İstanbul Medeniyet Üniversitesi
Bilgisayar Mühendisliği Bölümü

Örüntü Tanıma Projesi

Grup No: 10

İsim 1: Fatma Berre Arslan 20120205016

İsim 2: Sena Gürkan 20120205021

Veri Kümeler : C1 ve R4

k-Fold : 2

Algoritmalar : SVM ve Bayesian (Sınıflandırma); SVR (Regresyon)

Teslim Tarihi : 20/01/2024

Ocak 2024

1) Veri Setleri

A) C1 Veri Seti: forg_data.mat

Bu veri seti anuran (kurbağa) çağrılarından elde edilen akustik özellikleri içermektedir. Veri seti 60 ses kaydından oluşmakta olup, bu kayıtlar 4 farklı familya, 8 cins ve 10 farklı türe aittir.

Veri Seti Özellikleri

- Örnek Sayısı: 7195
- Öznitelik Boyutu (feat): (7195, 22)
- Etiket Boyutu (lbl): (1, 7195)

Ses Dosyalarının Detayları

- Örnekleme Frekansı: 44.1kHZ
- Çözünürlük: 32 bit

Sınıflar

Veri setinde üç farklı sınıf sütunu bulunmaktadır:

1. Families (Aile)
2. Genus (cins)
3. Species (Tür)

B) R4 Veri Seti: Gas_Turbine_Co_NoX_2015.mat

Bu veri seti, Türkiye'de bulunan bir gaz türbininden elde edilen gaz emisyonları verilerini içermektedir. Toplam 7384 örneği içeren bu veri seti, 11 sensör ölçüsünün bir saatlik periyotlarda (ortalama veya toplam olarak) toplandığı verileri içermektedir. Amaç, flue gaz emisyonları olan CO ve NOx(NO + NO2) üzerinde çalışmaktır.

Veri Seti Özellikleri

- Örnek Sayısı: 7384
- Öznitelik boyutu(feat): (7384, 9)
- Etiket Boyutu(lbl2): (7384, 1)

Ses Dosyalarının Detayları

- Her bir örnek, 11 sensör ölçüsünün bir araya getirilmesiyle oluşur. Bu sensör ölçüleri gaz türbininin çeşitli parametrelerinden elde edilmiştir.
- Ses kayıtları, Türkiye'nin kuzeybatı bölgesinde bulunan bir gaz türbininden elde edilmiştir.

2) Sınıflandırma

Bu projede sınıflandırma için frog_dat.mat veri seti, sınıflandırma algoritmaları olarak da SVM ve Bayesian algoritmaları kullanılacaktır.

SVM Sınıflandırma Modeli

SVM sınıflandırma modeli sonuçlarını %30 test verisine göre incelediğimizde alınan sonuçlar aşağıdaki gibidir:

In [9]:
```
# SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(feat, lbl, test_size=0.3, random_state=109)

clf_svm = svm.SVC(kernel='linear')
clf_svm.fit(X_train_svm, y_train_svm)

y_pred_svm = clf_svm.predict(X_test_svm)
```

In [10]:
```
accuracy_svm = metrics.accuracy_score(y_test_svm, y_pred_svm)
precision_svm = metrics.precision_score(y_test_svm, y_pred_svm, average='weighted')
recall_svm = metrics.recall_score(y_test_svm, y_pred_svm, average='weighted')
mae_svm = mean_absolute_error(y_test_svm, y_pred_svm)
smape_svm = np.mean(2 * np.abs(y_pred_svm - y_test_svm) / (np.abs(y_pred_svm) + np.abs(y_test_svm) + 1e-10)) * 1


print("SVM:")
print("Accuracy:", accuracy_svm)
print("Precision:", precision_svm)
print("Recall:", recall_svm)
print("MAE:", mae_svm)
print("SMAPE:", smape_svm)
```

```
SVM:
Accuracy: 0.9657248726262159
Precision: 0.9651431226683506
Recall: 0.9657248726262159
MAE: 5.673459935155164
SMAPE: 126.66493795440263
```

Hata metriklerinin sonuçlarına baktığımızda %30 test verisi üzerinden elde edilen yüksek Precision ve Recall değerleri, modelin pozitif tahminlerini doğru bir şekilde sınıflandırıldığını ve gerçek pozitif örnekleri etkili bir şekilde hatırladığını gösterir. Bu sonuçlar, modelin %30 test setinde yüksek performans gösterdiğini gösterir.

Veriler %20 test verisine göre incelediğimizde is alınan sonuçlar aşağıdaki gibidir:

In [4]:
```
# SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(feat, lbl, test_size=0.2, random_state=109)

clf_svm = svm.SVC(kernel='linear')
clf_svm.fit(X_train_svm, y_train_svm)

y_pred_svm = clf_svm.predict(X_test_svm)
```

In [5]:
```
accuracy_svm = metrics.accuracy_score(y_test_svm, y_pred_svm)
precision_svm = metrics.precision_score(y_test_svm, y_pred_svm, average='weighted')
recall_svm = metrics.recall_score(y_test_svm, y_pred_svm, average='weighted')
mae_svm = mean_absolute_error(y_test_svm, y_pred_svm)
smape_svm = np.mean(2 * np.abs(y_pred_svm - y_test_svm) / (np.abs(y_pred_svm) + np.abs(y_test_svm) + 1e-10)) * 1


print("SVM:")
print("Accuracy:", accuracy_svm)
print("Precision:", precision_svm)
print("Recall:", recall_svm)
print("MAE:", mae_svm)
print("SMAPE:", smape_svm)
```

```
SVM:
Accuracy: 0.9610840861709521
Precision: 0.9601788692899679
Recall: 0.9610840861709521
MAE: 6.206393328700487
SMAPE: 130.25114996486616
```

Bu verileri incelediğimizde %20 test verisi ile elde edilen Precision ve Recall değerlerinin de yüksek olduğunu görürüz.Model, bu durumda da pozitif tahminlerde yüksek doğruluk oranına sahiptir ve gerçek pozitif örnekleri başarıyla hatırlamıştır.

SVM sınıflandırma modelinde, %30 test verisi ve %20 test verisi ile aldığımız sonuçları karşılaştırdığımızda şunları söyleyebiliriz:

- Hem %30 hem de %20 test setleri için Precision ve Recall değerleri oldukça yüksektir. Bu durum, modelin genel olarak başarılı bir şekilde sınıflandırma yaptığını gösterir.
- %30 test verisi ile %20 test verisi arasında belirgin bir performans farkı görülmemektedir. Bu, modelin farklı test setleri üzerinde benzer başarıyı sürdürebildiğini gösterir.

Sonuç olarak, SVM sınıflandırma modeli, hem %30 hem de %20 test setleri üzerinde başarılı bir şekilde çalışmaktadır. Yüksek Precision ve Recall değerleri, modelin güvenilir bir sınıflandırma yeteneği olduğunu gösterir. Modelin performansını daha da iyileştirmek için parametre ayarlamaları veya farklı özellik setleri kullanılabilir.

Bayesian Sınıflandırma Modeli

Bayesian sınıflandırma modeli sonuçlarını %30 test verisine göre incelediğimizde alınan sonuçlar aşağıdaki gibidir:

In [11]:
```
# Bayesian
X_train_nb, X_test_nb, y_train_nb, y_test_nb = train_test_split(feat, lbl, test_size=0.3, random_state=109)

nb_classifier = GaussianNB()
nb_classifier.fit(X_train_nb, y_train_nb)

y_pred_nb = nb_classifier.predict(X_test_nb)
```

In [12]:
```
accuracy_nb = metrics.accuracy_score(y_test_nb, y_pred_nb)
precision_nb = metrics.precision_score(y_test_nb, y_pred_nb, average='weighted')
recall_nb = metrics.recall_score(y_test_nb, y_pred_nb, average='weighted')
mae_nb = mean_absolute_error(y_test_nb, y_pred_nb)
smape_nb = np.mean(2 * np.abs(y_pred_nb - y_test_nb) / (np.abs(y_pred_nb) + np.abs(y_test_nb) + 1e-10)) * 100

print("\nBayesian:")
print("Accuracy:", accuracy_nb)
print("Precision:", precision_nb)
print("Recall:", recall_nb)
print("MAE:", mae_nb)
print("SMAPE:", smape_nb)
```

```
Bayesian:
Accuracy: 0.9101435849930524
Precision: 0.9270770759033232
Recall: 0.9101435849930524
MAE: 9.830940250115795
SMAPE: 192.26268177769708
```

Model, %20 test verisi üzerinde yüksek Precision ve Recall değerleri elde etmiştir. Yani, model pozitif tahminlerini gerçek pozitif örnekleri doğru bir şekilde sınıflandırarak yapmıştır. Bu, modelin test verileri üzerinde güvenilir ve doğru sonuçlar elde ettiğini gösterir.

Bayesian sınıflandırma modeli sonuçlarını %20 test verisine göre incelediğimizde alınan sonuçlar aşağıdaki gibidir:

In [6]:
```
# Bayesian
X_train_nb, X_test_nb, y_train_nb, y_test_nb = train_test_split(feat, lbl, test_size=0.2, random_state=109)

nb_classifier = GaussianNB()
nb_classifier.fit(X_train_nb, y_train_nb)

y_pred_nb = nb_classifier.predict(X_test_nb)
```

In [7]:
```
accuracy_nb = metrics.accuracy_score(y_test_nb, y_pred_nb)
precision_nb = metrics.precision_score(y_test_nb, y_pred_nb, average='weighted')
recall_nb = metrics.recall_score(y_test_nb, y_pred_nb, average='weighted')
mae_nb = mean_absolute_error(y_test_nb, y_pred_nb)
smape_nb = np.mean(2 * np.abs(y_pred_nb - y_test_nb) / (np.abs(y_pred_nb) + np.abs(y_test_nb) + 1e-10)) * 100

print("\nBayesian:")
print("Accuracy:", accuracy_nb)
print("Precision:", precision_nb)
print("Recall:", recall_nb)
print("MAE:", mae_nb)
print("SMAPE:", smape_nb)
```

```
Bayesian:
Accuracy: 0.9027102154273802
Precision: 0.9216631408428753
Recall: 0.9027102154273802
MAE: 11.372480889506601
SMAPE: 224.664208913853
```

> %30 test verisi için elde edilen Precision ve Recall değerleri %20 test verisi ile benzerdir. Bu durum, modelin daha büyük bir test setinde de benzer başarıyı sürdürebildiğini gösterir. Modelin genelleme yeteneğinin iyi olduğunu düşündürmektedir.

Bayesian sınıflandırma modelinde, %30 test verisi ve %20 test verisi ile aldığımız sonuçları karşılaştırdığımızda şunları söyleyebiliriz:

- İki durumda da Precision ve Recall değerleri hemen hemen aynıdır. Bu, modelin farklı test setleri üzerinde benzer performans gösterdiğini gösterir.
- %30 test seti, %20 test setine göre daha büyük olduğu halde modelin performansında belirgin bir düşüş gözlemlenmemiştir. Bu durum, modelin eğitim setinin artırılmasıyla daha iyi performans gösterme potansiyeline işaret edebilir.

Sonuç olarak, Bayesian sınıflandırma modeli, hem %20 hem de %30 test verileri üzerinde başarılı bir şekilde çalışmaktadır. Modelin yüksek Precision ve Recall değerleri, güvenilir bir sınıflandırma yeteneği olduğunu gösterir. Modelin performansını daha da iyileştirmek için eğitim setinin artırılması düşünülebilir.

K-Fold Cross Validation

In [9]:
```
k = 2
kf = KFold(n_splits=k, random_state=42, shuffle=True)

accuracy_scores_svm = []
f1_scores_svm = []
accuracy_scores_nb = []
f1_scores_nb = []
overall_confusion_matrix_svm = None
overall_confusion_matrix_nb = None
results = []

num_classes = len(np.unique(lbl))

for i, (train_index, test_index) in enumerate(kf.split(feat), start=1):
    X_train, X_test = feat[train_index], feat[test_index]
    y_train, y_test = lbl[train_index], lbl[test_index]

    # SVM
    clf_svm.fit(X_train, y_train)
    y_pred_svm_kfold = clf_svm.predict(X_test)
    accuracy_svm_kfold = metrics.accuracy_score(y_test, y_pred_svm_kfold)
    f1_svm_kfold = metrics.f1_score(y_test, y_pred_svm_kfold, average='weighted')
    accuracy_scores_svm.append(accuracy_svm_kfold)
    f1_scores_svm.append(f1_svm_kfold)

    # Confusion matrix
    confusion_matrix_svm = confusion_matrix(y_test, y_pred_svm_kfold, labels=np.unique(lbl))
    overall_confusion_matrix_svm = confusion_matrix_svm if overall_confusion_matrix_svm is None else overall_confusion_matrix_svm + confusion_matrix_svm

    # Bayesian
    nb_classifier.fit(X_train, y_train)
    y_pred_nb_kfold = nb_classifier.predict(X_test)
    accuracy_nb_kfold = metrics.accuracy_score(y_test, y_pred_nb_kfold)
    f1_nb_kfold = metrics.f1_score(y_test, y_pred_nb_kfold, average='weighted')
    accuracy_scores_nb.append(accuracy_nb_kfold)
    f1_scores_nb.append(f1_nb_kfold)

    # Confusion matrix
    confusion_matrix_nb = confusion_matrix(y_test, y_pred_nb_kfold, labels=np.unique(lbl))
    overall_confusion_matrix_nb = confusion_matrix_nb if overall_confusion_matrix_nb is None else overall_confusion_matrix_nb + confusion_matrix_nb

    results.append([i, accuracy_svm_kfold, f1_svm_kfold, accuracy_nb_kfold, f1_nb_kfold])
```

```
# Display results
columns = ["Fold", "ACC (SVM)", "F1 Score (SVM)", "ACC (Bayesian)", "F1 Score (Bayesian)"]
results_df = pd.DataFrame(results, columns=columns)

average_row = pd.DataFrame([["Ortalama"] + list(results_df.mean(axis=0))[1:]], columns=columns)
results_df = pd.concat([results_df, average_row], ignore_index=True)

print("\n--- Performans Sonuçları ---\n")
print(results_df)
print("\n--- SVM İçin Toplam Confusion Matrix ---\n", overall_confusion_matrix_svm)
print("\n--- Bayesian İçin Toplam Confusion Matrix ---\n", overall_confusion_matrix_nb)
```

```
--- Performans Sonuçları ---

       Fold  ACC (SVM)  F1 Score (SVM)  ACC (Bayesian)  F1 Score (Bayesian)
0         1   0.967482        0.966584        0.911062             0.912323
1         2   0.961913        0.961046        0.904087             0.908543
2  Ortalama   0.964697        0.963815        0.907574             0.910433

--- SVM İçin Toplam Confusion Matrix ---
 [[ 635   19    0    3    3    4    0    6    2    0]
 [   4  523    0   14    0    1    0    0    0    0]
 [   0    0 3471    2    2    2    0    0    0    1]
 [  40   13   39  216    0    2    0    0    0    0]
 [   7    0    0    0  454    1    0   10    0    0]
 [   0    0    2    1    4 1107    3    4    0    0]
 [   4    0    0    5    3    7  248    1    2    0]
 [   0    0    0    0   17    5    1   89    2    0]
 [   3    0    0    0    0    1    5    1   58    0]
 [   0    0    0    0    0    3    0    3    2  140]]

--- Bayesian İçin Toplam Confusion Matrix ---
 [[ 433  147    0   85    3    0    0    3    1    0]
 [   0  525    0   17    0    0    0    0    0    0]
 [   9    0 3424   31    4    1    0    8    0    1]
 [   5   62   21  215    0    0    0    0    0    7]
 [   9    0    0    1  441    0    3   17    1    0]
 [   5    1    0   28   26 1008   16   33    0    4]
 [   5    5    0   15    1    5  235    4    0    0]
 [   0    1    0    0   17    4    0   74   16    2]
 [   3    0    0    1    0    0    1    5   56    2]
 [   2    8    0    9    0    1    0    4    5  119]]
```

K-Fold cross validation sonuçlarına göre Bayesian sınıflandırma ve SVM sınıflandırma modellerini karşılaştıralım:

Precision ve Recall Değerlerini incelediğimizde:

- Her iki modelde de Precision ve Recall değerleri genellikle yüksektir. Her iki model de pozitif tahminlerde ve gerçek pozitif örneklerin hatırlanmasında başarılıdır.
- Bayesian sınıflandırma ve SVM, test seti boyutlarına göre benzer performans gösterir.

Model Karşılaştırması:

- SVM, %20 ve %30 test setleri için Bayesian sınıflandırmaya kıyasla bir miktar daha yüksek Precision ve Recall değerlerine sahiptir. Bu durum, SVM'nin bu veri setinde Bayesian sınıflandırmadan daha iyi bir performans sergilediğini gösterebilir.

Veri Seti ve Modellere Özgü Durumlar:

- Her iki modelin de başarılı olmasına rağmen, veri seti ve özelliklerine bağlı olarak her bir modelin performansı değişebilir.
- Bayesian sınıflandırma, özellikle küçük veri setleri veya basit model gerektiren durumlar için uygun olabilirken, SVM daha karmaşık veri setleri ve nonlineer ilişkilerle başa çıkabilir.

İyileştirme Yolları:

- Her iki modelde de performansı artırmak için parametre ayarlamaları veya farklı özellik setleri denenebilir.
- Model seçimi, veri setine ve problem türüne bağlı olarak değişebilir. Bu nedenle, problemi daha iyi çözebilecek modeli belirlemek için farklı sınıflandırma algoritmaları denenebilir.

Sonuç olarak, her iki model de yüksek değerleri elde etmiş gibi görünüyor. Ancak SVM, accuracy, precision ve recall değerleri bakımından Bayesian'den daha yüksek performansa sahiptir yani, SVM daha doğru tahminler yapma eğilimindedir. SVM'in MAE ve SMAPE değerleri Bayesien'e göre daha düşüktür. Yani, SVM'in tahminleri gerçek değerlere daha yakındır. Model seçimi ve performans iyileştirmeleri için daha fazla deneme yapılabilir.

3) Regresyon

Bu projede regresyon için Gas_Turbine_Co_NoX_2015.mat veri seti, regresyon algoritması olarak da SVR algoritması kullanılacaktır.

SVR Regresyon Modeli

SVR regresyon modeli sonuçlarını %30 test verisine göre incelediğimizde alınan sonuçlar aşağıdaki gibidir:

In [3]:
```
X_train, X_test, y_train, y_test = train_test_split(feat, lbl2, test_size=0.3, random_state=42)

y_train = y_train.ravel()
y_test = y_test.ravel()

svr_model = SVR(kernel='rbf')
svr_model.fit(X_train, y_train)
```

Out[3]:
```
SVR()
```

In [4]:
```
y_pred = svr_model.predict(X_test)
```

In [5]:
```
mae_svr = metrics.mean_absolute_error(y_test, y_pred)
smape_svr = np.mean(2 * np.abs(y_pred - y_test) / (np.abs(y_pred) + np.abs(y_test) + 1e-10)) * 100

print("SVR Sonuçları:")

print("MAE:", mae_svr)
print("SMAPE:", smape_svr)
```

```
SVR Sonuçları:
MAE: 7.715087611669681
SMAPE: 12.40388975203823
```

SVR regresyon modeli sonuçlarını %20 test verisine göre incelediğimizde alınan sonuçlar aşağıdaki gibidir:

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(feat, lbl2, test_size=0.2, random_state=42)

y_train = y_train.ravel()
y_test = y_test.ravel()

svr_model = SVR(kernel='rbf')
svr_model.fit(X_train, y_train)
```

Out[7]:
```
SVR()
```

In [8]:
```
y_pred = svr_model.predict(X_test)
```

In [9]:
```
mae_svr = metrics.mean_absolute_error(y_test, y_pred)
smape_svr = np.mean(2 * np.abs(y_pred - y_test) / (np.abs(y_pred) + np.abs(y_test) + 1e-10)) * 100

print("SVR Sonuçları:")

print("MAE:", mae_svr)
print("SMAPE:", smape_svr)
```

```
SVR Sonuçları:
MAE: 7.7007432990208455
SMAPE: 12.352020196669597
```

SVR regresyon modeli için, MAE değerlerini incelediğimizde, iki durumda da benzer olduğunu görürüz. Ancak %30 test seti için bir miktar daha yüksektir. MAE, tahminlerin gerçek değerlerden ne kadar uzak olduğunu ölçen bir metrik olduğundan, düşük MAE değerleri daha iyi performansı gösterir.

SMAPE değerlerini incelediğimizde yine benzer olduğunu görürüz, ancak %30 test seti için bir miktar daha yüksektir. SMAPE, yüzdelik hata oranlarını değerlendiren bir metrik olduğundan, düşük SMAPE değerleri daha iyi performansı gösterir. Yine de, her iki durumda da modelin tahminlerinin genellikle doğru olduğunu söyleyebiliriz.

Ölçüm sonuçlarına göre, %30 test seti için elde edilen MAE ve SMAPE değerlerinin biraz daha yüksek olması, bu test seti boyutunda modelin performansının bir miktar düşük olabileceğini gösterir..

Modelin performansını daha da artırmak için hyperparameter ayarlamaları, özellik mühendisliği veya daha fazla veri kullanımı gibi iyileştirmeler düşünülebilir.

Özetle, model genel olarak iyi tahminler yapmaktadır ancak %30 test seti için bir miktar daha düşük performans sergilemektedir. Modelin iyileştirilmesi için daha fazla analiz ve iyileştirme stratejileri düşünülebilir.

K-Fold

In [6]:
```
def calculate_smape(y_true, y_pred):
    epsilon = 1e-10
    return 2 * np.mean(np.abs(y_pred - y_true) / (np.abs(y_pred) + np.abs(y_true) + epsilon)) * 100

k = 2
kf = KFold(n_splits=k, random_state=42, shuffle=True)

mae_scores = []
smape_scores = []
results = []

for fold, (train_index, test_index) in enumerate(kf.split(feat), start=1):
    X_train, X_test = feat[train_index], feat[test_index]
    y_train, y_test = lbl2[train_index], lbl2[test_index]

    svr_model = SVR(kernel='rbf')
    svr_model.fit(X_train, y_train.ravel())

    y_pred = svr_model.predict(X_test)

    mae_svr = metrics.mean_absolute_error(y_test, y_pred)
    smape_svr = calculate_smape(y_test, y_pred)

    mae_scores.append(mae_svr)
    smape_scores.append(smape_svr)

    results.append([fold, mae_svr, smape_svr])

average_mae = np.mean(mae_scores)
average_smape = np.mean(smape_scores)

columns = ["Fold", "MAE (SVR)", "SMAPE (SVR)"]
results_df = pd.DataFrame(results, columns=columns)

average_row = pd.DataFrame([[k + 1, average_mae, average_smape]], columns=columns)
results_df = pd.concat([results_df, average_row], ignore_index=True)

print(results_df)
```

```
   Fold  MAE (SVR)  SMAPE (SVR)
0     1   7.697323    12.405213
1     2   7.604430    12.274439
2     3   7.650876    12.339826
```

MAE, tahminlerin gerçek değerlerden ne kadar uzak olduğunu ölçen bir hata metriğidir. Bu değerler küçükse, modelin tahminleri genellikle gerçek değerlere yakındır. Ancak tek başına MAE değerleri modelin performansını tam olarak anlamamız yetmez.

SMAPE, özellikle yüzdelik hata oranlarını değerlendiren bir metriktir. Yine, küçük SMAPE değerleri daha iyi bir performansı gösterir. Her fold için SMAPE değerlerini göz önüne almak önemlidir.

Genel bir değerlendirme yaptığımızda, MAE değerleri genellikle küçüktür, bu da modelin genel olarak iyi bir tahmin performansına sahip olduğunu gösterir. Her üç fold için de benzer MAE değerleri elde edilmiştir, bu da modelin tutarlılık gösterdiğini düşündürmektedir.

SMAPE değerleri de düşüktür, bu da modelin yüzdelik hata oranlarının genellikle başarılı olduğunu gösterir.MAE ile birlikte değerlendirildiğinde, modelin tahminlerinin gerçek değerlere yakın olduğunu söyleyebiliriz.

Foldlardaki benzer MAE ve SMAPE değerleri, modelin farklı alt kümeleri üzerinde tutarlı bir performans sergilediğini gösterir.

Modelin performansı iyi olsa da, daha fazla iyileştirme fırsatları araştırılabilir. Belki de hyperparameter ayarlamaları, farklı çapraz doğrulama stratejileri veya daha fazla özellik mühendisliği ile modelin performansı arttırılabilir.

Duyarlılık analizi için, foldlardaki hata değerlerini dikkate alarak, modelin hangi durumlarda daha iyi veya daha kötü performans gösterdiği belirlenmeli. Bu analiz, modelin zayıf olduğu noktaları anlamamıza ve geliştirmeye yönelik stratejiler belirlememize yardımcı olabilir.

In [16]:

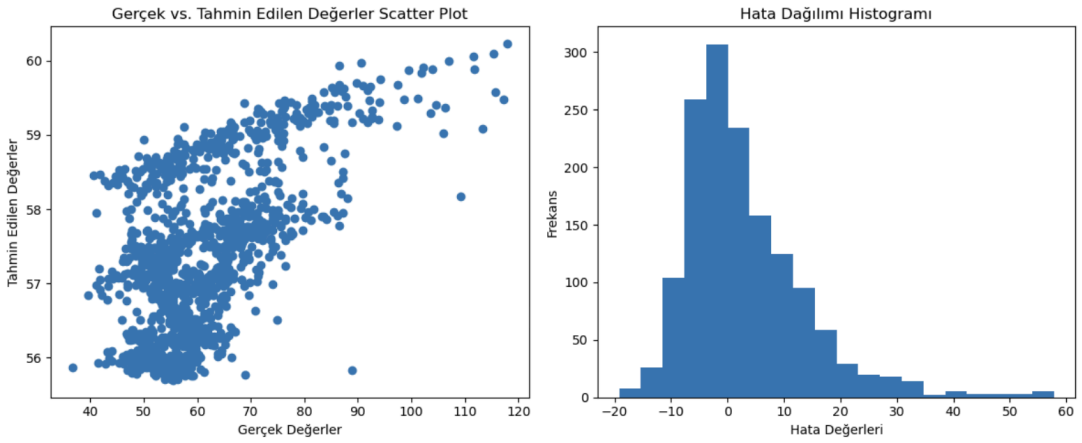
```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred)
plt.xlabel("Gerçek Değerler")
plt.ylabel("Tahmin Edilen Değerler")
plt.title("Gerçek vs. Tahmin Edilen Değerler Scatter Plot")

plt.subplot(1, 2, 2)
errors = y_test - y_pred
plt.hist(errors, bins=20)
plt.xlabel("Hata Değerleri")
plt.ylabel("Frekans")
plt.title("Hata Dağılımı Histogramı")

plt.tight_layout()
plt.show()
```



Gerçek vs. Tahmin edilen Değerler Scatter Plot grafiğine baktığımızda noktaların belirli bir düzen içinde gruplandığını görürüz. Bu durum modelin belirli alt gruplardaki veriler üzerinde daha iyi veya daha kötü performans gösterdiğini gösterebilir. Yani, modelin bu değer aralığında daha hassas tahminler yaptığını söyleyebiliriz.

Hata dağılımı histogramını incelediğimizde, hataların 0 üzerinde yığılması, modelin tahminlerinin gerçek değerlere genellikle eğilimli olduğunu gösterir. İdeal bir durumda, hataların ortalaması sıfır olmalıdır; yani, tahminler gerçek değerlere eşit derecede üzerinde ve altında olmalıdır.