

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN}$$

Pasiekti rezultatai

Model score: 0.757					
	precision	recall	f1-score	support	
0	0.85	0.62	0.72	60000	
1	0.70	0.89	0.79	60000	
accuracy			0.76	120000	
macro avg	0.78	0.76	0.75	120000	
weighted avg	0.78	0.76	0.75	120000	



GitHub repozitorija

https://github.com/import-tts/prescription_analyser

Demo time