Öğrenci Adı: Serra ÖZIŞIK
Öğrenci No: 244311007 (Biyoinformatik)
Teslim Tarihi: 29.06.2025
Ders Kodu: YZM506-Makine Öğrenmesi

FINAL PROJE RAPORU

GitHub Linki: https://github.com/SerraOZSK/244311007-Makine-Ogrenmesi-Final-Projesi-/tree/main

VERİLERİN ELDE EDİLMESİ:

Bu projede incelenen veriler 13.06.2025 tarihinde HIGGS.csv (Whiteson, D. 2014) veri setinden alınmıştır. Bu veri setine https://archive.ics.uci.edu/dataset/280/higgs web sitesinden ulaşılmıştır. Veri setinde Monte Carlo simülasyonları kullanılarak elde edilmiş fiziksel ve kimyasal 11.000.000 veri bulunmaktadır. Projede kullanılmak üzere rastgele 100.000 sample seçilmiştir. Bu işlem için kullanılan kod fig. 1'de gösterilmiştir.

```
#HIGGS verisetinde 11 milyon örnek var. Belleği zorlayacak biçimde fazla.
#100k örneğe indirgenmesi gerekli. Böylelikle analiz hızlanacak.
#Örnek sayısı da analiz için yeterli.

file_path = '/content/drive/MyDrive/HIGGS.csv'
df = pd.read_csv(file_path, header=None)
df.columns = ['label'] + [f'feature_{i}' for i in range(1, 29)]
df_sample = df.sample(n=100000, random_state=42)
print("Veri Boyutu:", df_sample.shape)

#birkaç örnek satır yazdırmak için df_sample.head() yaptım ancak çok uzun sürdü
#ben de sadece veri boyutunu yazdırdım.
```

```
Veri Boyutu: (100000, 29)
```

Fig. 1: HIGGS.csv veri setinin içeri aktarılması ve içerisinden rastgele 100 bin örnek seçimi

Rastgele seçilen verilerin de boyutu yüksek olduğundan işlemleri hızlandırmak adına sadece veri boyutunu gösteren bir çıktı istenmiştir.

PROJEDE KULLANILAN KÜTÜPHANE VE MODÜLLER:

Bu ödev Google Colab bulutu ve A100 GPU donanım hızlandırıcısı kullanılarak hazırlanmıştır. Fig. 2'de projede kullanılan tüm kütüphane ve modüller sıralanmıştır.

```python
#Kullanılan kütüphaneler:

import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectKBest, f_classif, mutual_info_classif
from sklearn.model_selection import GridSearchCV, StratifiedKFold, cross_val_score, cross_validate
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from sklearn.svm import LinearSVC
from itertools import product
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
import warnings

from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_curve, auc
```

Fig. 2: Kullanılan kütüphane ve modüller

BÖLÜM 1: VERİ ÖN İŞLEME (PREPROCESSING)

Verilerdeki aykırı değerler modellerin eğitiminde yanlışlıklara sebep olabileceği için bu değerlerin IQR yöntemi ile düzenlenmesi gerekmektedir. Bu aşama için öncelikle aykırı değerlere sınır değerler üzerinden kırpma işlemi uygulanmıştır. Sonrasında ise aykırı değerler sınır değerlere sabitlenmiştir. Aykırı değerlerin silinmesi veri setinde büyük kayıplara yol açabileceğinden sınır değerlerle değiştirilmesi daha uygun görülmüştür. Bu işlem aykırı değerlerin içeriğini bozsa da model kararlılığını arttırmaktadır. Aşağıda bulunan Fig. 3, aykırı değer analizi için kullanılan girdiyi göstermektedir.

```python
[ ]  #Aykırı Değer Analizi (IQR Yöntemi):

     #Aykırı değerler modellerin yanlış öğrenmesine sebep olabileceği için onları düzenlemek gerekli

     #Aykırı değerleri sınır değerlerle kırpmak (trimming) için
     def remove_outliers_iqr(df, feature_cols):
         for col in feature_cols:
             Q1 = df[col].quantile(0.25)
             Q3 = df[col].quantile(0.75)
             IQR = Q3 - Q1
             alt_limit = Q1 - 1.5 * IQR
             üst_limit = Q3 + 1.5 * IQR

             #aykırı değerleri sınır değerlere sabitlemek için
             df[col] = np.clip(df[col], alt_limit, üst_limit)
         return df

     #label dışındaki sütunlar için
     feature_cols = [col for col in df_sample.columns if col != 'label']

     #aykırı değerleri düzenlemek için
     df_sample = remove_outliers_iqr(df_sample, feature_cols)

     #Bu işlemde aykırı satırlar silinmedi, sadece sınır değerlere sabitlendi.
     # Bu sayede veri kaybının çok olması engellendi.
     # Her ne kadar aykırı içeriği bozulsa da model kararlılığının artması ön planda tutuldu.
```

Fig. 3: Aykırı değer analizi (IQR Yöntemi) için hazırlanan girdi

Modeller örneklerin ölçeklerine göre farklı sonuçlar verebilir. Bu projede birçok model üzerinde kıyaslama yapılacağı için herhangi bir farklılık istenmemektedir. Bu nedenle Min-Max normalizasyonu kullanılarak örnekler [0, 1] aralığına ölçeklendirilmiştir. Fig. 4 bu işlem için hazırlanan girdi satırlarını barındırmaktadır. Preprocessed edilmiş veriler "HIGGS_preprocessed.csv" dosyasına kaydedilmiştir.

```
[ ]  #Min-Max normalizasyonu:
     scaler = MinMaxScaler()
     df_sample[feature_cols] = scaler.fit_transform(df_sample[feature_cols])

     #Modeller farklı ölçeklere göre farklı sonuçlar verebilir. Normalizasyon şart.
     # [0,1] aralığına ölçeklendirildi

     #Kaydetmek için
     df_sample.to_csv("HIGGS_preprocessed.csv", index=False)
```

Fig. 4: Min-Max normalizasyonu kullanılarak veri ölçekleme girdisi

BÖLÜM 2: ÖZELLİK SEÇİMİ (FILTER-BASED FEATURE SELECTION)

Bu bölümde modelin başarısını en fazla etkileyen 15 özellik filtreleme bazlı özellik seçimi ile belirlenmiştir. Bu işlem için ANOVA F-score analizi kullanılmıştır. Her ne kadar mutual information yöntemi ile yapılan seçim değişkenler arasındaki herhangi bir bağlılığı ölçerek daha kapsamlı bir filtreleme sunsa da işlemin hızını yavaşlatacağından dolayı ANOVA yöntemi tercih edilmiştir. Aşağıdaki figürde bu aşama için hazırlanan girdi bulunmaktadır. Seçilen özellikler "HIGGS_secilen_ozellikler.csv" tablosuna kaydedilmiştir.

```
[ ]  #Feature Selection: modelin başarısını en fazla etkileyen 15 özellik seçilecek
     #ANOVA F-score kullanıldı. Hızlı çalıştığı ve filtre temelli olduğu için tercih edildi.

     #Özellikleri labellardan ayırmak için
     X = df_sample.drop("label", axis=1)
     y = df_sample["label"]

     #En iyi 15 özelliği seçmek için
     selector = SelectKBest(score_func=f_classif, k=15)
     X_seçilen = selector.fit_transform(X, y)

     #Seçilen özelliklerin isimlerini almak için
     seçilen_özellikler = X.columns[selector.get_support()]
     print("Seçilen Özellikler: ")
     print(seçilen_özellikler)

     #Yeni kümeyi kaydetmek için
     df_seçilen = pd.concat([y.reset_index(drop=True), pd.DataFrame(X_seçilen, columns=seçilen_özellikler)], axis=1)
     df_seçilen.to_csv("HIGGS_secilen_ozellikler.csv", index=False)
```

Fig. 5: ANOVA F-score yöntemi kullanılarak oluşturulan özellik filtreleme girdisi

Yapılan analiz sonucu toplam 29 özellik arasından 1, 4, 6, 10, 13, 14, 16, 17, 18, 21, 22, 23, 26, 27 ve 28. Özelliklerin modelin başarısını en çok etkileme potansiyeline sahip olduğu belirlenmiştir. Bu bulguları içeren çıktı fig. 6'da bulunmaktadır.

```
Seçilen Özellikler:
Index(['feature_1', 'feature_4', 'feature_6', 'feature_10', 'feature_13',
       'feature_14', 'feature_16', 'feature_17', 'feature_18', 'feature_21',
       'feature_22', 'feature_23', 'feature_26', 'feature_27', 'feature_28'],
      dtype='object')
```

Fig. 6: ANOVA F-score kullanılarak seçilen 15 özellik çıktısı

BÖLÜM 3: MODELLEME VE DEĞERLENDİRME

3A. Farklı Öznitelik Seçim Kombinasyonları Denenerek En iyi Modeli ve Öznitelikleri Belirleme:

Projenin bu aşamasında farklı öznitelik seçimleri uygulanarak en uygun öznitelik kümesi ve sınıflandırıcının belirlenmesi ve böylelikle model performansının optimize edilmesi amaçlanmıştır. Ödev bilgilendirme metninde bulunan flowchart A'ya sadık kalınmış olup flowchart A Fig. 7 olarak bu rapora eklenmiştir.

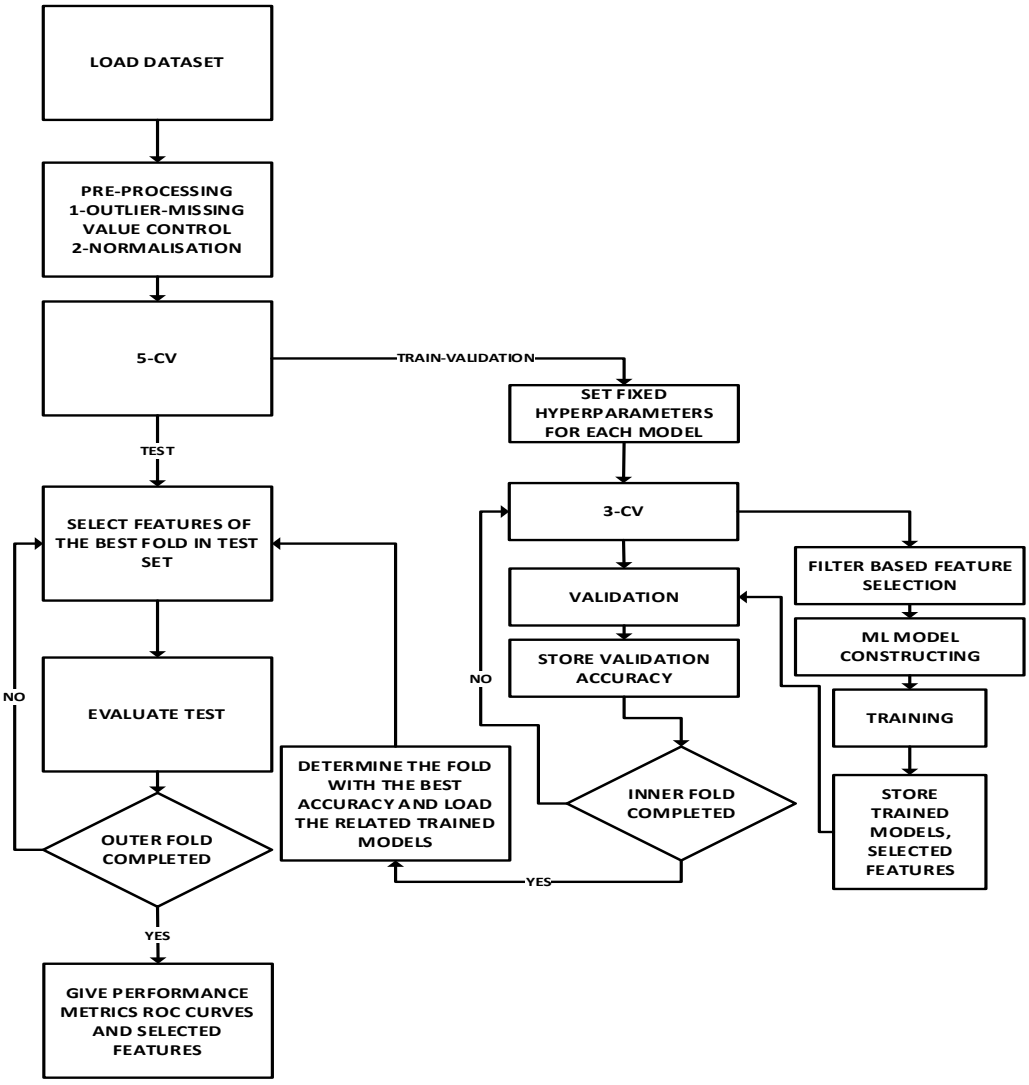

Fig. 7: Farklı öznitelik seçimi ve model kıyaslaması aşamasında takip edilen flowchart A

Bu doğrultuda öznitelik seçimi için ANOVA ve Mutual Information stratejileri belirlenmiş olup her biri için 10, 15 ve 20 değişken seçilerek farklı kombinasyonlar denenmiştir. Aşağıda bu işlem için hazırlanan girdi bulunmaktadır. Outer ve inner CV döngüleri modelin genellenebilirliğini artırmak amacıyla StratifiedKFold yöntemiyle yapılandırılmıştır. Sınıflandırma işlemi için K-Nearest Neighbors (KNN), LinearSVC, Multi-Layer Perceptron (MLP) ve XGBoost algoritmaları uygulanmıştır.

Support Vector Machine (SVC) algoritması 3B'de bahsedilen hiperparametre analizde ne VSCode ne de Google Colab üzerinde sonuç verdiğinden rbf kernelinin kullanılmamasına karar verilmiştir. Bu algoritma veri seti büyük olduğundan RBF'ye göre çok daha hızlı çalışır ve modelin aşırı uyum yapma riskini azaltarak daha stabil sonuçlar vermesine katkı sağlar.

```
#Bölüm 3A. İç döngüde farklı öznitelik seçim kombinasyonları denenerek en iyi model ve öznitelikler belirlenir.
#Bu işlemdeki farklı seçim kombinasyonları için hem anova hem mutual inf kullanılacak ve k değerleri 10, 15 ve 20 için kıyaslanacak

warnings.filterwarnings("ignore")  # MutualInfo warning'lerini kapatmak için

# Outer ve inner CV'yi hazırlamak için
outer_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
inner_cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# Özellik seçimi varyasyonları
k_list = [10, 15, 20]
score_funcs = {
    "ANOVA": f_classif,
    "MutualInfo": mutual_info_classif
}

# Modeller
models = {
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "MLP": MLPClassifier(hidden_layer_sizes=(100,), activation='relu', max_iter=300, early_stopping=True),
    "LinearSVC": LinearSVC(C=1.0, max_iter=1000),
    "XGBoost": XGBClassifier(tree_method='gpu_hist', use_label_encoder=False, eval_metric='logloss')
}
```

Fig. 8: Outer ve inner CV döngülerinin tanımlandığı ve farklı öznitelik seçim kombinasyonlarının bulunduğu girdi

Fig. 9, makine öğrenme modellerinin Nested CV ile sistematik bir şekilde değerlendirilmesini sağlayan kodları barındırmaktadır. Aşağıdaki figürde de görüldüğü üzere her model daha önce belirtilen öznitelik seçim stratejisi ile test edilmiştir. Öznitelikler üzerinde inner fold validasyonu için SelectKBest kullanılarak modelin doğruluğu belirlenmiş (Fig. 9A) ve outer test verisi üzerinde sınıflandırma başarısını ölçmek adına accuracy, precision, recall, f1-score ve mümkünse ROC-AUC performans metrikleri hesaplanmıştır (Fig. 9B). LinearSVC predict_proba metodunu desteklemediği için ROC-AUC hesaplaması sunmamaktadır (Scikit Learn, n.d.). Bu durum için decision function kullanılarak örneklerin sınıflandırma düzlemine olan uzaklıklarının elde edilmesiyle ROC eğrisi çizimi için gereken ROC-AUC skorunun hesaplanabilmesini sağlanmıştır. Böylelikle model kararlılığını ve öznitelik seçiminin etkisini detaylıca değerlendirmenin önü açılmıştır.

```python
# Sonuç listesi hazırlamak için
all_results = []

for model_name, model in models.items():
    print(f"\n Model: {model_name}")

    for sel_name, sel_func in score_funcs.items():
        for k in k_list:
            inner_fold_scores = []
            outer_metrics = []

            for fold, (train_idx, test_idx) in enumerate(outer_cv.split(X, y)):
                X_train_full, X_test_full = X.iloc[train_idx], X.iloc[test_idx]
                y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

                # Özellik seçimi
                selector = SelectKBest(score_func=sel_func, k=k)
                X_train = selector.fit_transform(X_train_full, y_train)
                X_test = selector.transform(X_test_full)

                # Inner CV doğruluğunu hesaplamak için
                try:
                    val_score = cross_val_score(model, X_train, y_train, cv=inner_cv, scoring='accuracy', n_jobs=-1).mean()
                except Exception as e:
                    print(f"Hata oluştu ({model_name}-{sel_name}-k{k}) inner CV sırasında: {e}")
                    val_score = 0
                inner_fold_scores.append(val_score)
```

A

```python
    # Outer test set hazırlığı ve performans metrikleri için
    try:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        acc = accuracy_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred, zero_division=0)
        rec = recall_score(y_test, y_pred, zero_division=0)
        f1 = f1_score(y_test, y_pred, zero_division=0)

        if hasattr(model, "predict_proba"):
            y_probs = model.predict_proba(X_test)[:, 1]
            roc = roc_auc_score(y_test, y_probs)
        elif hasattr(model, "decision_function"):
            decision_scores = model.decision_function(X_test)
            roc = roc_auc_score(y_test, decision_scores)
        else:
            roc = np.nan

        outer_metrics.append((acc, prec, rec, f1, roc))

    except Exception as e:
        print(f"Hata oluştu ({model_name}-{sel_name}-k{k}) outer test sırasında: {e}")
        outer_metrics.append((0, 0, 0, 0, None))
```

B

Fig. 9: Inner fold validasyonu (A) ve outer fold testini (B) hazırlamak ve sonuçlandırmak için hazırlanan girdi

Hesaplanan metrikleri sunmak ve tablolaştırmak adına aşağıda bulunan girdi kullanılmıştır.

```
            # Ortalama metrikleri kaydetmek için
            mean_outer = pd.DataFrame(outer_metrics, columns=["Accuracy", "Precision", "Recall", "F1 Score", "ROC-AUC"]).mean(numeric_only=True)
            roc_auc = mean_outer["ROC-AUC"] if "ROC-AUC" in mean_outer else np.nan

            all_results.append({
                "Model": model_name,
                "Selector": sel_name,
                "k": k,
                "Mean Accuracy": mean_outer["Accuracy"],
                "Mean Precision": mean_outer["Precision"],
                "Mean Recall": mean_outer["Recall"],
                "Mean F1 Score": mean_outer["F1 Score"],
                "Mean ROC-AUC": roc_auc,
                "Mean Inner CV Accuracy": sum(inner_fold_scores) / len(inner_fold_scores)
            })

# Sonuçları tablolaştırmak için
results_df = pd.DataFrame(all_results)
print("\n Ortalama Performans Özeti:")
display(results_df.sort_values("Mean Accuracy", ascending=False).round(4))
```

Fig. 10: Performans metriklerini kaydetmek ve tablolaştırmak için hazırlanan girdi

Tablo 1: Farklı Öznitelik Seçimi Kombinasyonlarına Ait Ortalama Performans Özeti

index	*Model*	*Selector*	*k*	*Mean Accuracy*	*Mean Precision*	*Mean Recall*	*Mean F1 Score*	*Mean ROC-AUC*	*Mean Inner CV Accuracy*
0	KNN	ANOVA	10	0.6575	0.6671	0.7018	0.684	0.7053	0.6552
1	KNN	ANOVA	15	0.6457	0.655	0.6954	0.6746	0.6917	0.643
2	KNN	ANOVA	20	0.6286	0.6377	0.6868	0.6614	0.6672	0.6256
3	KNN	MutualInfo	10	0.6521	0.66	0.7039	0.6813	0.6978	0.6478
4	KNN	MutualInfo	15	0.6434	0.6505	0.7022	0.6753	0.6869	0.6399
5	KNN	MutualInfo	20	0.6321	0.639	0.6973	0.6669	0.6713	0.6267
12	LinearSVC	ANOVA	10	0.638	0.6367	0.7325	0.6813	0.6755	0.6379
13	LinearSVC	ANOVA	15	0.6422	0.6408	0.7338	0.6842	0.6786	0.6418
14	LinearSVC	ANOVA	20	0.6422	0.6402	0.7363	0.6849	0.6803	0.6411
15	LinearSVC	MutualInfo	10	0.6353	0.6267	0.7654	0.6891	0.6643	0.6349
16	LinearSVC	MutualInfo	15	0.6407	0.6347	0.7529	0.6888	0.6774	0.6404
17	LinearSVC	MutualInfo	20	0.6426	0.6384	0.7455	0.6878	0.6834	0.6424
6	MLP	ANOVA	10	0.7012	0.7237	0.7022	0.7128	0.7682	0.6988
7	MLP	ANOVA	15	0.7069	0.7187	0.7316	0.7249	0.7777	0.7041
8	MLP	ANOVA	20	0.7093	0.7124	0.7544	0.7326	0.7796	0.7027
9	MLP	MutualInfo	10	0.6956	0.7101	0.7165	0.713	0.7636	0.6958
10	MLP	MutualInfo	15	0.7076	0.7174	0.7365	0.7267	0.7786	0.7047
11	MLP	MutualInfo	20	0.7109	0.7227	0.7354	0.7285	0.7829	0.7074
18	XGBoost	ANOVA	10	0.7057	0.7222	0.7194	0.7208	0.778	0.7024
19	XGBoost	ANOVA	15	0.7154	0.7296	0.7325	0.731	0.7901	0.7105
20	XGBoost	ANOVA	20	0.717	0.7305	0.7355	0.733	0.7926	0.7122
21	XGBoost	MutualInfo	10	0.7039	0.7157	0.7291	0.7223	0.7786	0.7001
22	XGBoost	MutualInfo	15	0.7213	0.7333	0.7422	0.7377	0.7981	0.7164
23	XGBoost	MutualInfo	20	0.722	0.7331	0.7448	0.7389	0.8002	0.7173

Yukarıda farklı öznitelik seçimi kombinasyonlarının model başarımı üzerindeki etkilerini birçok metrik ile gösteren sonuç tablosu bulunmaktadır. Tabloda ortalama accuracy, precision, recall, f1-score, ROC-AUC değeri ve inner CV accuracy skoru kıyaslanmaktadır. KNN modeli için en yüksek ortalama accuracy değeri k = 10 & ANOVA kombinasyonuna aittir ve %65,75 olarak kaydedilmiştir. Benzer şekilde bu kombinasyon KNN modeli için en yüksek ROC-AUC değerine (0.7053) sahiptir. Bu model için öznitelik sayısının artışı mean accuracy ve ROC-AUC skorlarının

azalmasıyla sonuçlanmıştır. Bu durum algoritmada fazla öznitelik seçiminin gürültüye sebep olabileceğine işaret edebilir. LinearSVC modeline ait sonuçlar incelendiğinde en yüksek mean accuracy değerinin %64,26 ile k = 20 & Mutual Information kombinasyonuna ait olduğu görülmektedir. Bu kombinasyona ait ROC-AUC değeri ise 0.6834 olarak kaydedilmiştir. LinearSVC modeline ait ROC-AUC değeri decision function ile hesaplanmasına rağmen recall değerleri (%73-76) oldukça yüksektir. Bu durum güçlü bir duyarlılık profili sergilediğini ifade etmektedir. MLP modeline dair sonuçlar arasında en yüksek mean accuracy değeri %71.09 olup k = 20 & Mutual Information kombinasyonuna aittir. ROC-AUC değeri ise 0.7829'dur. XGBoost modeline bakıldığında çoğu modele paralel biçimde en yüksek mean accuracy değerinin k = 20 & Mutual Information kombinasyonuna ve %72,2 değerine ait olduğu gözlemlenmektedir. ROC-AUC değeri ise 0.8002 ile tüm modeller arasındaki en yüksek değerdir.

```python
#OvA Yöntem ile ROC Plot çizdirmek için önce yukarıda eksik olan verileri toparlamak gerekli
roc_plot_data = []

for model_name, model in models.items():
    print(f"\n Model: {model_name}")

    for sel_name, sel_func in score_funcs.items():
        for k in k_list:
            inner_fold_scores = []
            outer_metrics = []

            fpr_list = []
            tpr_list = []
            auc_list = []

            for fold, (train_idx, test_idx) in enumerate(outer_cv.split(X, y)):
                X_train_full, X_test_full = X.iloc[train_idx], X.iloc[test_idx]
                y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

                selector = SelectKBest(score_func=sel_func, k=k)
                X_train = selector.fit_transform(X_train_full, y_train)
                X_test = selector.transform(X_test_full)
```

```
    try:
        val_score = cross_val_score(model, X_train, y_train, cv=inner_cv, scoring='accuracy', n_jobs=-1).mean()
    except Exception as e:
        print(f"Hata oluştu ({model_name}-{sel_name}-k{k}) inner CV sırasında: {e}")
        val_score = 0
    inner_fold_scores.append(val_score)

    try:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        if hasattr(model, "predict_proba"):
            scores = model.predict_proba(X_test)[:, 1]
        elif hasattr(model, "decision_function"):
            scores = model.decision_function(X_test)
        else:
            scores = None

        if scores is not None:
            from sklearn.metrics import roc_curve, roc_auc_score
            fpr, tpr, _ = roc_curve(y_test, scores)
            auc = roc_auc_score(y_test, scores)

            fpr_list.append(np.interp(np.linspace(0, 1, 100), fpr, tpr))
            auc_list.append(auc)

        acc = accuracy_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred, zero_division=0)
        rec = recall_score(y_test, y_pred, zero_division=0)
        f1 = f1_score(y_test, y_pred, zero_division=0)

        outer_metrics.append((acc, prec, rec, f1, auc))
mean_outer = pd.DataFrame(outer_metrics, columns=["Accuracy", "Precision", "Recall", "F1 Score", "ROC-AUC"]).mean(numeric_only=True)
roc_auc = mean_outer["ROC-AUC"] if "ROC-AUC" in mean_outer else np.nan

all_results.append({
    "Model": model_name,
    "Selector": sel_name,
    "k": k,
    "Mean Accuracy": mean_outer["Accuracy"],
    "Mean Precision": mean_outer["Precision"],
    "Mean Recall": mean_outer["Recall"],
    "Mean F1 Score": mean_outer["F1 Score"],
    "Mean ROC-AUC": roc_auc,
    "Mean Inner CV Accuracy": sum(inner_fold_scores) / len(inner_fold_scores)
})
```

Fig. 11: OvA yöntemi ile ROC-AUC plot hazırlayabilmek için hazırlanan ön aşama girdisi

Yukarıda bulunan figürde tüm modeller için seçici & öznitelik kombinasyon eğrilerini oluşturmaya yönelik hazırlanmıştır. Modellere uygun predict_proba ve decision function özellikleri doğrultusunda skorların elde edilmesi sağlanmıştır. Outer test kümesi kullanılarak FPR-TPR eğrileri fpr_list ve auc_list değişkenlerine kaydedilmiştir. s

```
            # ROC eğrisi için ortalama TPR hesapla ve kaydet
            if fpr_list and auc_list:
                mean_fpr = np.linspace(0, 1, 100)
                mean_tpr = np.mean(fpr_list, axis=0)
                mean_auc = np.mean(auc_list)

                roc_plot_data.append({
                    "Model": model_name,
                    "Selector": sel_name,
                    "k": k,
                    "FPR": mean_fpr,
                    "TPR": mean_tpr,
                    "ROC-AUC": mean_auc
                })


def plot_roc_per_model(roc_data):
    unique_models = sorted(set(entry["Model"] for entry in roc_data))

    for model in unique_models:
        for method in ["ANOVA", "MutualInfo"]:
            filtered = [d for d in roc_data if d["Model"] == model and d["Selector"] == method]

            if not filtered:
                continue

            plt.figure(figsize=(8, 6))
            palette = sns.color_palette("Set1", len(filtered))

            for i, entry in enumerate(filtered):
                plt.plot(entry["FPR"], entry["TPR"],
                         label=f'k={entry["k"]} (AUC={entry["ROC-AUC"]:.3f})',
                         color=palette[i])
            plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
            plt.title(f'{model} - {method} Feature Selection', fontsize=13)
            plt.xlabel("False Positive Rate")
            plt.ylabel("True Positive Rate")
            plt.legend(loc="lower right")
            plt.grid(True)
            plt.tight_layout()
            plt.show()
```

Fig. 12: ROC eğri grafiklerini çizdirmek için oluşturulan girdi

Fig. 12'de ROC eğrilerinin çizim aşamasını içeren bir kod bloğu bulunmaktadır. Mean_fpr, mean_tpr ve mean_auc değerleri hesaplanarak roc_plot_data listesine kaydedilmiştir. Bu sayede katmanlar arası varyasyon düzleştirilerek daha temiz ve karşılaştırılabilir bir görünüm elde edilmesi hedeflenmiştir.

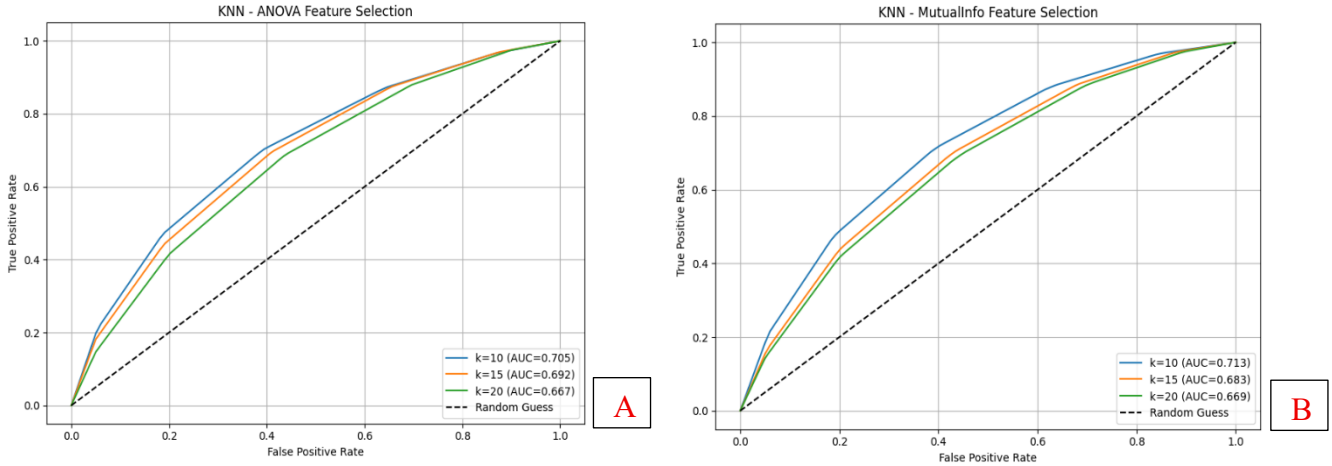

Fig. 13: KNN modeline ait farklı öznitelikler ve seçim (A: ANOVA & B: Mutual Information) kombinasyonlarına ait OvA yöntemiyle hazırlanmış ROC Plotlarının kıyaslaması

Fig. 13'te bulunan ROC eğrileri KNN modeline ait farklı öznitelik değerleri (k = 10, 15, 20) ve seçimlerinin kombinasyonlarına aittir. Fig. 13A'da ANOVA seçimi kıyaslanırken 13B ise Mutual Information seçim yöntemini kıyaslamaktadır. ANOVA ile seçilen k = 10 değerine ait ROC-AUC skor 0.705 iken bu skor k =15 için 0.692 ve k = 20 için 0.667'dir. Mutual Info ile seçilen k = 10 değerine ait ROC skoru 0.713, k = 15 için 0.683 ve k = 20 için 0.669'dur. Her iki grafikte de k = 10 değerlerine ait ROC-AUC değerleri daha yüksektir. Bu grafiklere bakılarak Mutual Info & k = 10 kombinasyonun en yüksek ROC eğrisini sağladığı söylenebilir.

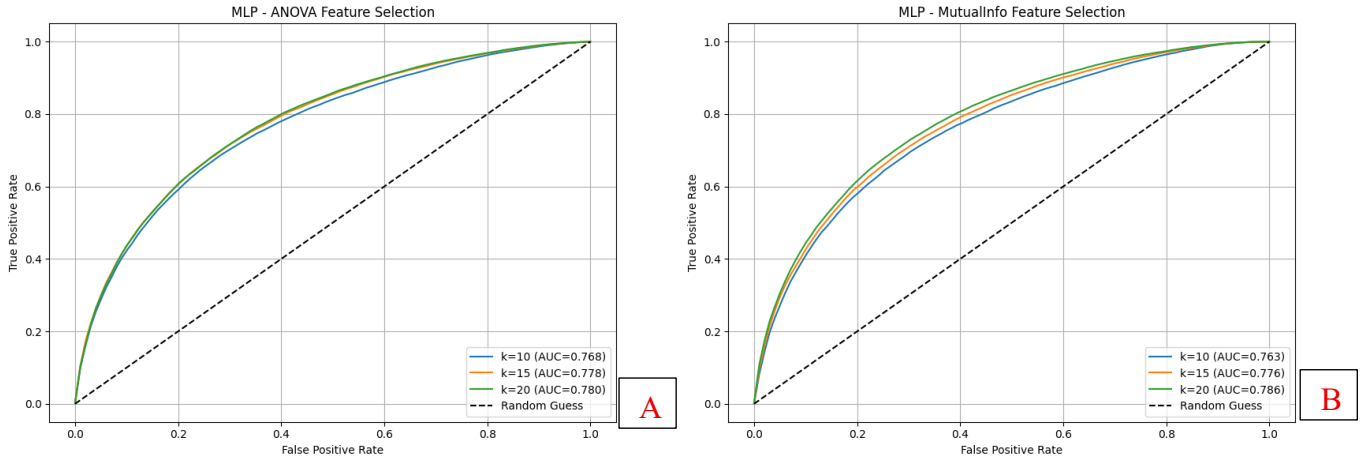

Fig. 14: MLP modeline ait farklı öznitelikler ve seçim (A: ANOVA & B: Mutual Information) kombinasyonlarına ait OvA yöntemiyle hazırlanmış ROC Plotlarının kıyaslaması

Fig. 14'te bulunan ROC eğrileri MLP modeline ait farklı öznitelik değerleri (k = 10, 15, 20) ve seçimlerinin kombinasyonlarına aittir. Fig. 14A'da ANOVA seçimi kıyaslanırken 14B ise Mutual Information seçim yöntemini kıyaslamaktadır. ANOVA ile seçilen k = 10 değerine ait ROC-AUC skor 0.768 iken bu skor k =15 için 0.778 ve k = 20 için 0.780'dir. Mutual Info ile seçilen k = 10 değerine ait ROC skoru 0.763, k = 15 için 0.776 ve k = 20 için 0.786'dır. Her ne kadar ROC-AUC skorları aynı k değerleri için birbirine oldukça yakın olsa da en yüksek ROC değerinin 0.006 farkla k = 20 & Mutual Information kombinasyonuna ait olduğu söylenebilir. ROC eğrilerinin birbirine paralel ilerlemesi, modelin farklı öznitelik sayılarında da benzer başarıya sahip olduğunu göstermektedir.

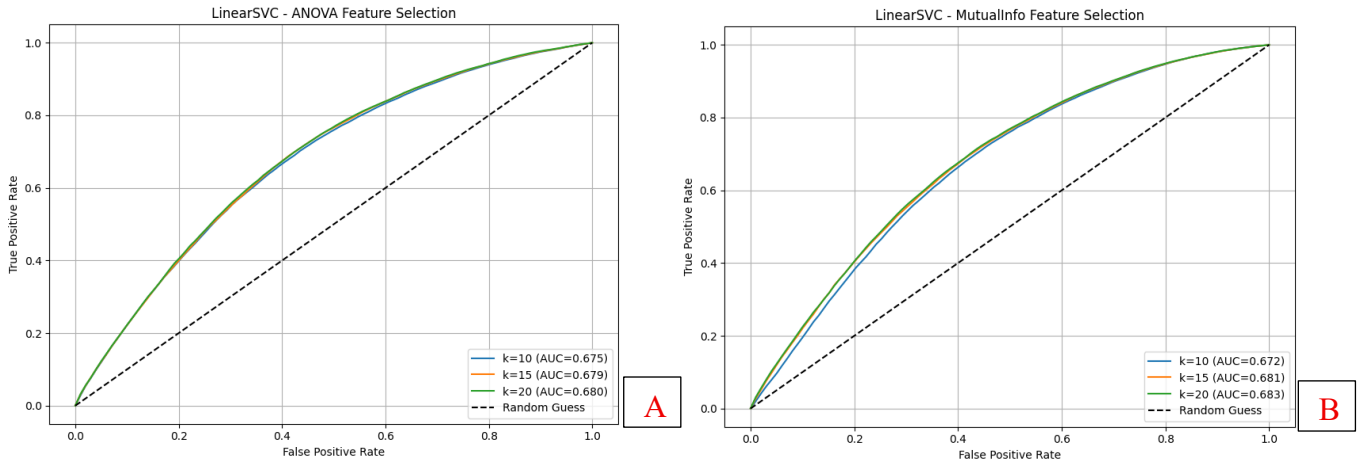

Fig. 15: LinearSVC modeline ait farklı öznitelikler ve seçim (A: ANOVA & B: Mutual Information) kombinasyonlarına ait OvA yöntemiyle hazırlanmış ROC Plotlarının kıyaslaması

Fig. 15'te bulunan ROC eğrileri LinearSVC modeline ait farklı öznitelik değerleri (k = 10, 15, 20) ve seçimlerinin kombinasyonlarına aittir. Fig. 15A'da ANOVA seçimi kıyaslanırken 15B ise Mutual Information seçim yöntemini kıyaslamaktadır. Daha önce de bahsedildiği gibi, LinearSVC direkt olarak predict_proba metodunu desteklemediği için ROC eğrisini hesaplamak için gereken skor decision function kullanılarak elde edilmiştir. ANOVA ile seçilen k = 10 değerine ait ROC-AUC skor 0.675 iken bu skor k =15 için 0.679 ve k = 20 için 0.680'dir. Mutual Info ile seçilen k = 10 değerine ait ROC skoru 0.672, k = 15 için 0.681 ve k = 20 için 0.683'tür. Her iki grafikte de ROC-AUC skorları, öznitelik sayısının artışıyla doğru orantılı olarak, kademeli bir artım sergilemiştir. Her iki seçim yöntemi için de aynı k değerleri için benzer ROC-AUC skorları kaydedilmiş olsa da Mutual Information yöntemine ait k = 10 ve k = 15 değerleri arasındaki belirgin yükselme Mutual Information'ın daha ayrıştırıcı olduğunu gösterebilir.

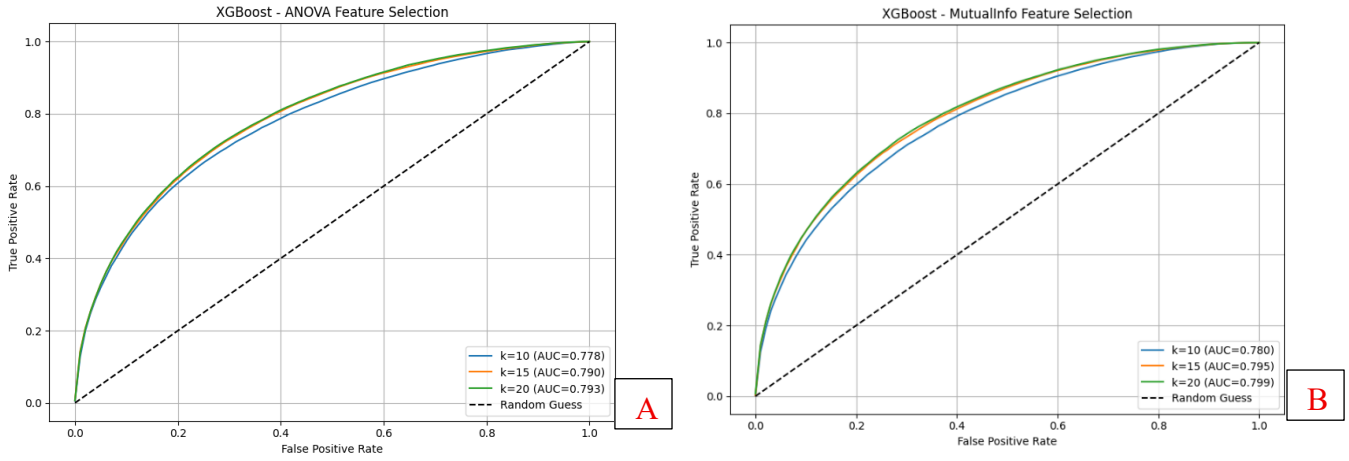

Fig. 16: XGBoost modeline ait farklı öznitelikler ve seçim (A: ANOVA & B: Mutual Information) kombinasyonlarına ait OvA yöntemiyle hazırlanmış ROC Plotlarının kıyaslaması

Fig. 16'da bulunan ROC eğrileri XGBoost modeline ait farklı öznitelik değerleri (k = 10, 15, 20) ve seçimlerinin kombinasyonlarına aittir. Fig. 14A'da ANOVA seçimi kıyaslanırken 16B ise Mutual Information seçim yöntemini kıyaslamaktadır. ANOVA ile seçilen k = 10 değerine ait ROC-

AUC skor 0.778 iken bu skor k =15 için 0.790 ve k = 20 için 0.793'tür. Mutual Info ile seçilen k = 10 değerine ait ROC skoru 0.780, k = 15 için 0.795 ve k = 20 için 0.799'dur. Görülüyor ki, her iki yöntem de artan k değerleri ile paralel sonuçlar vermektedir. Her ne kadar iki yöntem benzer performanslara sahip olsa da Mutual Information'a ait skorlar küçük bir farkla üstünlük sağlamaktadır. Mutual Information ve k = 20 kombinasyonu tüm modeller arasında en yüksek performansa sahiptir.

3B. Farklı Hiperparametre Kombinasyonları Denenerek En İyi Model ve Hiperparametreleri Belirleme:

Projenin bu aşamasında farklı hiperparametre kombinasyonları denenerel her sınıflandırma algoritması için en yüksek genellenebilirlik başarımına sahip yapılandırmaları belirlemek amaçlanmıştır. Bu süreçte ödev bilgilendirme metnindeki flowchart B takip edilmiştir. Flowchart B aşağıdaki figürde gösterilmiştir.

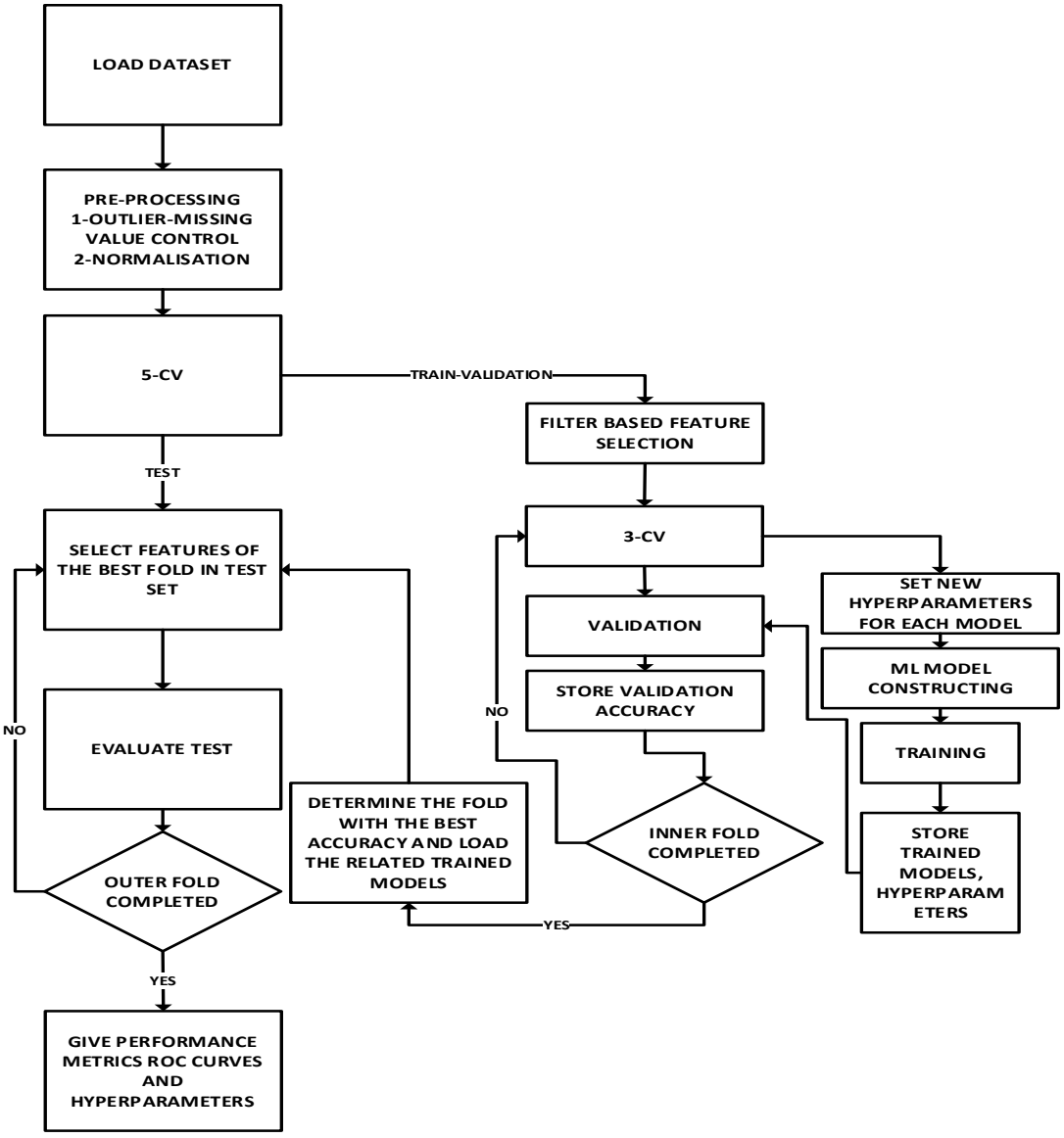

Fig. 17: Farklı hiperparametre kombinasyonlarını kıyaslayarak en iyi performansa sahip modeli seçme aşamasında takip edilen flowchart B

Bu aşamada, öncelikle, farklı hiperparametre ayarları sadece eğitim verisi kullanılarak test edilmiştir. Böylelikle öznitelik seçimi kaynaklı veri sızıntısının önüne geçilmiş ve outer test verisine karşı tarafsız bir değerlendirme yapılması sağlanmıştır. Kıyaslanacak hiperparametreler KNN algoritması için n_neighbors sayısı 3 ile 11 arasında, LinearSVC algoritması için C değeri 0.1, 1 ve 10 ve MLP algoritması için hidden_layer_sizes = [(50), (100)], activation = 'relu', 'tanh'] olarak belirlenmiştir. Hiperparametre optimizasyonuna yönelik hazırlanan bu sistematik tarama modellerin aşırı uyum riskini azaltmakla beraber en yüksek accuracy, precision, recall, f1-score, ve ROC-AUC performans metriklerinde optimum sonuçları elde etmeyi hedeflemiştir. Bahsedilen işlemleri yapabilmesi adına hazırlanan girdi aşağıda bulunan figürler 18 ve 19'da bulunmaktadır.

```python
# Gerekirse tüm kolonları göster:
pd.set_option("display.max_columns", None)
warnings.filterwarnings("ignore")

outer_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
inner_cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

fold_metrics = []
all_combination_scores = []

# ROC-AUC inner CV için uygun scoring türü
scoring_metrics = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc_ovr'  # OvR çok sınıflı durumlar için daha uyumlu
}

for fold, (train_idx, test_idx) in enumerate(outer_cv.split(X, y)):
    print(f"\n--- Outer Fold {fold + 1} ---")
    X_train_full, X_test_full = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

    selector = SelectKBest(score_func=f_classif, k=20)
    X_train = selector.fit_transform(X_train_full, y_train)
    X_test = selector.transform(X_test_full)
```

A

```python
model_scores = []

# KNN
for n in range(3, 12):
    knn = KNeighborsClassifier(n_neighbors=n)
    scores = cross_validate(knn, X_train, y_train, cv=inner_cv, scoring=scoring_metrics, n_jobs=-1)
    means = {m: np.mean(scores[f'test_{m}']) for m in scoring_metrics}
    model_scores.append((means['accuracy'], knn, {'n_neighbors': n}))
    all_combination_scores.append({'Outer Fold': f'Fold {fold + 1}', 'Model': 'KNN', 'Params': {'n_neighbors': n}, **means})

# MLP
for hls, act in product([(50,), (100,)], ['relu', 'tanh']):
    mlp = MLPClassifier(hidden_layer_sizes=hls, activation=act, max_iter=300)
    scores = cross_validate(mlp, X_train, y_train, cv=inner_cv, scoring=scoring_metrics, n_jobs=-1)
    means = {m: np.mean(scores[f'test_{m}']) for m in scoring_metrics}
    model_scores.append((means['accuracy'], mlp, {'hidden_layer_sizes': hls, 'activation': act}))
    all_combination_scores.append({'Outer Fold': f'Fold {fold + 1}', 'Model': 'MLP', 'Params': {'hidden_layer_sizes': hls, 'activation': act}, **means})

# LinearSVC
for c in [0.1, 1, 10]:
    lsvc = LinearSVC(C=c, max_iter=1000)
    scores = cross_validate(lsvc, X_train, y_train, cv=inner_cv, scoring=scoring_metrics, n_jobs=-1)
    means = {m: np.mean(scores[f'test_{m}']) for m in scoring_metrics}
    print(f"LinearSVC (C={c}) | Accuracy: {means['accuracy']:.4f}")
    model_scores.append((means['accuracy'], lsvc, {'C': c}))
    all_combination_scores.append({'Outer Fold': f'Fold {fold + 1}', 'Model': 'LinearSVC', 'Params': {'C': c}, **means})

# XGBoost
try:
    xgb = XGBClassifier(tree_method='gpu_hist', use_label_encoder=False, eval_metric='logloss')
    scores = cross_validate(xgb, X_train, y_train, cv=inner_cv, scoring=scoring_metrics, n_jobs=-1)
    means = {m: np.mean(scores[f'test_{m}']) for m in scoring_metrics}
    model_scores.append((means['accuracy'], xgb, {'default': True}))
    all_combination_scores.append({'Outer Fold': f'Fold {fold + 1}', 'Model': 'XGBClassifier', 'Params': {'default': True}, **means})
except Exception as e:
```

B

Fig. 18: Outer ve inner CV döngülerinin tanımlandığı (A) ve farklı hiperparametre kombinasyonlarının belirlendiği (B) girdi

```
    # En iyi modeli belirle
    best_score, best_model, best_params = max(model_scores, key=lambda x: x[0])
    best_model.fit(X_train, y_train)
    y_pred = best_model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, zero_division=0)
    rec = recall_score(y_test, y_pred, zero_division=0)
    f1 = f1_score(y_test, y_pred, zero_division=0)

    if hasattr(best_model, "predict_proba"):
        y_probs = best_model.predict_proba(X_test)[:, 1]
        roc = roc_auc_score(y_test, y_probs)
    elif hasattr(best_model, "decision_function"):
        scores = best_model.decision_function(X_test)
        roc = roc_auc_score(y_test, scores)
    else:
        roc = None

    fold_metrics.append({
        'Outer Fold': f'Fold {fold + 1}',
        'Model': type(best_model).__name__,
        'Best Params': best_params,
        'Accuracy': acc,
        'Precision': prec,
        'Recall': rec,
        'F1 Score': f1,
        'ROC-AUC': roc
    })

# 1) Outer Fold test seti sonuçları
results_df = pd.DataFrame(fold_metrics)
print("\n Outer Fold Performans Özeti:")
display(results_df.round(4))
print(f"\n Ortalama Accuracy: {results_df['Accuracy'].mean():.4f}")
```

A

```
# 2) Inner CV sonuçlarının detaylı tablosu
all_scores_df = pd.DataFrame(all_combination_scores)
print("\n Tüm Model ve Parametre Kombinasyonlarının Inner CV Sonuçları:")
display(all_scores_df.sort_values(['Outer Fold', 'Model', 'accuracy'], ascending=[True, True, False]).round(4))

#Taabloyu cvs olarak drivedan indirmek için
results_df.to_csv('results_df.csv', index=False)
all_scores_df.to_csv('all_scores_df.csv', index=False)

from google.colab import files
files.download('results_df.csv')
files.download('all_scores_df.csv')
```

B

Fig. 19: En iyi modelin seçimi (A), performans metriklerinin belirlenmesi ve tüm verilerin eldesi (B) için hazırlanan girdi

Outer Fold	Model	Params	accuracy	precision	recall	f1
Fold 1	KNN	{'n_neighbors': 3}	0.62	0.63	0.67	0.65
Fold 1	KNN	{'n_neighbors': 4}	0.61	0.67	0.52	0.59
Fold 1	KNN	{'n_neighbors': 5}	0.63	0.64	0.69	0.66
Fold 1	KNN	{'n_neighbors': 6}	0.63	0.67	0.58	0.62
Fold 1	KNN	{'n_neighbors': 7}	0.64	0.64	0.70	0.67
Fold 1	KNN	{'n_neighbors': 8}	0.64	0.67	0.61	0.64
Fold 1	KNN	{'n_neighbors': 9}	0.64	0.65	0.71	0.68
Fold 1	KNN	{'n_neighbors': 10}	0.64	0.67	0.64	0.65
Fold 1	KNN	{'n_neighbors': 11}	0.65	0.65	0.72	0.68
Fold 1	MLP	{'hidden_layer_sizes': (50,), 'activation': 'relu'}	0.72	0.72	0.76	0.74
Fold 1	MLP	{'hidden_layer_sizes': (50,), 'activation': 'tanh'}	0.71	0.71	0.76	0.74
Fold 1	MLP	{'hidden_layer_sizes': (100,), 'activation': 'relu'}	0.72	0.74	0.71	0.73
Fold 1	MLP	{'hidden_layer_sizes': (100,), 'activation': 'tanh'}	0.71	0.73	0.74	0.73
Fold 1	LinearSVC	{'C': 0.1}	0.64	0.64	0.75	0.69
Fold 1	LinearSVC	{'C': 1}	0.64	0.64	0.75	0.69
Fold 1	LinearSVC	{'C': 10}	0.64	0.64	0.75	0.69
Fold 1	XGBClassifier	{'default': True}	0.72	0.73	0.74	0.74
Fold 2	KNN	{'n_neighbors': 3}	0.62	0.63	0.67	0.65
Fold 2	KNN	{'n_neighbors': 4}	0.61	0.66	0.52	0.59
Fold 2	KNN	{'n_neighbors': 5}	0.63	0.64	0.69	0.66
Fold 2	KNN	{'n_neighbors': 6}	0.62	0.67	0.58	0.62
Fold 2	KNN	{'n_neighbors': 7}	0.64	0.64	0.70	0.67
Fold 2	KNN	{'n_neighbors': 8}	0.63	0.66	0.61	0.64
Fold 2	KNN	{'n_neighbors': 9}	0.64	0.64	0.71	0.67
Fold 2	KNN	{'n_neighbors': 10}	0.64	0.66	0.63	0.65
Fold 2	KNN	{'n_neighbors': 11}	0.64	0.64	0.72	0.68
Fold 2	MLP	{'hidden_layer_sizes': (50,), 'activation': 'relu'}	0.71	0.72	0.73	0.73
Fold 2	MLP	{'hidden_layer_sizes': (50,), 'activation': 'tanh'}	0.71	0.72	0.75	0.73
Fold 2	MLP	{'hidden_layer_sizes': (100,), 'activation': 'relu'}	0.71	0.73	0.73	0.73
Fold 2	MLP	{'hidden_layer_sizes': (100,), 'activation': 'tanh'}	0.71	0.73	0.72	0.72
Fold 2	LinearSVC	{'C': 0.1}	0.64	0.64	0.73	0.68
Fold 2	LinearSVC	{'C': 1}	0.64	0.64	0.73	0.68
Fold 2	LinearSVC	{'C': 10}	0.64	0.64	0.73	0.68
Fold 2	XGBClassifier	{'default': True}	0.71	0.73	0.73	0.73
Fold 3	KNN	{'n_neighbors': 3}	0.61	0.63	0.66	0.64
Fold 3	KNN	{'n_neighbors': 4}	0.61	0.67	0.52	0.58
Fold 3	KNN	{'n_neighbors': 5}	0.63	0.64	0.68	0.66
Fold 3	KNN	{'n_neighbors': 6}	0.62	0.66	0.57	0.62
Fold 3	KNN	{'n_neighbors': 7}	0.63	0.64	0.70	0.67
Fold 3	KNN	{'n_neighbors': 8}	0.63	0.66	0.61	0.63
Fold 3	KNN	{'n_neighbors': 9}	0.64	0.64	0.71	0.67
Fold 3	KNN	{'n_neighbors': 10}	0.63	0.66	0.63	0.65
Fold 3	KNN	{'n_neighbors': 11}	0.64	0.64	0.72	0.68
Fold 3	MLP	{'hidden_layer_sizes': (50,), 'activation': 'relu'}	0.71	0.71	0.74	0.73
Fold 3	MLP	{'hidden_layer_sizes': (50,), 'activation': 'tanh'}	0.70	0.70	0.77	0.73
Fold 3	MLP	{'hidden_layer_sizes': (100,), 'activation': 'relu'}	0.71	0.72	0.73	0.72
Fold 3	MLP	{'hidden_layer_sizes': (100,), 'activation': 'tanh'}	0.70	0.70	0.76	0.73
Fold 3	LinearSVC	{'C': 0.1}	0.64	0.64	0.73	0.68
Fold 3	LinearSVC	{'C': 1}	0.64	0.64	0.73	0.68
Fold 3	LinearSVC	{'C': 10}	0.64	0.64	0.73	0.68
Fold 3	XGBClassifier	{'default': True}	0.71	0.72	0.73	0.72
Fold 4	KNN	{'n_neighbors': 3}	0.61	0.62	0.67	0.64
Fold 4	KNN	{'n_neighbors': 4}	0.61	0.66	0.52	0.58
Fold 4	KNN	{'n_neighbors': 5}	0.62	0.63	0.69	0.66
Fold 4	KNN	{'n_neighbors': 6}	0.62	0.66	0.58	0.62
Fold 4	KNN	{'n_neighbors': 7}	0.63	0.64	0.70	0.67
Fold 4	KNN	{'n_neighbors': 8}	0.63	0.66	0.61	0.64
Fold 4	KNN	{'n_neighbors': 9}	0.64	0.64	0.71	0.67
Fold 4	KNN	{'n_neighbors': 10}	0.63	0.66	0.64	0.65
Fold 4	KNN	{'n_neighbors': 11}	0.64	0.64	0.72	0.68
Fold 4	MLP	{'hidden_layer_sizes': (50,), 'activation': 'relu'}	0.71	0.72	0.73	0.73
Fold 4	MLP	{'hidden_layer_sizes': (50,), 'activation': 'tanh'}	0.71	0.72	0.72	0.72
Fold 4	MLP	{'hidden_layer_sizes': (100,), 'activation': 'relu'}	0.71	0.72	0.73	0.73
Fold 4	MLP	{'hidden_layer_sizes': (100,), 'activation': 'tanh'}	0.70	0.69	0.78	0.73
Fold 4	LinearSVC	{'C': 0.1}	0.64	0.64	0.73	0.68
Fold 4	LinearSVC	{'C': 1}	0.64	0.64	0.73	0.68
Fold 4	LinearSVC	{'C': 10}	0.64	0.64	0.73	0.68
Fold 4	XGBClassifier	{'default': True}	0.71	0.72	0.73	0.73
Fold 5	KNN	{'n_neighbors': 3}	0.61	0.63	0.66	0.64
Fold 5	KNN	{'n_neighbors': 4}	0.61	0.66	0.52	0.58
Fold 5	KNN	{'n_neighbors': 5}	0.62	0.63	0.68	0.65
Fold 5	KNN	{'n_neighbors': 6}	0.62	0.66	0.57	0.61
Fold 5	KNN	{'n_neighbors': 7}	0.63	0.64	0.69	0.66
Fold 5	KNN	{'n_neighbors': 8}	0.63	0.66	0.60	0.63
Fold 5	KNN	{'n_neighbors': 9}	0.63	0.64	0.70	0.67
Fold 5	KNN	{'n_neighbors': 10}	0.63	0.66	0.63	0.64
Fold 5	KNN	{'n_neighbors': 11}	0.64	0.64	0.71	0.67
Fold 5	MLP	{'hidden_layer_sizes': (50,), 'activation': 'relu'}	0.71	0.73	0.72	0.73
Fold 5	MLP	{'hidden_layer_sizes': (50,), 'activation': 'tanh'}	0.71	0.71	0.75	0.73
Fold 5	MLP	{'hidden_layer_sizes': (100,), 'activation': 'relu'}	0.71	0.71	0.75	0.73
Fold 5	MLP	{'hidden_layer_sizes': (100,), 'activation': 'tanh'}	0.71	0.72	0.73	0.73
Fold 5	LinearSVC	{'C': 0.1}	0.64	0.64	0.73	0.68
Fold 5	LinearSVC	{'C': 1}	0.64	0.64	0.73	0.68
Fold 5	LinearSVC	{'C': 10}	0.64	0.64	0.73	0.68
Fold 5	XGBClassifier	{'default': True}	0.71	0.72	0.73	0.73

Fig. 20: Farklı parametrelerin denenerek modellerin performanslarının kıyaslandığı işlemden elde edilen inner CV sonuçlarına ait tüm metrik değerlerini içeren csv tablosu

Fig. 20'de modellerin farklı hiperparametre kombinasyonlarına göre inner CV validasyon sürecinde gösterdikleri accuracy, precision, recall ve f1-score metrikleri kaydedilmiştir. İncelendiğinde, özellikle XGBoost ve MLP modellerinin genel olarak yüksek accuracy ve f1 skorlarına sahip olduğu gözlenmektedir. LinearSVC modeli güçlü accuracy değerlerine sahip olsa da f1 skorları nispeten daha düşüktür. Bu durum sınıflar arası dengesizliklere daha duyarlı olduğunu işaret edebilir. Bu figürde yapılan derleme işleminin, model ve hiperparametre seçiminin sınıflandırma başarımına etkisini görselleştirerek en iyi yapılandırmaların belirlenmesine katkı sağlaması hedeflenmiştir.

Tablo 2: Outer Fold Test Seti Sonuçları

Outer Fold	*Model*	*Best Params*	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*ROC-AUC*
Fold 1	XGBClassifier	{'default': True}	0.7229	0.736322	0.740365	0.738338	0.800361
Fold 2	XGBClassifier	{'default': True}	0.71215	0.725964	0.730707	0.728328	0.788274
Fold 3	XGBClassifier	{'default': True}	0.7134	0.726352	0.733643	0.729979	0.787355
Fold 4	XGBClassifier	{'default': True}	0.71485	0.729263	0.73168	0.730469	0.792645
Fold 5	MLPClassifier	{'hidden_layer_sizes': (50,), 'activation': 'relu'}	0.71595	0.711061	0.778451	0.743232	0.791818

Bu tabloda outer test seti performanslarına göre en başarılı ilk beş model listelenmiştir. Net bir şekilde görüldüğü üzere, XGBoost dört farklı fold'da en yüksek accuracy, f1 skoru ve ROC-AUC değerleriyle öne çıkmaktadır. MLP modeli özellikle 5. fold'da sağladığı yüksek recall ve ROC-AUC değerleriyle dikkat çekmektedir. Bu değerler sırasıyla 0.778 ve 0.7918 olarak kaydedilmiştir. İlk beş modelin her biri tutarlı biçimde yüksek f1 skorları ve ROC-AUC değerleri göstermektedir. Bu durum sınıflar arası ayrımı başarıyla gerçekleştirebildiklerini ortaya koymaktadır.

SONUÇ:

Bu projede, HIGGS veri seti üzerinde kapsamlı bir makine öğrenmesi süreci uygulanarak veri ön işleme, öznitelik seçimi, modelleme ve değerlendirme adımlarının bütüncül bir yapıda nasıl kurgulanabileceği gösterilmiştir. Gerçek hayat verilerini simüle etmek amacıyla veriler rastgele seçilmiş, ardından eksik değerler, aykırılıklar ve ölçekleme gibi problemler analiz edilerek uygun şekilde düzeltilmiştir. Özellik seçiminde ANOVA ve Mutual Information yöntemleriyle farklı boyutlarda öznitelik kümeleri oluşturulmuş, ardından Nested Cross-Validation yapısıyla model başarımı iç ve dış döngülerle değerlendirilmiştir.

Projede KNN, MLP, LinearSVC ve XGBoost gibi dört farklı sınıflandırıcı modellenmiştir. Elde edilen sonuçlara göre, XGBoost algoritması, özellikle k=20 ve Mutual Information seçimiyle, ROC-AUC = 0.8002 ve Accuracy = %72.2 değerleriyle genel olarak en yüksek performansı sergilemiştir. XGBoost'un herhangi bir şekilde hiperparametre değişikliğine gidilmeden, yani sadece default haldeyken bile, sergilediği yüksek başarı, hiperparametrik ayar yapılmadan da güçlü bir model

sunduğunu göstermektedir. MLP modelinin performansı XGBoost modelini benzer değerler ile takip etmiştir.

Fig. 13-16'da bulunan ROC eğrileri aracılığıyla görselleştirilen performans analizleri, seçilen model ve veri temsil biçiminin sınıflandırma sonucuna etkisini ortaya koymuştur. Ne yazık ki, projenin 3B aşamasında ROC grafikleri oluşturulamamıştır. Onun yerine en iyi beş kombinasyon tablo halinde sunulmuş ve ROC-AUC değerleri de o tabloda kaydedilmiştir.

REFERANSLAR:

Scikit Learn, n.d., LinearSVC retrieved from: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html at 27.06.2025

Whiteson, D. (2014). HIGGS [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5V312