Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Mentor: doc. dr. sc. Krešimir Križanović

Klasifikacija_bolesti_na_temelju_podataka_o_ekspresiji_gena

SEMINAR 1 2024./2025.

Domagoj Sviličić

http://www.fer.unizg.hr

Učitavanje potrebnih biblioteka

```
# Učitaj osnovne biblioteke...
import sklearn
import matplotlib.pyplot as plt
# %pylab inline
%matplotlib inline
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

1. Klasifikacija karcinoma dojke na temelju podataka o ekspresiji gena

Struktura dataseta za klasifikaciju karcinoma dojke

Izlučivanje potrebnih stupaca...

```
from sklearn.preprocessing import LabelEncoder

# Učitavanje podataka iz CSV datoteke
file_path = 'Breast_GSE45827.csv'  # zamijeniti s točnom putanjom do datoteke
data = pd.read_csv(file_path)

# Brojanje ukupnog broja stupaca u podacima
total_columns = data.shape[1]
print(f"Ukupan broj stupaca: {total_columns}")

# Izdvajanje oznaka klase i značajki
y = data.iloc[:, 1]  # drugi stupac s oznakama klase stupac type
```

```
print("-----------------------------")
print(f"Struktura podataka oznaka (y):\n{y}")
print("-----------------------------")
X = data.iloc[:, 2:]  # svi stupci nakon drugog s razinama ekspresije gena

# Kodiranje oznaka klasa
le = LabelEncoder()
y_encoded = le.fit_transform(y)

print("-----------------------------")
print(f"Struktura podataka oznaka (y_encoded) nakon enkodiranja oznaka:\n{y_encoded}")
print("-----------------------------")

# Provjera dimenzija
print(f"Dimenzije značajki (X): {X.shape}")
print(f"Dimenzije oznaka (y_encoded): {y_encoded.shape}")
```

```
Ukupan broj stupaca: 54677
-----------------------------
Struktura podataka oznaka (y):
0          basal
1          basal
2          basal
3          basal
4          basal
         ...
146    luminal_B
147    luminal_B
148    luminal_B
149    luminal_B
150    luminal_B
Name: type, Length: 151, dtype: object
-----------------------------
-----------------------------
Struktura podataka oznaka (y_encoded) nakon enkodiranja oznaka:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2
2 2
 2 2 2 2 2 2 2 2 2 2 5 5 5 5 5 5 5 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3
 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4
 4 4 4]
-----------------------------
Dimenzije značajki (X): (151, 54675)
Dimenzije oznaka (y_encoded): (151,)
```

1.1 Višeklasna (6 klasa) klasifikacija raka dojke logističkom regresijom

Klase su sljedeće: **basal**, **HER**, **luminal_A**, **luminal_B**, **cell_line** i **normal**. Prvih pet oznaka predstavljaju različite vrste raka dojke, dok oznaka **normal** označava da osoba ne boluje od nijedne vrste raka dojke.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Funkcija za treniranje modela i evaluaciju točnosti za dani omjer
podjele
def evaluate_model(X, y_encoded, test_size, random_state=1):
    # Podjela podataka
    X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=test_size, random_state=0)  # postavljanje random_state za
replikaciju rezultata
    # Treniranje modela logističke regresije
    model = LogisticRegression(max_iter=100, solver='lbfgs')  #
automatski odabir metode za višeklasnu/multinomijalnu klasifikaciju /
defaultno l2 regularizacija i multiclass=multinomial
    model.fit(X_train, y_train)
    # Predikcija i točnost na testnom skupu
    y_test_pred = model.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    # Predikcija i točnost na skupu za učenje
    y_train_pred = model.predict(X_train)
    train_accuracy = accuracy_score(y_train, y_train_pred)
    return test_accuracy, train_accuracy

# Evaluacija za različite omjere
accuracy_7_3_test, accuracy_7_3_train = evaluate_model(X, y_encoded,
test_size=0.3)
accuracy_6_4_test, accuracy_6_4_train = evaluate_model(X, y_encoded,
test_size=0.4)
accuracy_1_1_test, accuracy_1_1_train = evaluate_model(X, y_encoded,
test_size=0.5)

# Ispis rezultata
print("Točnost za različite omjere podjele podataka za višeklasnu (6
klasa) klasifikacija raka dojke logističkom regresijom:")
print(f"Omjer 7:3: Točnost na testnom skupu = {accuracy_7_3_test:.4f},
Točnost na skupu za učenje = {accuracy_7_3_train:.4f}")
print(f"Omjer 6:4: Točnost na testnom skupu = {accuracy_6_4_test:.4f},
Točnost na skupu za učenje = {accuracy_6_4_train:.4f}")
print(f"Omjer 1:1: Točnost na testnom skupu = {accuracy_1_1_test:.4f},
Točnost na skupu za učenje = {accuracy_1_1_train:.4f}")
```

```
Točnost za različite omjere podjele podataka za višeklasnu (6 klasa)
klasifikacija raka dojke logističkom regresijom:
```

```
Omjer 7:3: Točnost na testnom skupu = 0.8913, Točnost na skupu za
učenje = 1.0000
Omjer 6:4: Točnost na testnom skupu = 0.9344, Točnost na skupu za
učenje = 1.0000
Omjer 1:1: Točnost na testnom skupu = 0.9342, Točnost na skupu za
učenje = 1.0000
```

1.2 Višeklasna (6 klasa) klasifikacija karcinoma dojke pomoću SVM modela s linearnom jezgrenom metodom

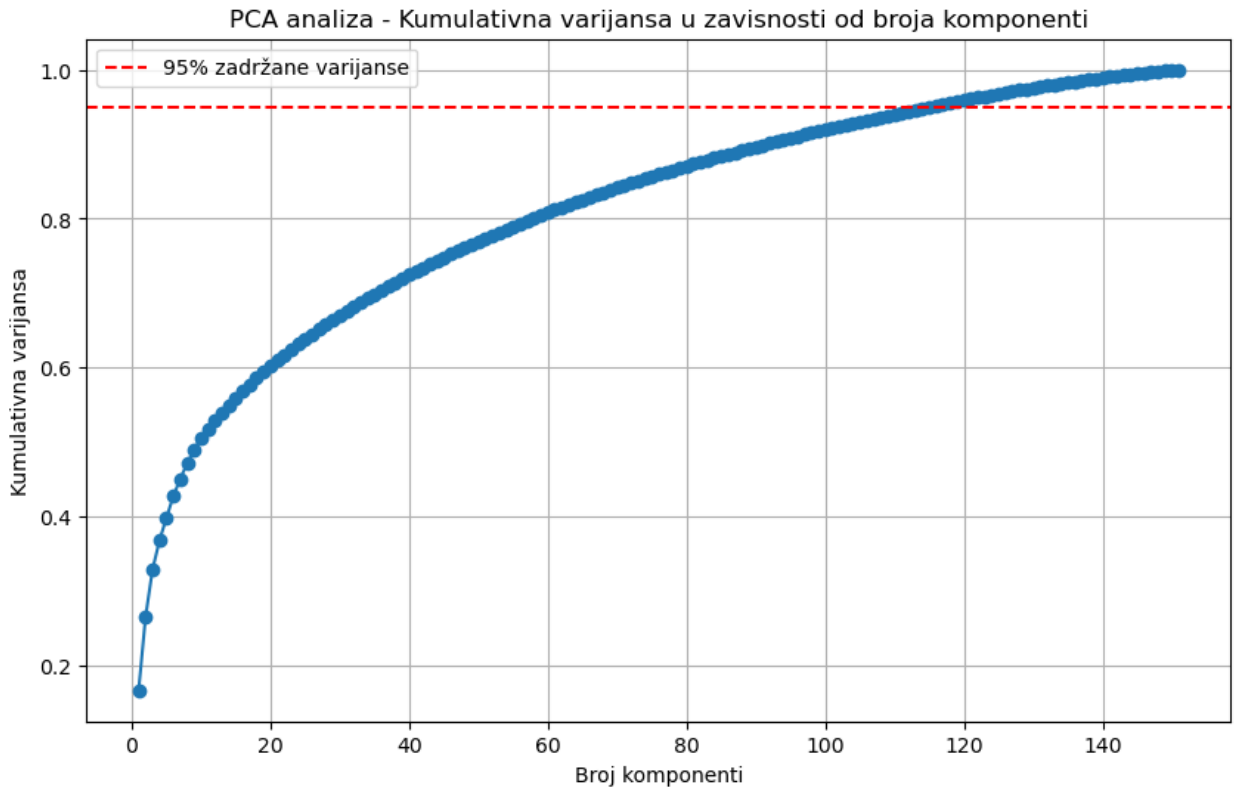
```
from sklearn.svm import SVC


# Funkcija za treniranje modela i evaluaciju točnosti za dani omjer
podjele
def evaluate_svm_model(X, y_encoded, test_size, random_state=0):
    # Podjela podataka
    X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=test_size, random_state=random_state)  # postavljanje
random_state za replikaciju rezultata
    # Treniranje SVM modela s linearnom jezgrom
    model = SVC(kernel='linear')  # SVM s linearnom jezgrom
    model.fit(X_train, y_train)
    # Predikcija i točnost na testnom skupu
    y_test_pred = model.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    # Predikcija i točnost na skupu za učenje
    y_train_pred = model.predict(X_train)
    train_accuracy = accuracy_score(y_train, y_train_pred)
    return test_accuracy, train_accuracy

# Evaluacija za različite omjere
accuracy_7_3_test, accuracy_7_3_train = evaluate_svm_model(X,
y_encoded, test_size=0.3)
accuracy_6_4_test, accuracy_6_4_train = evaluate_svm_model(X,
y_encoded, test_size=0.4)
accuracy_1_1_test, accuracy_1_1_train = evaluate_svm_model(X,
y_encoded, test_size=0.5)

# Ispis rezultata
print("Točnost za različite omjere podjele podataka za višeklasnu (6
klasa) klasifikacija raka dojke SVM s linearnom jezgrom:")
print(f"Omjer 7:3: Točnost na testnom skupu = {accuracy_7_3_test:.4f},
Točnost na skupu za učenje = {accuracy_7_3_train:.4f}")
print(f"Omjer 6:4: Točnost na testnom skupu = {accuracy_6_4_test:.4f},
Točnost na skupu za učenje = {accuracy_6_4_train:.4f}")
print(f"Omjer 1:1: Točnost na testnom skupu = {accuracy_1_1_test:.4f},
Točnost na skupu za učenje = {accuracy_1_1_train:.4f}")
```

```
Točnost za različite omjere podjele podataka za višeklasnu (6 klasa)
klasifikacija raka dojke SVM s linearnom jezgrom:
Omjer 7:3: Točnost na testnom skupu = 0.9130, Točnost na skupu za
učenje = 1.0000
Omjer 6:4: Točnost na testnom skupu = 0.9344, Točnost na skupu za
učenje = 1.0000
Omjer 1:1: Točnost na testnom skupu = 0.9474, Točnost na skupu za
učenje = 1.0000
```

1.3 Višeklasna (6 klasa) klasifikacija karcinoma dojke pomoću modela dubokog učenja i PCA metode

PCA (Principal Component Analysis) je tehnika smanjenja dimenzionalnosti koja se koristi za identificiranje glavnih komponenti u skupu podataka. Ove glavne komponente su novi, ortogonalni pravci (dimenzije) koji maksimalno zadržavaju varijansu originalnih podataka. Postupak započinje standardizacijom podataka, zatim se izračunava kovarijacijska matrica kako bi se utvrdila povezanost između značajki. Sljedeći korak je računanje vlastitih vrijednosti i vektora kovarijacijske matrice, koji određuju glavne komponente. Konačno, podaci se transformiraju u novi koordinatni sustav definiran tim glavnim komponentama, čime se smanjuje broj dimenzija uz minimalni gubitak informacija.

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt


# Primena PCA
pca = PCA()
X_pca = pca.fit_transform(X)

# Izračunavanje kumulativne varijanse
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)

# Prikazivanje grafa kumulativne varijanse
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,
marker='o')
plt.xlabel('Broj komponenti')
plt.ylabel('Kumulativna varijansa')
plt.title('PCA analiza - Kumulativna varijansa u zavisnosti od broja
komponenti')
plt.axhline(y=0.95, color='r', linestyle='--', label='95% zadržane
varijanse')
plt.legend()
plt.grid(True)
plt.show()
```



PCA reducira broj komponenti na otprilike 117 komponenti pri kojima je zadržano 95% varijance. Te komponente prosljeđujemo našoj neuronskoj mreži i nad njima treniramo mrežu.

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Podjela podataka
X_train, X_test, y_train, y_test = train_test_split(X_pca, y_encoded,
test_size=0.2)

# Definiranje modela
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))  #
Prvi skriveni sloj
model.add(Dense(32, activation='relu'))  # Drugi skriveni sloj
model.add(Dense(6, activation='softmax'))  # Izlazni sloj sa 6 klasa

# Kompiliranje modela
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

# Treniranje modela
model.fit(X_train, y_train, epochs=50, batch_size=32)
```

```python
# Evaluacija modela na testnom skupu
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Točnost modela sa PCA komponentama: {accuracy:.4f}')
```

```
Epoch 1/50
4/4 [==============================] - 0s 3ms/step - loss: 9.3782 -
accuracy: 0.3500
Epoch 2/50
4/4 [==============================] - 0s 2ms/step - loss: 5.5328 -
accuracy: 0.4917
Epoch 3/50
4/4 [==============================] - 0s 2ms/step - loss: 3.1194 -
accuracy: 0.5750
Epoch 4/50
4/4 [==============================] - 0s 3ms/step - loss: 1.9497 -
accuracy: 0.6833
Epoch 5/50
4/4 [==============================] - 0s 2ms/step - loss: 1.0661 -
accuracy: 0.8000
Epoch 6/50
4/4 [==============================] - 0s 3ms/step - loss: 0.5559 -
accuracy: 0.8583
Epoch 7/50
4/4 [==============================] - 0s 3ms/step - loss: 0.2990 -
accuracy: 0.9167
Epoch 8/50
4/4 [==============================] - 0s 2ms/step - loss: 0.1530 -
accuracy: 0.9583
Epoch 9/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0869 -
accuracy: 0.9833
Epoch 10/50
4/4 [==============================] - 0s 1ms/step - loss: 0.0498 -
accuracy: 0.9917
Epoch 11/50
4/4 [==============================] - 0s 6ms/step - loss: 0.0335 -
accuracy: 0.9917
Epoch 12/50
4/4 [==============================] - 0s 5ms/step - loss: 0.0212 -
accuracy: 0.9917
Epoch 13/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0142 -
accuracy: 1.0000
Epoch 14/50
4/4 [==============================] - 0s 3ms/step - loss: 0.0117 -
accuracy: 1.0000
Epoch 15/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0070 -
accuracy: 1.0000
Epoch 16/50
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.0062 -
accuracy: 1.0000
Epoch 17/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0051 -
accuracy: 1.0000
Epoch 18/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0045 -
accuracy: 1.0000
Epoch 19/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0040 -
accuracy: 1.0000
Epoch 20/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0037 -
accuracy: 1.0000
Epoch 21/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0035 -
accuracy: 1.0000
Epoch 22/50
4/4 [==============================] - 0s 3ms/step - loss: 0.0032 -
accuracy: 1.0000
Epoch 23/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0030 -
accuracy: 1.0000
Epoch 24/50
4/4 [==============================] - 0s 1ms/step - loss: 0.0029 -
accuracy: 1.0000
Epoch 25/50
4/4 [==============================] - 0s 3ms/step - loss: 0.0028 -
accuracy: 1.0000
Epoch 26/50
4/4 [==============================] - 0s 1ms/step - loss: 0.0026 -
accuracy: 1.0000
Epoch 27/50
4/4 [==============================] - 0s 3ms/step - loss: 0.0025 -
accuracy: 1.0000
Epoch 28/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0024 -
accuracy: 1.0000
Epoch 29/50
4/4 [==============================] - 0s 1ms/step - loss: 0.0023 -
accuracy: 1.0000
Epoch 30/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0023 -
accuracy: 1.0000
Epoch 31/50
4/4 [==============================] - 0s 1ms/step - loss: 0.0022 -
accuracy: 1.0000
Epoch 32/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0021 -
```

```
accuracy: 1.0000
Epoch 33/50
4/4 [==============================] - 0s 1ms/step - loss: 0.0020 -
accuracy: 1.0000
Epoch 34/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0020 -
accuracy: 1.0000
Epoch 35/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0019 -
accuracy: 1.0000
Epoch 36/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0019 -
accuracy: 1.0000
Epoch 37/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0018 -
accuracy: 1.0000
Epoch 38/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0018 -
accuracy: 1.0000
Epoch 39/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0017 -
accuracy: 1.0000
Epoch 40/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0017 -
accuracy: 1.0000
Epoch 41/50
4/4 [==============================] - 0s 1ms/step - loss: 0.0016 -
accuracy: 1.0000
Epoch 42/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0016 -
accuracy: 1.0000
Epoch 43/50
4/4 [==============================] - 0s 1ms/step - loss: 0.0016 -
accuracy: 1.0000
Epoch 44/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0015 -
accuracy: 1.0000
Epoch 45/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0015 -
accuracy: 1.0000
Epoch 46/50
4/4 [==============================] - 0s 1ms/step - loss: 0.0014 -
accuracy: 1.0000
Epoch 47/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0014 -
accuracy: 1.0000
Epoch 48/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0014 -
accuracy: 1.0000
```

```
Epoch 49/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0014 -
accuracy: 1.0000
Epoch 50/50
4/4 [==============================] - 0s 2ms/step - loss: 0.0013 -
accuracy: 1.0000
Točnost modela sa PCA komponentama: 0.6452
```

1.4 Primjena biblioteke "Lazy Predict"

Lazy Predict je Python biblioteka koja olakšava izgradnju i evaluaciju različitih modela strojnog učenja bez potrebe za ručnim podešavanjem parametara. Ova biblioteka omogućuje brzo testiranje više modela i prikazivanje njihove performanse kroz detaljne izvještaje.

```
from lazypredict.Supervised import LazyClassifier

# Podjela podataka
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.3, random_state=0)

# Primjena Lazy Predict na skupu podataka
clf = LazyClassifier(verbose=0, ignore_warnings=True,
custom_metric=None)
models, predictions = clf.fit(X_train, X_test, y_train, y_test)

print(models)
```

```
 97%|█████████▋| 30/31 [01:32<00:07,  7.34s/it]

[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.228127 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1455901
[LightGBM] [Info] Number of data points in the train set: 75, number
of used features: 54675
[LightGBM] [Info] Start training from score -1.752539
[LightGBM] [Info] Start training from score -1.181994
[LightGBM] [Info] Start training from score -2.371578
[LightGBM] [Info] Start training from score -1.678431
[LightGBM] [Info] Start training from score -1.544899
[LightGBM] [Info] Start training from score -3.624341
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
```

```
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf

100%|██████████| 31/31 [02:00<00:00,  3.87s/it]

                                   Accuracy  Balanced Accuracy ROC AUC  F1
Score  \
Model

LogisticRegression                     0.96               0.97    None
0.96
LinearDiscriminantAnalysis             0.92               0.94    None
0.92
BernoulliNB                            0.92               0.93    None
0.92
SGDClassifier                          0.92               0.93    None
0.92
RidgeClassifierCV                      0.95               0.93    None
0.95
RidgeClassifier                        0.95               0.93    None
0.95
ExtraTreesClassifier                   0.93               0.93    None
0.93
Perceptron                             0.89               0.91    None
0.90
LinearSVC                              0.88               0.90    None
0.89
PassiveAggressiveClassifier            0.88               0.90    None
0.89
NearestCentroid                        0.88               0.88    None
0.88
CalibratedClassifierCV                 0.87               0.87    None
0.87
RandomForestClassifier                 0.92               0.87    None
0.91
XGBClassifier                          0.88               0.84    None
0.87
LGBMClassifier                         0.88               0.80    None
0.87
```

BaggingClassifier	0.87	0.79	None	0.85
GaussianNB	0.86	0.77	None	0.83
SVC	0.80	0.73	None	0.77
KNeighborsClassifier	0.80	0.72	None	0.78
DecisionTreeClassifier	0.78	0.71	None	0.76
ExtraTreeClassifier	0.68	0.64	None	0.64
AdaBoostClassifier	0.42	0.36	None	0.32
QuadraticDiscriminantAnalysis	0.17	0.24	None	0.17
LabelSpreading	0.22	0.17	None	0.08
LabelPropagation	0.22	0.17	None	0.08
DummyClassifier	0.24	0.17	None	0.09

Model	Time Taken
LogisticRegression	1.75
LinearDiscriminantAnalysis	2.05
BernoulliNB	0.94
SGDClassifier	1.15
RidgeClassifierCV	0.84
RidgeClassifier	0.77
ExtraTreesClassifier	0.93
Perceptron	1.11
LinearSVC	2.87
PassiveAggressiveClassifier	2.37
NearestCentroid	0.80
CalibratedClassifierCV	5.36
RandomForestClassifier	1.06
XGBClassifier	36.63
LGBMClassifier	27.56
BaggingClassifier	6.25
GaussianNB	1.01
SVC	1.17
KNeighborsClassifier	0.86
DecisionTreeClassifier	1.70
ExtraTreeClassifier	0.68
AdaBoostClassifier	17.66
QuadraticDiscriminantAnalysis	1.10
LabelSpreading	0.91

```
LabelPropagation                        0.87
DummyClassifier                         0.67
```

Analiza rezltata:

- Najbolje Performanse: LogisticRegression postiže najbolje rezultate sa točnošću od 0.96 i vremenskim trajanjem treniranja od 1.75 sekundi.
- Brzi Modeli: BernoulliNB i RidgeClassifier su među najbržima sa treniranjem ispod 1 sekunde, a imaju solidne performanse.
- Varijabilnost u Performansama: KNeighborsClassifier i SVC pokazuju niže performanse u usporedbi s drugim modelima, ali njihova točnost je još uvijek respektabilna.
- Najsloženiji Modeli: XGBClassifier i LGBMClassifier imaju znatno duže vrijeme treniranja, ali njihove performanse nisu najbolje, što sugerira da dodatna kompleksnost nije nužno korisna za vaš skup podataka.
- Izuzetno Niske Performanse: Modeli poput QuadraticDiscriminantAnalysis i LabelSpreading pokazuju znatno niže performanse i možda nisu prikladni za ovaj zadatak.

```
logistic_model = clf.models['LogisticRegression']
print(logistic_model)

# Prikazivanje parametara modela u svrhu usporedbe sa ručno treniranim
modelom
print(logistic_model.get_params())

Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',

Pipeline(steps=[('imputer',

SimpleImputer()),

('scaler',

StandardScaler())]),
                                                 Index(['1007_s_at',
'1053_at', '117_at', '121_at', '1255_g_at', '1294_at',
       '1316_at', '1320_at', '1405_i_at', '1431_at',
       ...
       'AFFX-r2-Ec-bioD-3_at', 'AFFX-r2-Ec-bioD-5_at', 'AFFX-r2-P1-
cre-3_at',
       'AFFX-r2-P1-cre-5_a...
```

```
SimpleImputer(fill_value='missing',

strategy='constant')),

('encoding',

OneHotEncoder(handle_unknown='ignore',

sparse_output=False))]),
                                                  Index([],
dtype='object')),
                                                 ('categorical_high',

Pipeline(steps=[('imputer',

SimpleImputer(fill_value='missing',

strategy='constant')),

('encoding',

OrdinalEncoder())]),
                                                  Index([],
dtype='object'))])),
                ('classifier', LogisticRegression(random_state=42))])
{'memory': None, 'steps': [('preprocessor',
ColumnTransformer(transformers=[('numeric',
                                 Pipeline(steps=[('imputer',
SimpleImputer()),
                                                 ('scaler',
StandardScaler())]),
                                 Index(['1007_s_at', '1053_at',
'117_at', '121_at', '1255_g_at', '1294_at',
       '1316_at', '1320_at', '1405_i_at', '1431_at',
       ...
       'AFFX-r2-Ec-bioD-3_at', 'AFFX-r2-Ec-bioD-5_at', 'AFFX-r2-P1-
cre-3_at',
       'AFFX-r2-P1-cre-5_at', 'AFFX-ThrX-3_at', 'AFFX-ThrX-5...
                                ('categorical_low',
                                 Pipeline(steps=[('imputer',

SimpleImputer(fill_value='missing',

strategy='constant')),
                                                 ('encoding',

OneHotEncoder(handle_unknown='ignore',

sparse_output=False))]),
```

```
                                  Index([], dtype='object')),
                                 ('categorical_high',
                                  Pipeline(steps=[('imputer',

SimpleImputer(fill_value='missing',

strategy='constant')),
                                                  ('encoding',
                                                   OrdinalEncoder())]),
                                  Index([], dtype='object'))])),
('classifier', LogisticRegression(random_state=42))], 'verbose':
False, 'preprocessor': ColumnTransformer(transformers=[('numeric',
                                  Pipeline(steps=[('imputer',
SimpleImputer()),
                                                  ('scaler',
StandardScaler())]),
                                  Index(['1007_s_at', '1053_at',
'117_at', '121_at', '1255_g_at', '1294_at',
       '1316_at', '1320_at', '1405_i_at', '1431_at',
       ...
       'AFFX-r2-Ec-bioD-3_at', 'AFFX-r2-Ec-bioD-5_at', 'AFFX-r2-P1-
cre-3_at',
       'AFFX-r2-P1-cre-5_at', 'AFFX-ThrX-3_at', 'AFFX-ThrX-5...
                                 ('categorical_low',
                                  Pipeline(steps=[('imputer',

SimpleImputer(fill_value='missing',

strategy='constant')),
                                                  ('encoding',

OneHotEncoder(handle_unknown='ignore',

sparse_output=False))]),
                                  Index([], dtype='object')),
                                 ('categorical_high',
                                  Pipeline(steps=[('imputer',

SimpleImputer(fill_value='missing',

strategy='constant')),
                                                  ('encoding',
                                                   OrdinalEncoder())]),
                                  Index([], dtype='object'))]),
'classifier': LogisticRegression(random_state=42),
'preprocessor__force_int_remainder_cols': True,
'preprocessor__n_jobs': None, 'preprocessor__remainder': 'drop',
'preprocessor__sparse_threshold': 0.3,
'preprocessor__transformer_weights': None,
'preprocessor__transformers': [('numeric', Pipeline(steps=[('imputer',
```

```
SimpleImputer()), ('scaler', StandardScaler())]), Index(['1007_s_at',
'1053_at', '117_at', '121_at', '1255_g_at', '1294_at',
       '1316_at', '1320_at', '1405_i_at', '1431_at',
       ...
       'AFFX-r2-Ec-bioD-3_at', 'AFFX-r2-Ec-bioD-5_at', 'AFFX-r2-P1-
cre-3_at',
       'AFFX-r2-P1-cre-5_at', 'AFFX-ThrX-3_at', 'AFFX-ThrX-5_at',
       'AFFX-ThrX-M_at', 'AFFX-TrpnX-3_at', 'AFFX-TrpnX-5_at',
       'AFFX-TrpnX-M_at'],
      dtype='object', length=54675)), ('categorical_low',
Pipeline(steps=[('imputer',
                 SimpleImputer(fill_value='missing',
strategy='constant')),
                ('encoding',
                 OneHotEncoder(handle_unknown='ignore',
sparse_output=False))]), Index([], dtype='object')),
('categorical_high', Pipeline(steps=[('imputer',
                 SimpleImputer(fill_value='missing',
strategy='constant')),
                ('encoding', OrdinalEncoder())]), Index([],
dtype='object'))], 'preprocessor__verbose': False,
'preprocessor__verbose_feature_names_out': True,
'preprocessor__numeric': Pipeline(steps=[('imputer', SimpleImputer()),
('scaler', StandardScaler())]), 'preprocessor__categorical_low':
Pipeline(steps=[('imputer',
                 SimpleImputer(fill_value='missing',
strategy='constant')),
                ('encoding',
                 OneHotEncoder(handle_unknown='ignore',
sparse_output=False))]), 'preprocessor__categorical_high':
Pipeline(steps=[('imputer',
                 SimpleImputer(fill_value='missing',
strategy='constant')),
                ('encoding', OrdinalEncoder())]),
'preprocessor__numeric__memory': None, 'preprocessor__numeric__steps':
[('imputer', SimpleImputer()), ('scaler', StandardScaler())],
'preprocessor__numeric__verbose': False,
'preprocessor__numeric__imputer': SimpleImputer(),
'preprocessor__numeric__scaler': StandardScaler(),
'preprocessor__numeric__imputer__add_indicator': False,
'preprocessor__numeric__imputer__copy': True,
'preprocessor__numeric__imputer__fill_value': None,
'preprocessor__numeric__imputer__keep_empty_features': False,
'preprocessor__numeric__imputer__missing_values': nan,
'preprocessor__numeric__imputer__strategy': 'mean',
'preprocessor__numeric__scaler__copy': True,
'preprocessor__numeric__scaler__with_mean': True,
'preprocessor__numeric__scaler__with_std': True,
'preprocessor__categorical_low__memory': None,
```

```
'preprocessor__categorical_low__steps': [('imputer',
SimpleImputer(fill_value='missing', strategy='constant')),
('encoding', OneHotEncoder(handle_unknown='ignore',
sparse_output=False))], 'preprocessor__categorical_low__verbose':
False, 'preprocessor__categorical_low__imputer':
SimpleImputer(fill_value='missing', strategy='constant'),
'preprocessor__categorical_low__encoding':
OneHotEncoder(handle_unknown='ignore', sparse_output=False),
'preprocessor__categorical_low__imputer__add_indicator': False,
'preprocessor__categorical_low__imputer__copy': True,
'preprocessor__categorical_low__imputer__fill_value': 'missing',
'preprocessor__categorical_low__imputer__keep_empty_features': False,
'preprocessor__categorical_low__imputer__missing_values': nan,
'preprocessor__categorical_low__imputer__strategy': 'constant',
'preprocessor__categorical_low__encoding__categories': 'auto',
'preprocessor__categorical_low__encoding__drop': None,
'preprocessor__categorical_low__encoding__dtype': <class
'numpy.float64'>,
'preprocessor__categorical_low__encoding__feature_name_combiner':
'concat', 'preprocessor__categorical_low__encoding__handle_unknown':
'ignore', 'preprocessor__categorical_low__encoding__max_categories':
None, 'preprocessor__categorical_low__encoding__min_frequency': None,
'preprocessor__categorical_low__encoding__sparse_output': False,
'preprocessor__categorical_high__memory': None,
'preprocessor__categorical_high__steps': [('imputer',
SimpleImputer(fill_value='missing', strategy='constant')),
('encoding', OrdinalEncoder())],
'preprocessor__categorical_high__verbose': False,
'preprocessor__categorical_high__imputer':
SimpleImputer(fill_value='missing', strategy='constant'),
'preprocessor__categorical_high__encoding': OrdinalEncoder(),
'preprocessor__categorical_high__imputer__add_indicator': False,
'preprocessor__categorical_high__imputer__copy': True,
'preprocessor__categorical_high__imputer__fill_value': 'missing',
'preprocessor__categorical_high__imputer__keep_empty_features': False,
'preprocessor__categorical_high__imputer__missing_values': nan,
'preprocessor__categorical_high__imputer__strategy': 'constant',
'preprocessor__categorical_high__encoding__categories': 'auto',
'preprocessor__categorical_high__encoding__dtype': <class
'numpy.float64'>,
'preprocessor__categorical_high__encoding__encoded_missing_value':
nan, 'preprocessor__categorical_high__encoding__handle_unknown':
'error', 'preprocessor__categorical_high__encoding__max_categories':
None, 'preprocessor__categorical_high__encoding__min_frequency': None,
'preprocessor__categorical_high__encoding__unknown_value': None,
'classifier__C': 1.0, 'classifier__class_weight': None,
'classifier__dual': False, 'classifier__fit_intercept': True,
'classifier__intercept_scaling': 1, 'classifier__l1_ratio': None,
'classifier__max_iter': 100, 'classifier__multi_class': 'deprecated',
```

```
'classifier__n_jobs': None, 'classifier__penalty': 'l2',
'classifier__random_state': 42, 'classifier__solver': 'lbfgs',
'classifier__tol': 0.0001, 'classifier__verbose': 0,
'classifier__warm_start': False}
```

Razlika između rezultata LazyPredict i ručne implementacije logističke regresije često proizlazi iz dodatnih automatskih koraka koje LazyPredict poduzima, kao što su imputacija nedostajućih vrijednosti, skaliranje numeričkih podataka i enkodiranje kategorijskih podataka. Također, LazyPredict možda koristi unaprijed definirane postavke ili dodatne optimizacije koje nisu eksplicitno navedene u ručnoj implementaciji. Ove razlike u obradi podataka i podešavanju hiperparametara mogu značajno utjecati na performanse modela, objašnjavajući bolji rezultat LazyPredict biblioteke.

1.5 Primjena XGBoost modela

XGBoost, poznat i kao eXtreme Gradient Boosting, je napredna implementacija algoritma gradient boosting. Funkcionira tako što koristi postupak zvan boosting, gdje se nekoliko slabih modela, najčešće jednostavnih stabala odlučivanja, trenira sekvencijalno. Svaki model nastoji ispraviti greške prethodnog modela.

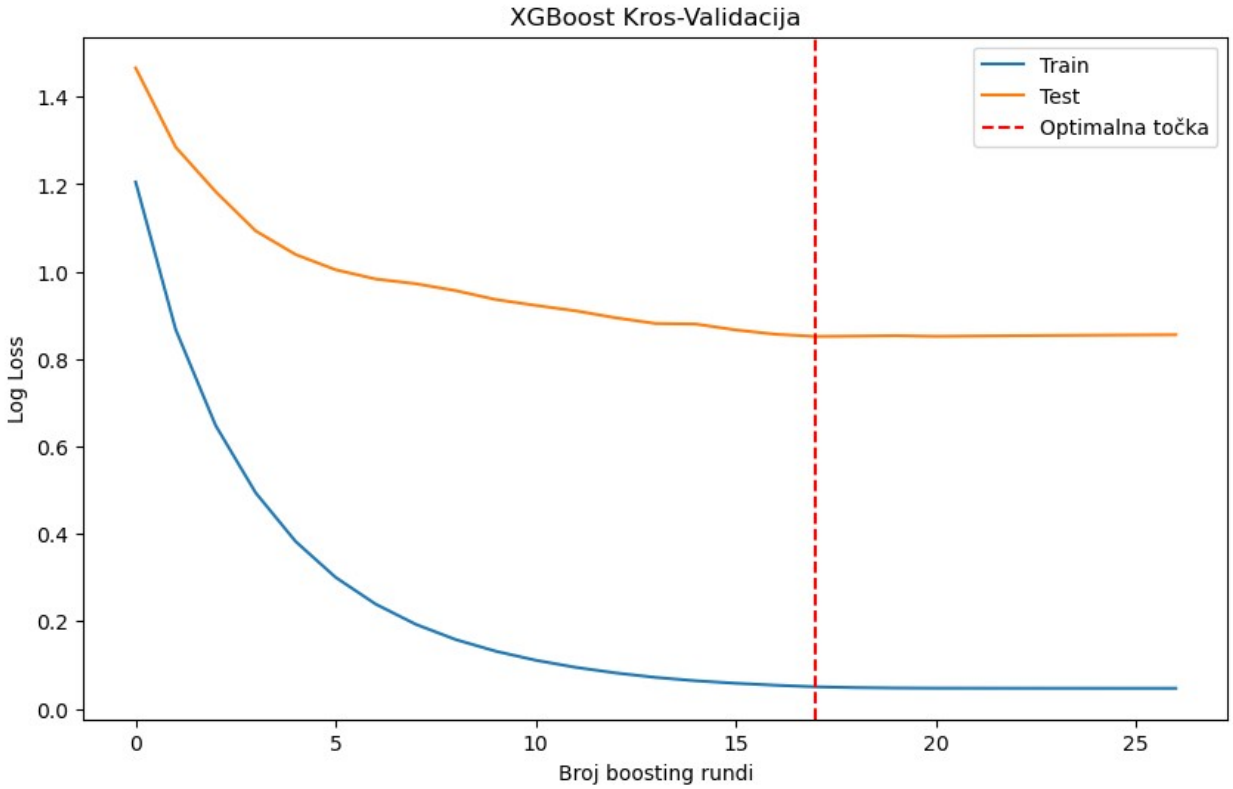
Algoritam koristi gradient descent optimizaciju za minimiziranje funkcije gubitka. To znači da se modeli prilagođavaju na osnovi gradijentnih grešaka, odnosno pokušavaju minimizirati razliku između stvarnih vrijednosti i predikcija.

XGBoost uključuje i regularizaciju kako bi se spriječilo preprilagođavanje, odnosno overfitting. Regularizacija pomaže u održavanju balansa između modela koji je prekompleksan i onog koji je prejednostavan.

Također je poznat po svojoj brzini i učinkovitosti, koristeći unutarnje paralelizme, optimizirano računarstvo i sposobnost rukovanja s velikim količinama podataka.

```
from lazypredict.Supervised import LazyClassifier
import xgboost as xgb
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np

# Podjela podataka
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.5, random_state=0)

# Kreiranje DMatrix-a za XGBoost
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Postavljanje parametara za XGBoost
params = {
    'objective': 'multi:softmax',  # ili 'multi:softprob' za
vjerojatnosti
```

```
    'num_class': len(set(y_encoded)),
    'max_depth': 6,
    'eta': 0.3,
    'seed': 0
}

# Kros-validacija za pronalazak optimalne točke
cv_results = xgb.cv(dtrain=dtrain, params=params, nfold=5,
num_boost_round=27, metrics="mlogloss", as_pandas=True, seed=0)
optimal_num_boost_rounds = cv_results['test-mlogloss-mean'].argmin()

# Treniranje modela
bst = xgb.train(params, dtrain,
num_boost_round=optimal_num_boost_rounds)

# Predikcija na testnom skupu
y_pred = bst.predict(dtest)
accuracy = accuracy_score(y_test, y_pred)

print(f'Točnost modela: {accuracy:.2f}')
print(f'Optimalan broj boosting rundi: {optimal_num_boost_rounds}')

# Crtanje grafa kros-validacije
plt.figure(figsize=(10, 6))
plt.plot(cv_results['train-mlogloss-mean'], label='Train')
plt.plot(cv_results['test-mlogloss-mean'], label='Test')
plt.axvline(x=optimal_num_boost_rounds, color='r', linestyle='--',
label='Optimalna točka')
plt.xlabel('Broj boosting rundi')
plt.ylabel('Log Loss')
plt.title('XGBoost Kros-Validacija')
plt.legend()
plt.show()

Točnost modela: 0.88
Optimalan broj boosting rundi: 17
```



Graf kros validacije pokazuje trenutak pojavljivanja konvergencije na testnom skupu. Zbog sofisticarnog mehanizma primjene kros validacije u funkciji cv() ne uočavamo karakteristični skok na testnom skupu već njegovo svojstvo konvergencije. Model XGBoost-a je toliko dobro regulariziran da uopće nije povećao pogrešku na testnom skupu Opcionalno postoji parametar early_stopping_rounds u xgb.cv() funkciji koji pazi da se model ne vrti bez veze u zoni konvergencije. Npr. early_stopping_rounds=7 znači da će trening biti zaustavljen ako performanse modela (mjereno prema metrikama kao što je gubitak) ne pokažu poboljšanje nakon 7 uzastopnih rundi.

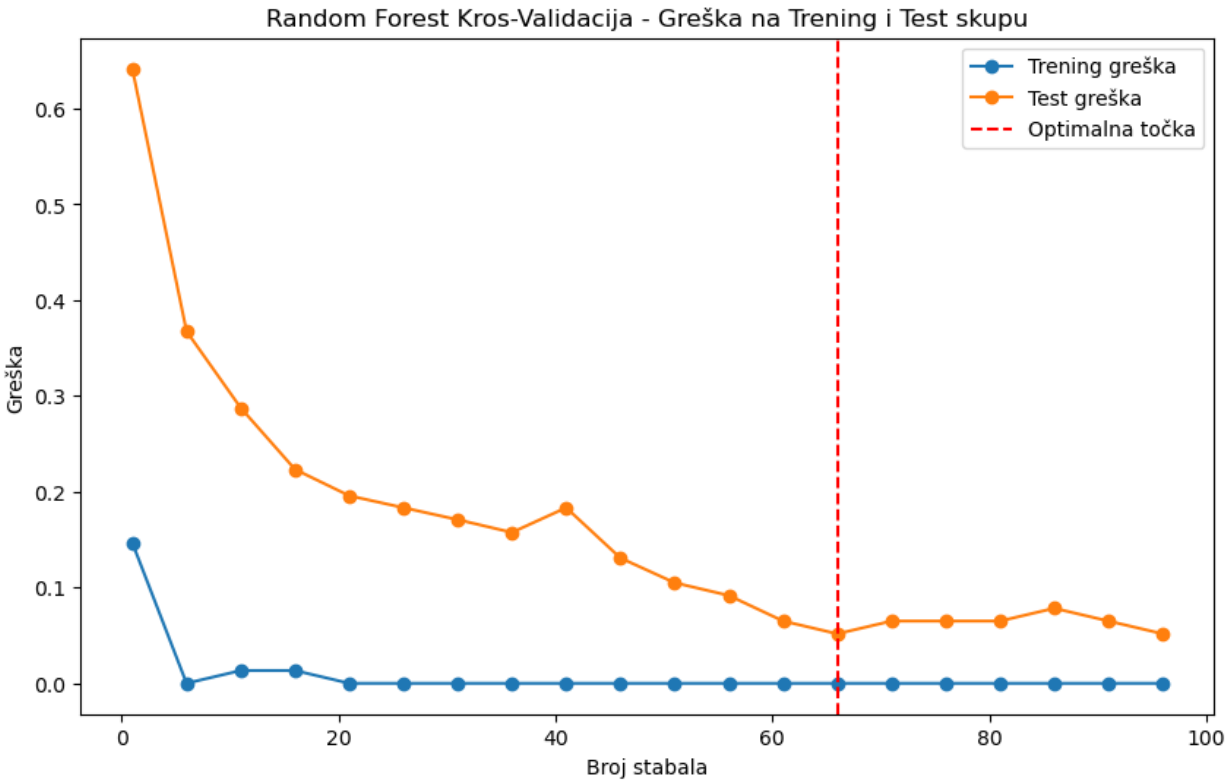
1.6 Random forest

Random forest je ansambl metoda koja koristi više odluka stabala za klasifikaciju ili regresiju. Svako stablo u šumi trenira se na različitim podskupovima podataka s nasumičnim uzorkovanjem, a konačna odluka se donosi većinskim glasanjem ili prosjekom rezultata svih stabala. Ova metoda poboljšava preciznost i smanjuje preprilagođavanje (overfitting) usporedbom s pojedinačnim stablom. Random forest je robusna i fleksibilna tehnika koja može raditi s različitim vrstama podataka i lako se nositi s velikim skupovima podataka.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np

# Podjela podataka
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.5, random_state=0)

# Parametri za Random Forest
num_trees = range(1, 101, 5)  # Ispitivanje broja stabala od 1 do 100
u koracima od 5
train_errors = []
test_errors = []

# Kros-validacija za različite brojeve stabala
for n in num_trees:
    rf = RandomForestClassifier(n_estimators=n, random_state=0)
    rf.fit(X_train, y_train)

    # Greška na trening skupu
    y_train_pred = rf.predict(X_train)
    train_error = 1 - accuracy_score(y_train, y_train_pred)
    train_errors.append(train_error)

    # Kros-validacija na testnom skupu
    test_error = 1 - cross_val_score(rf, X_test, y_test, cv=5,
scoring='accuracy').mean()
    test_errors.append(test_error)

# Pronalaženje optimalnog broja stabala (minimum greške na testnom
skupu)
optimal_trees = num_trees[np.argmin(test_errors)]

# Treniranje konačnog modela sa optimalnim brojem stabala
rf_final = RandomForestClassifier(n_estimators=optimal_trees,
random_state=0)
rf_final.fit(X_train, y_train)

# Predikcija na testnom skupu
y_pred = rf_final.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f'Točnost modela: {accuracy:.2f}')
print(f'Optimalan broj stabala: {optimal_trees}')

# Crtanje grafa kros-validacije
plt.figure(figsize=(10, 6))
plt.plot(num_trees, train_errors, marker='o', label='Trening greška')
plt.plot(num_trees, test_errors, marker='o', label='Test greška')
plt.axvline(x=optimal_trees, color='r', linestyle='--',
label='Optimalna točka')
plt.xlabel('Broj stabala')
plt.ylabel('Greška')
plt.title('Random Forest Kros-Validacija - Greška na Trening i Test
skupu')
```

```
plt.legend()
plt.show()

Točnost modela: 0.91
Optimalan broj stabala: 66
```



Cross-validation graf prikazuje greške modela na trening i test skupovima za različite brojeve stabala. Ovo pomaže procijeniti kako se model ponaša s različitim brojem stabala.

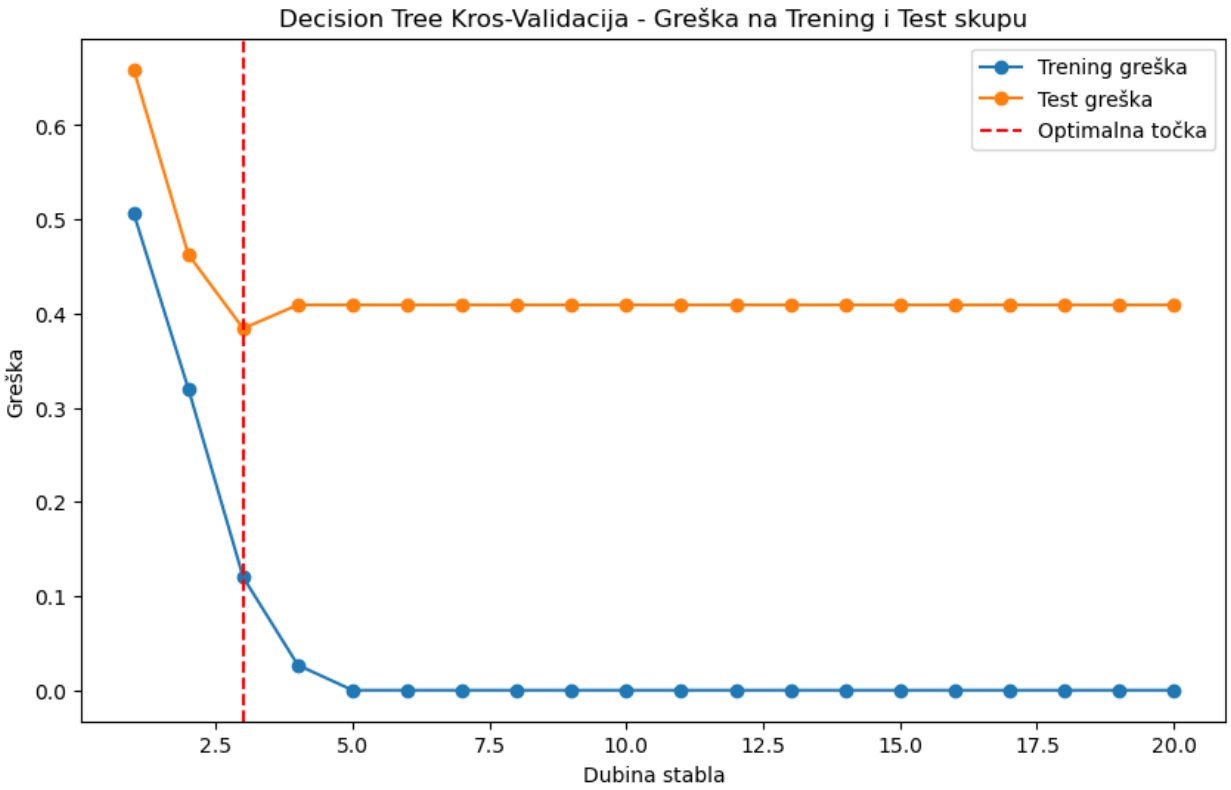
1.7 DecisionTree

Decision Tree je model strojnog učenja koji koristi strukturu stabla za donošenje odluka. Svaki unutarnji čvor predstavlja pitanje ili uvjet vezan za značajku podataka, dok svaki list (završni čvor) predstavlja ishod ili odluku. Kroz niz binarnih podjela, model iterativno dijeli podatke na manje podskupove dok se ne postigne najbolje moguće razdvajanje. Svaka podjela maksimizira informativnost. Na kraju, put od korijenskog čvora do lista pruža predikciju za novi uzorak. Stabla odluke su laka za interpretirati i vizualizirati, ali su sklona preprilagođavanju ako nisu pravilno ograničena. Poboljšanje koncepta stabala odluke predstavljaju random forests koja su i našem slučaju ostvarila bolju točnost.

```
from sklearn.tree import DecisionTreeClassifier

# Podjela podataka
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.5, random_state=0)
```

```
# Parametri za Decision Tree
max_depths = range(1, 21)  # Ispitivanje dubine stabla od 1 do 20
train_errors = []
test_errors = []

# Kros-validacija za različite dubine stabla
for depth in max_depths:
    dt = DecisionTreeClassifier(max_depth=depth, random_state=0)
    dt.fit(X_train, y_train)

    # Greška na trening skupu
    y_train_pred = dt.predict(X_train)
    train_error = 1 - accuracy_score(y_train, y_train_pred)
    train_errors.append(train_error)

    # Kros-validacija na testnom skupu
    test_error = 1 - cross_val_score(dt, X_test, y_test, cv=5,
scoring='accuracy').mean()
    test_errors.append(test_error)

# Pronalaženje optimalne dubine stabla (minimum greške na testnom
skupu)
optimal_depth = max_depths[np.argmin(test_errors)]

# Treniranje konačnog modela sa optimalnom dubinom stabla
dt_final = DecisionTreeClassifier(max_depth=optimal_depth,
random_state=0)
dt_final.fit(X_train, y_train)

# Predikcija na testnom skupu
y_pred = dt_final.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f'Točnost modela: {accuracy:.2f}')
print(f'Optimalna dubina stabla: {optimal_depth}')

# Crtanje grafa kros-validacije
plt.figure(figsize=(10, 6))
plt.plot(max_depths, train_errors, marker='o', label='Trening greška')
plt.plot(max_depths, test_errors, marker='o', label='Test greška')
plt.axvline(x=optimal_depth, color='r', linestyle='--',
label='Optimalna točka')
plt.xlabel('Dubina stabla')
plt.ylabel('Greška')
plt.title('Decision Tree Kros-Validacija - Greška na Trening i Test
skupu')
plt.legend()
plt.show()
```

```
Točnost modela: 0.71
Optimalna dubina stabla: 3
```



2. Klasifikacija leukemije na temelju podataka o ekspresiji gena

Struktura datasetova za klasifikaciju leukemije:

Potrebno je malo manipulacije podacima: iz "data_set_ALL_AML_train.csv" datoteke vadim stupce s oznakama prisutnosti gena (A,P ili M) i formiram novu tablicu (38 redaka/7129 stupaca) tako da svaki izvučeni stupac postane redak u toj tablici. Trebamo dobiti sljedeću tablicu: Tablica 5. "train1.csv" (38 redaka/7129 stupaca)

```
# Učitaj osnovne biblioteke...
import sklearn
import matplotlib.pyplot as plt
# %pylab inline
%matplotlib inline
import numpy as np
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

import pandas as pd

# Funkcija za enkodiranje oznaka
def encode_labels(label):
    if label == 'A':
        return -1
    elif label == 'P':
        return 1
    elif label == 'M':
        return 0
    else:
        return label

file_path = 'leukemia_data/data_set_ALL_AML_train.csv'  # zamijeniti s
točnom putanjom do datoteke
data = pd.read_csv(file_path)

# Brojanje ukupnog broja stupaca u podacima
total_columns = data.shape[1]
print(f"Ukupan broj stupaca u originalnoj csv datoteci:
{total_columns}")

# Ispisivanje prvog naslovnog redka CSV datoteke
print("Naslovni redak CSV datoteke:")
print(data.columns.values)

# Brojanje ukupnog broja redaka u podacima
total_r = data.shape[0]
print(f"Ukupan broj redaka u originalnoj csv datoteci: {total_r}")

# Preimenovanje stupca "call" u "call.0"
data.columns = ['call.0' if col == 'call' else col for col in
data.columns]

# Generiranje i sortiranje stupaca prema zadanim uputama
sorted_columns = ['Gene Description', 'Gene Accession Number']
sorted_data = data[sorted_columns]  # Stvaranje nove tablice s
osnovnim stupcima

for i in range(1, 39):
    num_str = str(i)
    if num_str in data.columns:
        num_index = data.columns.get_loc(num_str)  # Dohvaćanje
indeksa stupca num_str
        call_col = data.columns[num_index + 1]     # Dohvaćanje stupca
odmah desno
```

```
        sorted_data = pd.concat([sorted_data, data[[num_str,
call_col]]], axis=1)

# Ispisivanje naslovnog redka nove tablice sorted
print("\nNaslovni redak nove tablice sorted:")
print(sorted_data.columns.values)

# Izdvajanje stupaca 4., 6., 8. itd.
columns_to_extract = sorted_data.columns[3::2]
extracted_columns = sorted_data[columns_to_extract]

# Pretvaranje stupaca u redove nove tablice bez sortiranja
train1_x = extracted_columns.transpose()

print("\n")
print("Izgled nove tablice:")
# Ispis nove tablice
print(train1_x)

train1_x = train1_x.applymap(encode_labels)
print("izgled tablice train1_x nakon enkodiranja oznaka:")
print(train1_x)
```

```
Ukupan broj stupaca u originalnoj csv datoteci: 78
Naslovni redak CSV datoteke:
['Gene Description' 'Gene Accession Number' '1' 'call' '2' 'call.1'
'3'
 'call.2' '4' 'call.3' '5' 'call.4' '6' 'call.5' '7' 'call.6' '8'
'call.7'
 '9' 'call.8' '10' 'call.9' '11' 'call.10' '12' 'call.11' '13'
'call.12'
 '14' 'call.13' '15' 'call.14' '16' 'call.15' '17' 'call.16' '18'
 'call.17' '19' 'call.18' '20' 'call.19' '21' 'call.20' '22' 'call.21'
 '23' 'call.22' '24' 'call.23' '25' 'call.24' '26' 'call.25' '27'
 'call.26' '34' 'call.27' '35' 'call.28' '36' 'call.29' '37' 'call.30'
 '38' 'call.31' '28' 'call.32' '29' 'call.33' '30' 'call.34' '31'
 'call.35' '32' 'call.36' '33' 'call.37']
Ukupan broj redaka u originalnoj csv datoteci: 7129

Naslovni redak nove tablice sorted:
['Gene Description' 'Gene Accession Number' '1' 'call.0' '2' 'call.1'
'3'
 'call.2' '4' 'call.3' '5' 'call.4' '6' 'call.5' '7' 'call.6' '8'
'call.7'
 '9' 'call.8' '10' 'call.9' '11' 'call.10' '12' 'call.11' '13'
'call.12'
```

```
 '14' 'call.13' '15' 'call.14' '16' 'call.15' '17' 'call.16' '18'
 'call.17' '19' 'call.18' '20' 'call.19' '21' 'call.20' '22' 'call.21'
 '23' 'call.22' '24' 'call.23' '25' 'call.24' '26' 'call.25' '27'
 'call.26' '28' 'call.32' '29' 'call.33' '30' 'call.34' '31' 'call.35'
 '32' 'call.36' '33' 'call.37' '34' 'call.27' '35' 'call.28' '36'
 'call.29' '37' 'call.30' '38' 'call.31']


Izgled nove tablice:
        0    1    2    3    4    5    6    7    8    9     ... 7119
7120 7121  \
call.0     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.1     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.2     A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.3     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.4     A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.5     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.6     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.7     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.8     A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.9     A    A    A    M    A    A    A    A    A    A  ...    A
P    A
call.10    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.11    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.12    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.13    A    A    A    M    A    A    A    A    A    A  ...    P
A    A
call.14    A    A    A    A    A    A    A    A    A    A  ...    A
P    A
call.15    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.16    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.17    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.18    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
```

```
call.19    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.20    A    A    A    A    A    A    A    A    A    A  ...    A
P    A
call.21    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.22    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.23    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.24    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.25    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.26    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.32    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.33    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.34    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.35    A    A    A    P    A    A    A    A    A    A  ...    A
A    A
call.36    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.37    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.27    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.28    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.29    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.30    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.31    A    A    A    P    A    A    A    A    A    A  ...    A
A    A

         7122 7123 7124 7125 7126 7127 7128
call.0     P    A    A    A    A    A    A
call.1     A    A    A    A    A    A    A
call.2     A    A    A    P    A    A    A
call.3     A    A    A    A    A    A    A
call.4     P    A    A    A    A    A    A
call.5     A    A    A    A    A    A    A
call.6     A    A    A    A    A    P    A
call.7     A    A    A    P    A    P    A
call.8     A    A    A    A    P    A    A
```

```
call.9     A    A    A    P    A    M    A
call.10    A    A    A    A    A    A    A
call.11    A    A    A    A    A    A    A
call.12    P    A    A    A    A    A    A
call.13    A    A    A    A    A    A    A
call.14    P    A    A    A    A    A    A
call.15    A    A    A    P    A    P    A
call.16    A    A    A    P    A    A    A
call.17    P    A    A    P    A    A    A
call.18    P    A    A    A    A    A    A
call.19    A    A    A    A    A    P    A
call.20    A    A    A    A    A    A    A
call.21    P    A    A    A    A    M    A
call.22    A    A    A    A    A    M    A
call.23    A    A    A    A    A    A    A
call.24    P    A    A    A    A    P    A
call.25    P    A    A    P    A    A    A
call.26    A    A    A    A    A    A    A
call.32    P    A    A    A    A    P    A
call.33    P    A    A    A    A    P    A
call.34    P    A    A    A    A    A    A
call.35    A    A    A    A    A    A    A
call.36    A    A    A    A    A    A    A
call.37    A    A    A    A    A    P    A
call.27    A    A    A    A    A    M    A
call.28    A    A    A    A    A    P    A
call.29    P    A    A    A    A    A    A
call.30    A    A    A    A    A    P    A
call.31    A    A    A    A    A    P    A

[38 rows x 7129 columns]
izgled tablice train1_x nakon enkodiranja oznaka:
         0     1     2     3     4     5     6     7     8
9     ...  \
call.0     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.1     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.2     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.3     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.4     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.5     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.6     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.7     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
```

```
1  ...
call.8     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.9     -1    -1    -1     0    -1    -1    -1    -1    -1    -
1  ...
call.10    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.11    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.12    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.13    -1    -1    -1     0    -1    -1    -1    -1    -1    -
1  ...
call.14    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.15    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.16    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.17    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.18    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.19    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.20    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.21    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.22    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.23    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.24    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.25    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.26    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.32    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.33    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.34    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.35    -1    -1    -1     1    -1    -1    -1    -1    -1    -
1  ...
call.36    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
```

```
call.37    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.27    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.28    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.29    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.30    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.31    -1    -1    -1     1    -1    -1    -1    -1    -1    -
1  ...

          7119  7120  7121  7122  7123  7124  7125  7126  7127  7128
call.0      -1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.1      -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.2       1    -1    -1    -1    -1    -1     1    -1    -1    -1
call.3      -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.4       1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.5      -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.6      -1    -1    -1    -1    -1    -1    -1    -1     1    -1
call.7      -1    -1    -1    -1    -1    -1     1    -1     1    -1
call.8       1    -1    -1    -1    -1    -1    -1     1    -1    -1
call.9      -1     1    -1    -1    -1    -1     1    -1     0    -1
call.10      1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.11     -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.12      1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.13      1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.14     -1     1    -1     1    -1    -1    -1    -1    -1    -1
call.15     -1    -1    -1    -1    -1    -1     1    -1     1    -1
call.16      1    -1    -1    -1    -1    -1     1    -1    -1    -1
call.17      1    -1    -1     1    -1    -1     1    -1    -1    -1
call.18     -1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.19      1    -1    -1    -1    -1    -1    -1    -1     1    -1
call.20     -1     1    -1    -1    -1    -1    -1    -1    -1    -1
call.21      1    -1    -1     1    -1    -1    -1    -1     0    -1
call.22     -1    -1    -1    -1    -1    -1    -1    -1     0    -1
call.23     -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.24     -1    -1    -1     1    -1    -1    -1    -1     1    -1
call.25      1    -1    -1     1    -1    -1     1    -1    -1    -1
call.26     -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.32      1    -1    -1     1    -1    -1    -1    -1     1    -1
call.33     -1    -1    -1     1    -1    -1    -1    -1     1    -1
call.34      1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.35     -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.36      1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.37     -1    -1    -1    -1    -1    -1    -1    -1     1    -1
call.27     -1    -1    -1    -1    -1    -1    -1    -1     0    -1
call.28     -1    -1    -1    -1    -1    -1    -1    -1     1    -1
```

```
call.29     1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.30    -1    -1    -1    -1    -1    -1    -1    -1     1    -1
call.31    -1    -1    -1    -1    -1    -1    -1    -1     1    -1

[38 rows x 7129 columns]
```

Postupak ponovimo ali izvlačimo numeričke stupce sa informacijama o razini ekspresije gena. Dobivamo tablicu: Tablica 6. "train2.csv" (38 redaka/7129 stupaca)

```
# Izdvajanje stupaca 4., 6., 8. itd.
columns_to_extract = sorted_data.columns[2::2]
extracted_columns = sorted_data[columns_to_extract]

# Pretvaranje stupaca u redove nove tablice bez sortiranja
train2_x = extracted_columns.transpose()

print("\n")
print("Izgled nove tablice:")
# Ispis nove tablice
print(train2_x)
```

```
Izgled nove tablice:
    0     1     2     3     4     5     6     7     8     9     ...
7119  \
1   -214  -153   -58    88  -295  -558   199  -176   252   206  ...
185
2   -139   -73    -1   283  -264  -400  -330  -168   101    74  ...
169
3    -76   -49  -307   309  -376  -650    33  -367   206  -215  ...
315
4   -135  -114   265    12  -419  -585   158  -253    49    31  ...
240
5   -106  -125   -76   168  -230  -284     4  -122    70   252  ...
156
6   -138   -85   215    71  -272  -558    67  -186    87   193  ...
115
7    -72  -144   238    55  -399  -551   131  -179   126   -20  ...
30
8   -413  -260     7    -2  -541  -790  -275  -463    70  -169  ...
289
9      5  -127   106   268  -210  -535     0  -174    24   506  ...
356
10   -88  -105    42   219  -178  -246   328  -148   177   183  ...
42
11  -165  -155   -71    82  -163  -430   100  -109    56   350  ...
185
12   -67   -93    84    25  -179  -323  -135  -127    -2   -66  ...
48
```

```
13   -92  -119   -31   173  -233  -227   -49   -62    13   230  ...
213
14  -113  -147  -118   243  -127  -398  -249  -228   -37   113  ...
267
15  -107   -72  -126   149  -205  -284  -166  -185     1   -23  ...
120
16  -117  -219   -50   257  -218  -402   228  -147    65    67  ...
79
17  -476  -213   -18   301  -403  -394   -42  -144    98   173  ...
241
18   -81  -150  -119    78  -152  -340   -36  -141    96   -55  ...
186
19   -44   -51   100   207  -146  -221    83  -198    34   -20  ...
318
20    17  -229    79   218  -262  -404   326  -201     6   469  ...
225
21  -144  -199  -157   132  -151  -347  -118   -24   126  -201  ...
103
22  -247   -90  -168   -24  -308  -571  -170  -224   124  -117  ...
158
23   -74  -321   -11   -36  -317  -499  -138  -119   115   -17  ...
129
24  -120  -263  -114   255  -342  -396  -412  -153   184  -162  ...
176
25   -81  -150   -85   316  -418  -461   -66  -184   164    -5  ...
138
26  -112  -233   -78    54  -244  -275  -479  -108   136   -86  ...
190
27  -273  -327   -76    81  -439  -616   419  -251   165   350  ...
120
28    -4  -116  -125   241  -191  -411   -31  -240   150    24  ...
173
29    15  -114     2   193   -51  -155    29  -105    42   524  ...
173
30  -318  -192   -95   312  -139  -344   324  -237   105   167  ...
225
31   -32   -49    49   230  -367  -508  -349  -194    34   -56  ...
36
32  -124   -79   -37   330  -188  -423   -31  -223   -82   176  ...
348
33  -135  -186   -70   337  -407  -566  -141  -315   206   321  ...
209
34   -20  -207   -50   101  -369  -529    14  -365   153    29  ...
260
35     7  -100   -57   132  -377  -478  -351  -290   283   247  ...
93
36  -213  -252   136   318  -209  -557    40  -243   119  -131  ...
234
37   -25   -20   124   325  -396  -464  -221  -390    -1   358  ...
```

```
146
38   -72  -139    -1   392  -324  -510  -350  -202   249   561  ...
103

    7120  7121  7122  7123  7124  7125  7126  7127  7128
1    511  -125   389   -37   793   329    36   191   -37
2    837   -36   442   -17   782   295    11    76   -14
3   1199    33   168    52  1138   777    41   228   -41
4    835   218   174  -110   627   170   -50   126   -91
5    649    57   504   -26   250   314    14    56   -25
6   1221   -76   172   -74   645   341    26   193   -53
7    819  -178   151   -18  1140   482    10   369   -42
8    629   -86   302    23  1799   446    59   781    20
9    980     6   177   -12   758   385   115   244   -39
10   986    26   101    21   570   359     9   171     7
11   642    32   137   -81   672   208    25   116   -62
12   224    60   194   -10   291    41     8    -2   -80
13   583     3   530   -39   696   302    24    74   -11
14   440    52   229    -4   431   269     8   163   -22
15   722    20   332    -5   195    59    31   116   -18
16   631   -26   455   -62   736   445    42   246   -43
17  1215   127   255    50  1701  1109    61   526   -83
18   573   -57   694   -19   636   205    17   127   -13
19   397   -48  1939   -18   538    90   -50   333   -24
20  1020  -110   209   -51  1435   255    53   545   -16
21   595   -12    36    26   208   113    -8    22   -22
22   402    57   253   -52  1010   405    19   270   -27
23  1058   140   176   -22   617   336     9   243    36
24   725    13   249     1   646   391    81   203   -94
25   392     8   506    24  1034    69    24   807   -41
26   678    77  2527   -36   838   313    21   145   -19
27   816    45    62   -71   583   677    -1   208    10
28   755   -23   573    42   987   279    22   662   -46
29   492    54   277   -13   279    51     6  2484    -2
30   737    63   472    33   737   227    -9   371   -31
31   592    57   215   -22   588   361   -26   133   -32
32   938   -15   433    -2  1170   284    39   298    -3
33   634   -58   375   -23  2315   250   -12   790   -10
34  1009   -55   139   -57   834   557   -12   335   -65
35   336   -45   170    12   752   295    28  1558   -67
36  1653    67   486   -88  1293   342    26   246    23
37   486   -32   334    35  1733   304    12  3193   -33
38  1121   102   330  -112  1567   627    21  2520     0

[38 rows x 7129 columns]
```

Ponovimo to i sa skupom za testiranje i dobivamo skupove test1_x i test2_x. Također očekivane izlaze zapisane u "actual.csv" podijelimo sukladno podijeli na skup za učenje i treniranje: Prvih 38 ide kao oznake za train1_x i train2_x ulaze. preostalih 34 ide za oznake test1_x (A,P,M - kategorički podaci) i test2.x (numerički podaci) skupovima.

```
import pandas as pd

# Učitavanje podataka iz CSV datoteke
file_path = 'leukemia_data/data_set_ALL_AML_independent.csv'
data = pd.read_csv(file_path)

# Brojanje ukupnog broja stupaca u podacima
total_columns = data.shape[1]
print(f"Ukupan broj stupaca u originalnoj csv datoteci:
{total_columns}")

# Brojanje ukupnog broja redaka u podacima
total_r = data.shape[0]
print(f"Ukupan broj redaka u originalnoj csv datoteci: {total_r}")

# Ispisivanje prvog naslovnog redka CSV datoteke
print("Naslovni redak originalne CSV datoteke:")
print(data.columns.values)

# Preimenovanje stupca "call" u "call.0"
data.columns = ['call.0' if col == 'call' else col for col in
data.columns]

# Generiranje i sortiranje stupaca prema zadanim uputama
sorted_columns = ['Gene Description', 'Gene Accession Number']
sorted_data = data[sorted_columns]  # Stvaranje nove tablice s
osnovnim stupcima

for i in range(39, 73):
    num_str = str(i)
    if num_str in data.columns:
        num_index = data.columns.get_loc(num_str)  # Dohvaćanje
indeksa stupca num_str
        call_col = data.columns[num_index + 1]     # Dohvaćanje stupca
odmah desno
        sorted_data = pd.concat([sorted_data, data[[num_str,
call_col]]], axis=1)

# Ispisivanje naslovnog redka nove tablice sorted
print("\nNaslovni redak nove tablice sorted:")
print(sorted_data.columns.values)

# Izdvajanje stupaca 4., 6., 8. itd.
columns_to_extract = sorted_data.columns[3::2]
extracted_columns = sorted_data[columns_to_extract]

# Pretvaranje stupaca u redove nove tablice bez sortiranja
test1_x = extracted_columns.transpose()

print("\nIzgled nove tablice test1_x:")
```

```
print(test1_x)

# Enkodiranje oznaka
test1_x = test1_x.applymap(encode_labels)

print("\nIzgled nove tablice s oznakama prisutnosti gena nakon
enkodiranja:")
print(test1_x)

# Izdvajanje stupaca 3., 5., 7. itd.
columns_to_extract = sorted_data.columns[2::2]
extracted_columns = sorted_data[columns_to_extract]

# Pretvaranje stupaca u redove nove tablice bez sortiranja
test2_x = extracted_columns.transpose()

print("\nIzgled nove tablice test2_x:")
print(test2_x)

# Učitavanje datoteke actual.csv
actual_file_path = 'leukemia_data/actual.csv'
actual_data = pd.read_csv(actual_file_path)

# Izdvajanje prvih 38 redaka nakon prvog redka sa nazivima stupaca za
train_y
train_y = actual_data.iloc[0:38, 1]

print("\nIzgled train_y (prvih 38 redaka):")
print(train_y)
print(f"Duljina train_y: {len(train_y)}")

# Izdvajanje preostalih 34 redka nakon prvih 38 za test_y i drugi
stupac
test_y = actual_data.iloc[38:73, 1]

print("\nIzgled test_y (preostalih 34 redka):")
print(test_y)
print(f"Duljina test_y: {len(test_y)}")

Ukupan broj stupaca u originalnoj csv datoteci: 70
Ukupan broj redaka u originalnoj csv datoteci: 7129
Naslovni redak originalne CSV datoteke:
['Gene Description' 'Gene Accession Number' '39' 'call' '40' 'call.1'
'42'
 'call.2' '47' 'call.3' '48' 'call.4' '49' 'call.5' '41' 'call.6' '43'
 'call.7' '44' 'call.8' '45' 'call.9' '46' 'call.10' '70' 'call.11'
'71'
 'call.12' '72' 'call.13' '68' 'call.14' '69' 'call.15' '67' 'call.16'
 '55' 'call.17' '56' 'call.18' '59' 'call.19' '52' 'call.20' '53'
 'call.21' '51' 'call.22' '50' 'call.23' '54' 'call.24' '57' 'call.25'
```

```
 '58' 'call.26' '60' 'call.27' '61' 'call.28' '65' 'call.29' '66'
 'call.30' '63' 'call.31' '64' 'call.32' '62' 'call.33']

Naslovni redak nove tablice sorted:
['Gene Description' 'Gene Accession Number' '39' 'call.0' '40'
'call.1'
 '41' 'call.6' '42' 'call.2' '43' 'call.7' '44' 'call.8' '45' 'call.9'
 '46' 'call.10' '47' 'call.3' '48' 'call.4' '49' 'call.5' '50'
'call.23'
 '51' 'call.22' '52' 'call.20' '53' 'call.21' '54' 'call.24' '55'
 'call.17' '56' 'call.18' '57' 'call.25' '58' 'call.26' '59' 'call.19'
 '60' 'call.27' '61' 'call.28' '62' 'call.33' '63' 'call.31' '64'
 'call.32' '65' 'call.29' '66' 'call.30' '67' 'call.16' '68' 'call.14'
 '69' 'call.15' '70' 'call.11' '71' 'call.12' '72' 'call.13']

Izgled nove tablice test1_x:
        0    1    2    3    4    5    6    7    8    9     ... 7119
7120 7121  \
call.0     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.1     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.6     A    A    A    A    A    A    A    A    A    A  ...    A
P    A
call.2     A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.7     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.8     A    A    A    A    A    A    A    A    A    A  ...    A
P    A
call.9     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.10    A    A    A    A    A    A    A    A    A    A  ...    A
P    A
call.3     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.4     A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.5     A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.23    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.22    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.20    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.21    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.24    A    A    A    P    A    A    A    A    A    A  ...    A
```

```
A    A
call.17    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.18    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.25    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.26    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.19    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.27    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.28    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.33    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.31    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.32    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.29    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.30    A    A    A    A    A    A    A    A    A    A  ...    A
A    A
call.16    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.14    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.15    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.11    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.12    A    A    A    A    A    A    A    A    A    A  ...    P
A    A
call.13    A    A    A    A    A    A    A    A    A    A  ...    A
A    A

         7122 7123 7124 7125 7126 7127 7128
call.0      A    A    A    A    A    A    A
call.1      P    A    A    A    A    A    A
call.6      A    A    A    A    A    A    A
call.2      P    A    P    A    A    P    A
call.7      P    A    A    A    A    P    A
call.8      P    A    A    A    A    A    A
call.9      A    A    A    A    A    A    A
call.10     A    A    A    A    A    A    A
call.3      A    A    A    A    A    P    A
call.4      P    A    A    A    A    A    A
```

```
call.5     A    A    A    A    A    A    A
call.23    A    A    A    A    A    P    A
call.22    A    A    A    A    A    P    A
call.20    A    A    A    A    A    A    A
call.21    A    A    A    P    A    A    A
call.24    A    A    A    A    A    A    A
call.17    A    A    A    A    A    A    A
call.18    P    A    A    A    A    P    A
call.25    A    A    A    A    A    A    A
call.26    A    A    A    P    A    A    A
call.19    P    A    A    A    A    A    A
call.27    A    A    A    A    A    M    A
call.28    A    A    A    A    A    A    A
call.33    P    A    A    A    A    A    A
call.31    P    A    A    A    A    A    A
call.32    A    A    A    A    A    A    A
call.29    P    A    A    A    A    A    A
call.30    P    A    A    A    A    P    A
call.16    A    A    A    A    A    A    A
call.14    P    A    A    A    A    A    A
call.15    P    A    A    A    A    A    A
call.11    A    A    A    A    A    P    P
call.12    P    A    A    A    A    A    A
call.13    P    A    A    P    A    A    A

[34 rows x 7129 columns]

Izgled nove tablice s oznakama prisutnosti gena nakon enkodiranja:
         0     1     2     3     4     5     6     7     8
9     ...  \
call.0     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.1     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.6     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.2     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.7     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.8     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.9     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.10    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.3     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.4     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
```

```
1  ...
call.5     -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.23    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.22    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.20    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.21    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.24    -1    -1    -1     1    -1    -1    -1    -1    -1    -
1  ...
call.17    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.18    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.25    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.26    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.19    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.27    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.28    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.33    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.31    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.32    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.29    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.30    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.16    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.14    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.15    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.11    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.12    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
call.13    -1    -1    -1    -1    -1    -1    -1    -1    -1    -
1  ...
```

```
         7119  7120  7121  7122  7123  7124  7125  7126  7127  7128
call.0     -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.1     -1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.6     -1     1    -1    -1    -1    -1    -1    -1    -1    -1
call.2      1    -1    -1     1    -1     1    -1    -1     1    -1
call.7     -1    -1    -1     1    -1    -1    -1    -1     1    -1
call.8     -1     1    -1     1    -1    -1    -1    -1    -1    -1
call.9     -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.10    -1     1    -1    -1    -1    -1    -1    -1    -1    -1
call.3     -1    -1    -1    -1    -1    -1    -1    -1     1    -1
call.4      1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.5     -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.23    -1    -1    -1    -1    -1    -1    -1    -1     1    -1
call.22    -1    -1    -1    -1    -1    -1    -1    -1     1    -1
call.20    -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.21    -1    -1    -1    -1    -1    -1     1    -1    -1    -1
call.24    -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.17    -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.18    -1    -1    -1     1    -1    -1    -1    -1     1    -1
call.25    -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.26    -1    -1    -1    -1    -1    -1     1    -1    -1    -1
call.19    -1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.27    -1    -1    -1    -1    -1    -1    -1    -1     0    -1
call.28     1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.33    -1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.31    -1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.32    -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.29    -1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.30    -1    -1    -1     1    -1    -1    -1    -1     1    -1
call.16     1    -1    -1    -1    -1    -1    -1    -1    -1    -1
call.14     1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.15     1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.11     1    -1    -1    -1    -1    -1    -1    -1     1     1
call.12     1    -1    -1     1    -1    -1    -1    -1    -1    -1
call.13    -1    -1    -1     1    -1    -1     1    -1    -1    -1

[34 rows x 7129 columns]

Izgled nove tablice test2_x:
    0     1     2     3     4     5     6     7     8     9     ...
7119  \
39  -342  -200    41   328  -224  -427  -656  -292   137  -144  ...
277
40   -87  -248   262   295  -226  -493   367  -452   194   162  ...
83
41   -62   -23    -7   142  -233  -284  -167   -97   -12   -70  ...
129
42    22  -153    17   276  -211  -250    55  -141     0   500  ...
413
```

```
43    86   -36  -141   252  -201  -384  -420  -197   -60  -468  ...
341
44  -146   -74   170   174   -32  -318     8  -152  -148    17  ...
180
45  -187  -187   312   142   114  -148  -184  -133    12    97  ...
37
46   -56   -43    43   177  -116  -184  -105   -62     0   -40  ...
103
47  -243  -218  -163   182  -289  -268  -285  -172    52  -134  ...
174
48  -130  -177   -28   266  -170  -326  -222   -93    10   159  ...
233
49  -256  -249  -410    24  -535  -810   709  -316    27    14  ...
76
50  -118  -142   212   314  -401  -452  -336  -310   177  -131  ...
134
51  -112  -185    24   170  -197  -400  -215  -227   100   307  ...
146
52   -21   -13     8    38  -128  -245   409  -102    85   281  ...
-58
53  -202  -274    59   309  -456  -581  -159  -343   236    -7  ...
205
54   -90   -87   102   319  -283  -385  -726  -271   -12  -104  ...
118
55   -34  -144   -17   152  -174  -289   361   -89    87   104  ...
87
56   -95  -118    59   270  -229  -383   172  -187   185   157  ...
206
57  -137   -51   -82   178  -135  -320   -13   -11   112  -176  ...
133
58  -157  -370   -77   340  -438  -364  -216  -210   -86   253  ...
318
59   -12  -172    12   172  -137  -205   358  -104   -25   147  ...
195
60  -172  -122    38    31  -201  -226   242  -117    -6   179  ...
157
61   -47  -442   -21   396  -351  -394   236   -39    95   203  ...
342
62  -176  -284   -81     9  -294  -493  -393  -141   166   -37  ...
81
63  -161  -215   -46   146  -172  -596  -122  -341   171  -147  ...
130
64   -48  -531  -124   431  -496  -696 -1038  -441   235   157  ...
84
65   -62  -198    -5   141  -256  -206  -298  -218   -14   100  ...
92
66   -58  -217    63    95  -191  -230   -86  -152    -6  -249  ...
63
67   -76   -98  -153   237  -215  -122   -68  -118     6   208  ...
```

```
179
68  -154  -136    49   180  -257  -273   141  -123    52   878  ...
214
69   -79  -118   -30    68  -110  -264   -28   -61    40  -217  ...
409
70   -55   -44    12   129  -108  -301  -222  -133   136   320  ...
131
71   -59  -114    23   146  -171  -227   -73  -126    -6   149  ...
214
72  -131  -126   -50   211  -206  -287   -34  -114    62   341  ...
206

    7120  7121  7122  7123  7124  7125  7126  7127  7128
39  1023    67   214  -135  1074   475    48   168   -70
40   529  -295   352   -67    67   263   -33   -33   -21
41   383    46   104    15   245   164    84   100   -18
42   399    16   558    24   893   297     6  1971   -42
43    91   -84   615   -52  1235     9     7  1545   -81
44   690  -142   249  -220   354   -42  -100    45  -108
45   125  -185    13  -148   304    -1  -207   112  -190
46   593     0   -24    18   625   173    63    63   -62
47   277     6    81     2   722   170     0   510   -73
48   643    51   450   -46   612   370    29   333   -19
49  1455  -123   491   -55  1950   906    79   170   -64
50   690    -8   331   -62   882   264    73   315     7
51   621    34   295     4  1110   174     8   533    -4
52   229   -11   201   -35   133    50   -51    91   -43
53   470   -53   392  -106   523   577   -26   208   -71
54   244    14   462  -104   618   308     0   196    20
55   753   -22   259    47   806   342    14   239    24
56   700   -61   381  -105  1068   412   -43   702    18
57   430    31   261   -58   507    64   -11   198   -33
58   408  -275   352    18  1372   642    -9   608   -71
59   922    53   302   -78   673   208   -68   226    78
60   334  -130   242   -53    87    98   -26   153   -49
61   970  -106   240   -86  1111   459    -8    73   -41
62   574   132   618    -9   551   194    20   379   -60
63   639   -27   548   -39   809   445    -2   210    16
64  1141  -121   197  -108   466   349     0   284   -73
65   532   -34   239   -78   707   354   -22   260     5
66   297    36   358     2   423    41     0  1777   -49
67   497    31   241   -20   441    99    -8    80   -12
68   540    13  1075   -45   524   249    40   -68    -1
69   617   -34   738    11   742   234    72   109   -30
70   318    35   241   -66   320   174    -4   176    40
71   760   -38   201   -55   348   208     0    74   -12
72   697     3  1046    27   874   393    34   237    -2

[34 rows x 7129 columns]
```

```
Izgled train_y (prvih 38 redaka):
0     ALL
1     ALL
2     ALL
3     ALL
4     ALL
5     ALL
6     ALL
7     ALL
8     ALL
9     ALL
10    ALL
11    ALL
12    ALL
13    ALL
14    ALL
15    ALL
16    ALL
17    ALL
18    ALL
19    ALL
20    ALL
21    ALL
22    ALL
23    ALL
24    ALL
25    ALL
26    ALL
27    AML
28    AML
29    AML
30    AML
31    AML
32    AML
33    AML
34    AML
35    AML
36    AML
37    AML
Name: cancer, dtype: object
Duljina train_y: 38

Izgled test_y (preostalih 34 redka):
38    ALL
39    ALL
40    ALL
41    ALL
42    ALL
43    ALL
```

```
44    ALL
45    ALL
46    ALL
47    ALL
48    ALL
49    AML
50    AML
51    AML
52    AML
53    AML
54    ALL
55    ALL
56    AML
57    AML
58    ALL
59    AML
60    AML
61    AML
62    AML
63    AML
64    AML
65    AML
66    ALL
67    ALL
68    ALL
69    ALL
70    ALL
71    ALL
Name: cancer, dtype: object
Duljina test_y: 34
```

2.1 klasifikacija leukemije logističkom regresijom

Naš skup podataka sadrži dvije vrste oznaka: **ALL** koja predstavlja akutnu limfoblastičnu leukemiju i **AML** koja predstavlja akutnu mijeloičnu leukemiju. To su ujedno i klase kojima pridružujemo primjere.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import pandas as pd

# Inicijalizacija LabelEncoder-a
label_encoder = LabelEncoder()

# Pretvorba oznaka u numeričke vrijednosti : u principu nije
eksplicitno potrebno to raditi jer će modeli sami raditi enkodiranje
train_y_encoded = label_encoder.fit_transform(train_y)
```

```
test_y_encoded = label_encoder.transform(test_y)

# Pretvorba u pandas Series
train_y_encoded = pd.Series(train_y_encoded)
test_y_encoded = pd.Series(test_y_encoded)

# Enkodiranje oznaka
#train1_x = train1_x.applymap(encode_labels)

# Treniranje modela logističke regresije na train1_x i train_y_encoded
model = LogisticRegression(max_iter=1000, solver='lbfgs')
model.fit(train1_x, train_y)

# Predikcija i točnost na test1_x i test_y
predictions = model.predict(test1_x)
test_accuracy = accuracy_score(test_y, predictions)
print(f"Točnost na skupu za testiranje za značajke koje pokazuju
prisutnost gena (A,P,M)(skupovi test1_x i test_y):
{test_accuracy:.4f}")

predictions2 = model.predict(train1_x)
test_accuracy2 = accuracy_score(train_y, predictions2)
print(f"Točnost na skupu za učenje za značajke koje pokazuju
prisutnost gena (A,P,M)(skupovi train1_x i train_y):
{test_accuracy2:.4f}")

# Treniranje modela logističke regresije na train2_x i train_y_encoded
model.fit(train2_x, train_y_encoded)

# Predikcija i točnost na test2_x i test_y_encoded
predictions3 = model.predict(test2_x)
test_accuracy = accuracy_score(test_y_encoded, predictions3)
print(f"Točnost na skupu za testiranje za numeričke značajke razine
ekspresije (test2_x i test_y): {test_accuracy:.4f}")

train_accuracy = accuracy_score(train_y_encoded,
model.predict(train2_x))
print(f"Točnost na skupu za učenje za numeričke značajke razine
ekspresije (train2_x i train_y): {train_accuracy:.4f}")

# Kombinirani podaci za train1_x i test1_x
combined_x = pd.concat([train1_x, test1_x])
combined_y = pd.concat([train_y_encoded, test_y_encoded])

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train, X_test, y_train, y_test = train_test_split(combined_x,
combined_y, test_size=0.3, random_state=42)

# Treniranje modela logističke regresije na novom skupu za učenje
model.fit(X_train, y_train)
```

```
# Predikcija i točnost na novom skupu za testiranje
predictions = model.predict(X_test)
test_accuracy = accuracy_score(y_test, predictions)
print(f"Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3)
sa značajkama prisutnosti gena -> Točnost na novom skupu za testiranje
: {test_accuracy:.4f}")

predictions4 = model.predict(X_train)
test_accuracy = accuracy_score(y_train, predictions4)
print(f"Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3)
sa značajkama prisutnosti gena -> Točnost na skupu za učenje :
{test_accuracy:.4f}")

# Kombinirani podaci za train2_x i test2_x
combined_x = pd.concat([train2_x, test2_x])
combined_y = pd.concat([train_y_encoded, test_y_encoded])

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train, X_test, y_train, y_test = train_test_split(combined_x,
combined_y, test_size=0.3, random_state=42)

# Treniranje modela logističke regresije na novom skupu za učenje
model.fit(X_train, y_train)

# Predikcija i točnost na novom skupu za testiranje
predictions = model.predict(X_test)
test_accuracy = accuracy_score(y_test, predictions)
print(f"Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3)
za numeričke značajke razine ekspresije gena -> Točnost na novom skupu
za testiranje : {test_accuracy:.4f}")

predictions4 = model.predict(X_train)
test_accuracy = accuracy_score(y_train, predictions4)
print(f"Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3)
za numeričke značajke razine ekspresije gena -> Točnost na skupu za
učenje : {test_accuracy:.4f}")


# print(train_y)
# print(test_y)

# print(f"Tip strukture train_x: {type(train1_x)}")
# print(f"Tip strukture train_y_encoded: {type(train_y_encoded)}")
# print(f"Tip strukture test_x: {type(test1_x)}")
# print(f"Tip strukture test_y_encoded: {type(test_y_encoded)}")


Točnost na skupu za testiranje za značajke koje pokazuju prisutnost
gena (A,P,M)(skupovi test1_x i test_y): 0.9118
Točnost na skupu za učenje za značajke koje pokazuju prisutnost gena
```

```
(A,P,M)(skupovi train1_x i train_y): 1.0000
Točnost na skupu za testiranje za numeričke značajke razine ekspresije
(test2_x i test_y): 0.9706
Točnost na skupu za učenje za numeričke značajke razine ekspresije
(train2_x i train_y): 1.0000
Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3) sa
značajkama prisutnosti gena -> Točnost na novom skupu za testiranje :
1.0000
Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3) sa
značajkama prisutnosti gena -> Točnost na skupu za učenje : 1.0000
Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3) za
numeričke značajke razine ekspresije gena -> Točnost na novom skupu za
testiranje : 1.0000
Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3) za
numeričke značajke razine ekspresije gena -> Točnost na skupu za
učenje : 1.0000
```

Usporedba s referentnom implementacijom: Tablica 7. "rezultati logističke regresije"

2.2 klasifikacija leukemije pomoću SVM-a s linearnim jezgrenim funkcijama

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split


# Pretvorite numpy array u pandas DataFrame ako već nisu
train1_x = pd.DataFrame(train1_x)
test1_x = pd.DataFrame(test1_x)
train2_x = pd.DataFrame(train2_x)
test2_x = pd.DataFrame(test2_x)

# Treniranje SVM modela s linearnom jezgrom na train1_x i
train_y_encoded
model = SVC(kernel='linear', max_iter=1000)
model.fit(train1_x, train_y)

# Predikcija i točnost na test1_x i test_y_encode
predictions = model.predict(test1_x)
test_accuracy = accuracy_score(test_y, predictions)
print(f"Točnost na skupu za testiranje za značajke koje pokazuju
prisutnost gena (A,P,M)(skupovi test1_x i test_y):
{test_accuracy:.4f}")

predictions2 = model.predict(train1_x)
test_accuracy2 = accuracy_score(train_y, predictions2)
print(f"Točnost na skupu za učenje za značajke koje pokazuju
prisutnost gena (A,P,M)(skupovi train1_x i train_y):
```

```
{test_accuracy2:.4f}")

#-------------------------------------------------------------
# Treniranje SVM modela s linearnom jezgrom na train2_x i
train_y_encoded
model = SVC(kernel='linear', max_iter=1000)
model.fit(train2_x, train_y_encoded) #slobodno i train_y jer su to
kategoricki podaci pa ih ne treba enkodeirati

# Predikcija i točnost na test2_x i test_y_encoded
predictions3 = model.predict(test2_x)
test_accuracy = accuracy_score(test_y_encoded, predictions3)
print(f"Točnost na skupu za testiranje za numeričke značajke razine
ekspresije (test2_x i test_y): {test_accuracy:.4f}")

train_accuracy = accuracy_score(train_y_encoded,
model.predict(train2_x))
print(f"Točnost na skupu za učenje za numeričke značajke razine
ekspresije(train2_x i train_y): {train_accuracy:.4f}")

#-------------------------------------------------------------

# Kombinirani podaci za train1_x i test1_x, te train_y_encoded i
test_y_encoded
combined_x = pd.concat([train1_x, test1_x], ignore_index=True)
combined_y = pd.concat([train_y_encoded, test_y_encoded],
ignore_index=True)

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train, X_test, y_train, y_test = train_test_split(combined_x,
combined_y, test_size=0.3, random_state=42)

# Treniranje SVM modela s linearnom jezgrom na novom skupu za učenje
model = SVC(kernel='linear', max_iter=10000)
model.fit(X_train, y_train)

# Predikcija i točnost na novom skupu za testiranje
predictions = model.predict(X_test)
test_accuracy = accuracy_score(y_test, predictions)
print(f"Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3)
sa značajkama prisutnosti gena -> Točnost na novom skupu za testiranje
: {test_accuracy:.4f}")

predictions4 = model.predict(X_train)
test_accuracy = accuracy_score(y_train, predictions4)
print(f"Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3)
sa značajkama prisutnosti gena -> Točnost na skupu za učenje :
{test_accuracy:.4f}")

#-------------------------------------------------------------
```

```
# Kombinirani podaci za train2_x i test2_x, te train_y_encoded i
test_y_encoded
combined_x = pd.concat([train2_x, test2_x], ignore_index=True)
combined_y = pd.concat([train_y_encoded, test_y_encoded],
ignore_index=True)

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train, X_test, y_train, y_test = train_test_split(combined_x,
combined_y, test_size=0.3, random_state=42)

# Treniranje SVM modela s linearnom jezgrom na novom skupu za učenje
model.fit(X_train, y_train)

# Predikcija i točnost na novom skupu za testiranje
predictions = model.predict(X_test)
test_accuracy = accuracy_score(y_test, predictions)
print(f"Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3)
za numeričke značajke razine ekspresije gena -> Točnost na novom skupu
za testiranje : {test_accuracy:.4f}")

predictions4 = model.predict(X_train)
test_accuracy = accuracy_score(y_train, predictions4)
print(f"Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3)
za numeričke značajke razine ekspresije gena -> Točnost na skupu za
učenje : {test_accuracy:.4f}")
```

```
Točnost na skupu za testiranje za značajke koje pokazuju prisutnost
gena (A,P,M)(skupovi test1_x i test_y): 0.9118
Točnost na skupu za učenje za značajke koje pokazuju prisutnost gena
(A,P,M)(skupovi train1_x i train_y): 1.0000
Točnost na skupu za testiranje za numeričke značajke razine ekspresije
(test2_x i test_y): 0.9706
Točnost na skupu za učenje za numeričke značajke razine
ekspresije(train2_x i train_y): 1.0000
Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3) sa
značajkama prisutnosti gena -> Točnost na novom skupu za testiranje :
1.0000
Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3) sa
značajkama prisutnosti gena -> Točnost na skupu za učenje : 1.0000
Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3) za
numeričke značajke razine ekspresije gena -> Točnost na novom skupu za
testiranje : 1.0000
Slučajno odabrani skupovi za učenje i testiranje (omjer 7:3) za
numeričke značajke razine ekspresije gena -> Točnost na skupu za
učenje : 1.0000
```

2.3 klasifikacija leukemije pomoću biblioteke Lazy predict

```
import pandas as pd

from lazypredict.Supervised import LazyClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder


# Prvi skup: Kombinirani podaci za train1_x i test1_x, te
train_y_encoded i test_y_encoded
combined_x1 = pd.concat([train1_x, test1_x], ignore_index=True)
combined_y1 = pd.concat([train_y_encoded, test_y_encoded],
ignore_index=True)

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train1, X_test1, y_train1, y_test1 = train_test_split(combined_x1,
combined_y1, test_size=0.3, random_state=42)

# Drugi skup: Kombinirani podaci za train2_x i test2_x, te
train_y_encoded i test_y_encoded
combined_x2 = pd.concat([train2_x, test2_x], ignore_index=True)
combined_y2 = pd.concat([train_y_encoded, test_y_encoded],
ignore_index=True)

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train2, X_test2, y_train2, y_test2 = train_test_split(combined_x2,
combined_y2, test_size=0.3, random_state=42)

# Kreiranje LazyClassifier objekta
clf = LazyClassifier(verbose=0, ignore_warnings=True,
custom_metric=None)

# Evaluacija modela na prvom skupu
models1, predictions1 = clf.fit(X_train1, X_test1, y_train1, y_test1)
print(f"Rezultati za značajke prisutnosti gena:\n{models1}")

# Evaluacija modela na drugom skupu
models2, predictions2 = clf.fit(X_train2, X_test2, y_train2, y_test2)
print(f"\nRezultati za numeričke značajke razine ekspresije gena:\
n{models2}")
```

```
100%|██████████| 29/29 [00:03<00:00,  8.10it/s]

[LightGBM] [Info] Number of positive: 16, number of negative: 34
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000618 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2728
[LightGBM] [Info] Number of data points in the train set: 50, number
of used features: 778
```

```
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
Rezultati za značajke prisutnosti gena:
                               Accuracy  Balanced Accuracy  ROC AUC
F1 Score  \
Model

LinearSVC                          1.00               1.00     1.00
1.00
SGDClassifier                      1.00               1.00     1.00
1.00
Perceptron                         1.00               1.00     1.00
1.00
PassiveAggressiveClassifier        1.00               1.00     1.00
1.00
LogisticRegression                 0.95               0.94     0.94
0.95
RidgeClassifierCV                  0.95               0.94     0.94
0.95
RidgeClassifier                    0.95               0.94     0.94
0.95
RandomForestClassifier             0.95               0.94     0.94
0.95
NearestCentroid                    0.95               0.94     0.94
0.95
LGBMClassifier                     0.95               0.94     0.94
0.95
BernoulliNB                        0.95               0.94     0.94
0.95
ExtraTreesClassifier               0.95               0.94     0.94
0.95
AdaBoostClassifier                 0.91               0.89     0.89
0.91
XGBClassifier                      0.91               0.89     0.89
0.91
KNeighborsClassifier               0.91               0.89     0.89
0.91
BaggingClassifier                  0.86               0.83     0.83
0.86
NuSVC                              0.82               0.78     0.78
0.80
DecisionTreeClassifier             0.77               0.76     0.76
0.77
ExtraTreeClassifier                0.64               0.66     0.66
```

```
0.64
QuadraticDiscriminantAnalysis      0.50               0.56     0.56
0.45
CalibratedClassifierCV             0.64               0.56     0.56
0.53
GaussianNB                         0.64               0.56     0.56
0.53
LinearDiscriminantAnalysis         0.64               0.56     0.56
0.53
DummyClassifier                    0.59               0.55     0.55
0.57
LabelSpreading                     0.59               0.50     0.50
0.44
SVC                                0.59               0.50     0.50
0.44
LabelPropagation                   0.59               0.50     0.50
0.44

                                Time Taken
Model
LinearSVC                             0.09
SGDClassifier                         0.10
Perceptron                            0.09
PassiveAggressiveClassifier           0.10
LogisticRegression                    0.12
RidgeClassifierCV                     0.09
RidgeClassifier                       0.10
RandomForestClassifier                0.17
NearestCentroid                       0.09
LGBMClassifier                        0.17
BernoulliNB                           0.11
ExtraTreesClassifier                  0.14
AdaBoostClassifier                    0.31
XGBClassifier                         0.45
KNeighborsClassifier                  0.10
BaggingClassifier                     0.15
NuSVC                                 0.11
DecisionTreeClassifier                0.09
ExtraTreeClassifier                   0.09
QuadraticDiscriminantAnalysis         0.10
CalibratedClassifierCV                0.13
GaussianNB                            0.09
LinearDiscriminantAnalysis            0.12
DummyClassifier                       0.09
LabelSpreading                        0.09
SVC                                   0.11
LabelPropagation                      0.09
'tuple' object has no attribute '__name__'
Invalid Classifier(s)
```

```
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf

Rezultati za numeričke značajke razine ekspresije gena:
                               Accuracy  Balanced Accuracy  ROC AUC
F1 Score  \
Model

AdaBoostClassifier                 1.00               1.00     1.00
1.00
GaussianNB                         1.00               1.00     1.00
1.00
SGDClassifier                      1.00               1.00     1.00
1.00
Perceptron                         1.00               1.00     1.00
1.00
LGBMClassifier                     1.00               1.00     1.00
1.00
LinearSVC                          0.95               0.96     0.96
0.95
BaggingClassifier                  0.95               0.96     0.96
0.95
PassiveAggressiveClassifier        0.95               0.96     0.96
0.95
RandomForestClassifier             0.95               0.94     0.94
0.95
ExtraTreesClassifier               0.95               0.94     0.94
0.95
DecisionTreeClassifier             0.95               0.94     0.94
0.95
XGBClassifier                      0.95               0.94     0.94
0.95
LogisticRegression                 0.95               0.94     0.94
0.95
RidgeClassifierCV                  0.95               0.94     0.94
0.95
```

RidgeClassifier	0.95	0.94	0.94	0.95
BernoulliNB	0.82	0.79	0.79	0.81
NearestCentroid	0.82	0.79	0.79	0.81
LinearDiscriminantAnalysis	0.82	0.78	0.78	0.80
SVC	0.77	0.72	0.72	0.75
NuSVC	0.77	0.72	0.72	0.75
CalibratedClassifierCV	0.73	0.67	0.67	0.68
KNeighborsClassifier	0.68	0.61	0.61	0.61
QuadraticDiscriminantAnalysis	0.50	0.56	0.56	0.45
DummyClassifier	0.59	0.55	0.55	0.57
LabelSpreading	0.59	0.50	0.50	0.44
LabelPropagation	0.59	0.50	0.50	0.44
ExtraTreeClassifier	0.50	0.49	0.49	0.50

Model	Time Taken
AdaBoostClassifier	1.13
GaussianNB	0.09
SGDClassifier	0.09
Perceptron	0.09
LGBMClassifier	0.38
LinearSVC	0.10
BaggingClassifier	0.24
PassiveAggressiveClassifier	0.10
RandomForestClassifier	0.20
ExtraTreesClassifier	0.15
DecisionTreeClassifier	0.11
XGBClassifier	0.44
LogisticRegression	0.12
RidgeClassifierCV	0.09
RidgeClassifier	0.09
BernoulliNB	0.10
NearestCentroid	0.10
LinearDiscriminantAnalysis	0.12
SVC	0.11
NuSVC	0.11

```
CalibratedClassifierCV                0.14
KNeighborsClassifier                  0.10
QuadraticDiscriminantAnalysis         0.12
DummyClassifier                       0.09
LabelSpreading                        0.09
LabelPropagation                      0.09
ExtraTreeClassifier                   0.09
```

Uvidom u rezultate za podatke o **značajkama prisutnosti gena** uočavam kako su LinearSVC, SGDClassifier, Perceptron i PassiveAggressiveClassifier modeli koji imaju točnost, uravnoteženu točnost, ROC AUC i F1 Score od 1.00. To znači da su ti modeli ispravno klasificirali sve instance u skupu podataka.

Analogno uvidom u rezultate za **numeričke podatke o razini ekspresije gena** uočavam kako su AdaBoostClassifier, GaussianNB, SGDClassifier, Perceptron i LGBMClassifier imali najbolje moguće rezultate

Analiza stupca ROC AUC...

ROC (Receiver Operating Characteristic) Curve

ROC krivulja prikazuje odnos između True Positive Rate (TPR) i False Positive Rate (FPR) za različite pragove klasifikacije.

- **True Positive Rate (TPR)**: Također poznat kao osjetljivost ili recall, izračunava se kao: [TPR = \frac{TP}{TP + FN}] Gdje je:
  - **TP**: True Positives (točne pozitivne predikcije)
  - **FN**: False Negatives (pogrešne negativne predikcije)
- **False Positive Rate (FPR)**: Izračunava se kao: [FPR = \frac{FP}{FP + TN}] Gdje je:
  - **FP**: False Positives (pogrešne pozitivne predikcije)
  - **TN**: True Negatives (točne negativne predikcije)

AUC (Area Under Curve)

AUC je površina ispod ROC krivulje, a vrijednosti se kreću između 0 i 1.

- **AUC = 1**: Savršen model. Model točno klasificira sve primjere.
- **AUC = 0.5**: Nasumičan model. Model ne razlikuje pozitivne od negativnih primjera bolje od slučajnosti.
- **AUC < 0.5**: Loš model. Model ima lošije performanse od nasumičnog modela.

Zašto je ROC AUC važan?

- **Robustnost na neuravnotežene podatke**: ROC AUC je korisna kada su klase neuravnotežene jer se temelji na osjetljivosti i specifičnosti, a ne samo na točnosti.
- **Prag klasifikacije**: ROC krivulja omogućuje analizu performansi modela za različite pragove klasifikacije, što omogućava odabir optimalnog praga prema specifičnim potrebama.

Ovaj grafički prikaz pomaže u vizualizaciji sposobnosti modela da razlikuje između pozitivnih i negativnih klasa kroz različite pragove, čime se pruža uvid u ukupnu učinkovitost modela.

2.4 Klasifikacija leukemije pomoću XGBoost

```
from xgboost import XGBClassifier

# Prvi skup: Kombinirani podaci za train1_x i test1_x, te
train_y_encoded i test_y_encoded
combined_x1 = pd.concat([train1_x, test1_x], ignore_index=True)
combined_y1 = pd.concat([train_y_encoded, test_y_encoded],
ignore_index=True)

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train1, X_test1, y_train1, y_test1 = train_test_split(combined_x1,
combined_y1, test_size=0.3, random_state=42)

# Drugi skup: Kombinirani podaci za train2_x i test2_x, te
train_y_encoded i test_y_encoded
combined_x2 = pd.concat([train2_x, test2_x], ignore_index=True)
combined_y2 = pd.concat([train_y_encoded, test_y_encoded],
ignore_index=True)

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train2, X_test2, y_train2, y_test2 = train_test_split(combined_x2,
combined_y2, test_size=0.3, random_state=42)

# Treniranje XGBoost modela na prvom skupu
xgb_model1 = XGBClassifier(use_label_encoder=False,
eval_metric='logloss')
xgb_model1.fit(X_train1, y_train1)
predictions1 = xgb_model1.predict(X_test1)
accuracy1 = accuracy_score(y_test1, predictions1)
print(f"Točnost na prvom skupu (prisutnost gena): {accuracy1:.4f}")

# Treniranje XGBoost modela na drugom skupu
xgb_model2 = XGBClassifier(use_label_encoder=False,
eval_metric='logloss')
xgb_model2.fit(X_train2, y_train2)
predictions2 = xgb_model2.predict(X_test2)
accuracy2 = accuracy_score(y_test2, predictions2)
print(f"Točnost na drugom skupu (razina ekspresije gena):
{accuracy2:.4f}")
```

```
Točnost na prvom skupu (prisutnost gena): 0.9091
Točnost na drugom skupu (razina ekspresije gena): 0.9545
```

2.5 Klasifikacija leukemije pomoću Random forest

```
from sklearn.ensemble import RandomForestClassifier

# Prvi skup: Kombinirani podaci za train1_x i test1_x, te
train_y_encoded i test_y_encoded
combined_x1 = pd.concat([train1_x, test1_x], ignore_index=True)
combined_y1 = pd.concat([train_y_encoded, test_y_encoded],
ignore_index=True)

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train1, X_test1, y_train1, y_test1 = train_test_split(combined_x1,
combined_y1, test_size=0.3, random_state=42)

# Drugi skup: Kombinirani podaci za train2_x i test2_x, te
train_y_encoded i test_y_encoded
combined_x2 = pd.concat([train2_x, test2_x], ignore_index=True)
combined_y2 = pd.concat([train_y_encoded, test_y_encoded],
ignore_index=True)

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train2, X_test2, y_train2, y_test2 = train_test_split(combined_x2,
combined_y2, test_size=0.3, random_state=42)

# Treniranje Random Forest modela na prvom skupu
rf_model1 = RandomForestClassifier(random_state=42)
rf_model1.fit(X_train1, y_train1)
predictions1 = rf_model1.predict(X_test1)
accuracy1 = accuracy_score(y_test1, predictions1)
print(f"Točnost na prvom skupu (prisutnost gena): {accuracy1:.4f}")

# Treniranje Random Forest modela na drugom skupu
rf_model2 = RandomForestClassifier(random_state=42)
rf_model2.fit(X_train2, y_train2)
predictions2 = rf_model2.predict(X_test2)
accuracy2 = accuracy_score(y_test2, predictions2)
print(f"Točnost na drugom skupu (razina ekspresije gena):
{accuracy2:.4f}")
```

```
Točnost na prvom skupu (prisutnost gena): 0.9545
Točnost na drugom skupu (razina ekspresije gena): 0.9545
```

2.6 Klasifikacija leukemije pomoću DecisionTree

```
from sklearn.tree import DecisionTreeClassifier


# Prvi skup: Kombinirani podaci za train1_x i test1_x, te
train_y_encoded i test_y_encoded
combined_x1 = pd.concat([train1_x, test1_x], ignore_index=True)
```

```
combined_y1 = pd.concat([train_y_encoded, test_y_encoded],
ignore_index=True)

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train1, X_test1, y_train1, y_test1 = train_test_split(combined_x1,
combined_y1, test_size=0.3, random_state=42)

# Drugi skup: Kombinirani podaci za train2_x i test2_x, te
train_y_encoded i test_y_encoded
combined_x2 = pd.concat([train2_x, test2_x], ignore_index=True)
combined_y2 = pd.concat([train_y_encoded, test_y_encoded],
ignore_index=True)

# Podjela na skupove za učenje i testiranje u omjeru 7:3
X_train2, X_test2, y_train2, y_test2 = train_test_split(combined_x2,
combined_y2, test_size=0.3, random_state=42)

# Treniranje Decision Tree modela na prvom skupu
dt_model1 = DecisionTreeClassifier(random_state=42)
dt_model1.fit(X_train1, y_train1)
predictions1 = dt_model1.predict(X_test1)
accuracy1 = accuracy_score(y_test1, predictions1)
print(f"Točnost na prvom skupu (prisutnost gena): {accuracy1:.4f}")

# Treniranje Decision Tree modela na drugom skupu
dt_model2 = DecisionTreeClassifier(random_state=42)
dt_model2.fit(X_train2, y_train2)
predictions2 = dt_model2.predict(X_test2)
accuracy2 = accuracy_score(y_test2, predictions2)
print(f"Točnost na drugom skupu (razina ekspresije gena):
{accuracy2:.4f}")
```

```
Točnost na prvom skupu (prisutnost gena): 0.7727
Točnost na drugom skupu (razina ekspresije gena): 0.9545
```