
T.C.

**ONDOKUZ MAYIS ÜNİVERSİTESİ**

**LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ**

**BİLGİSAYAR MÜHENDİSLİĞİ**

# ÖRÜNTÜ TANIMA-BM605

## Feature Extraction of Images Analiz Raporu

**FIRAT KAAN BİTMEZ - 23281855**

**SAMSUN, 2024-2025 Eğitim Öğretim Yılı Güz Yarıyılı**

# 1. Giriş

Örüntü tanıma ve bilgisayarla görme alanlarında, görsel verilerden anlamlı bilgi çıkarmak en temel amaçlardan biridir. Bizde bu çalışmamızda görsel verilerden bilgi çıkarma üzerinde çalışmalar yapacağız. Bu kapsamda karakter tanıma (Optical Character Recognition, OCR), görüntü tabanlı bilgilerin metin formuna dönüştürülmesi için sıkça başvurulan ve köklü uygulamalara sahip bir problemdir. Özellikle el yazısı rakam tanıma, finans dokümanlarının otomatik işlenmesi, araç plakalarının okunması, çek tanıma ve formların otomatik işlenmesi gibi pek çok alanda büyük önem taşımaktadır.

El yazısı karakter tanıma probleminde karşılaşılan temel güçlükler şöyledir:

- **Kişiden kişiye değişen yazım stili:** Aynı rakam farklı kişilerce çok farklı biçimlerde yazılabilir.
- **Gürültü ve bozulmalar:** Görüntüler üzerinde lekeler, bulanıklık, düşük çözünürlük, değişen kontrast gibi problemler olabilir.
- **Sınıflar arası benzerlik:** Özellikle “2” ve “3” gibi şekilsel olarak benzer rakamlar arasında ayrım yapmak zordur.
- **Farklı aydınlatma ve kalem kalınlıkları:** Bu durum dokusal çeşitliliği artırır ve sınıflandırmayı zorlaştırır.

Ham piksel değerlerine dayalı doğrudan sınıflandırma, bu zorluklar karşısında yetersiz kalabilir. Bu noktada devreye **dokusal özellik çıkarma yöntemleri** girer. Dokusal özellikler, görüntüdeki desenleri, piksel dağılımlarını, tekrarları, yerel yapı ve istatistikleri hesaba katar. Böylece görüntü, doğrudan piksellerle temsil edilmek yerine daha ayrıştırıcı, gürültüye dayanıklı ve istatistiksel olarak anlamlı vektörlerle ifade edilir.

Bu çalışmada, el yazısı rakam tanıma problemi için MNIST veri seti kullanılarak dört farklı dokusal özellik çıkarma yöntemi incelenmiştir:

1. **Gray Level Co-occurrence Matrix (GLCM):** Piksel çiftlerinin belirli uzaklık ve açı yönünde birlikte görülme olasılığını analiz ederek kontrast, enerji, homojenlik, korelasyon gibi istatistikler çıkarır.
2. **Local Binary Pattern (LBP):** Her pikselin komşularına göre parlaklık ilişkisini ikili bir kodla ifade eder ve bu kodların histogramını dokusal imza olarak kullanır.
3. **Local Binary Gray Level Co-occurrence Matrix (LBGLCM):** Önce LBP ile yerel desen bilgisi elde edilir, ardından bu LBP çıktısı üzerinden GLCM hesaplanarak hem yerel desen hem de gri seviye ortak dağılımları bir araya getirilir.
4. **Gray Level Run Length Matrix (GLRLM):** Belirli yönlerde aynı gri seviyeyi ardışık taşıyan piksel dizilerinin (run) istatistiklerini inceleyerek görüntünün tekrar eden yapısını ortaya koyar.

Elde edilen dokusal özellik vektörleri beş farklı makine öğrenimi sınıflandırıcısı ile test edilmiştir:

- Destek Vektör Makineleri (SVM)
- k-En Yakın Komşu (K-NN)
- Rastgele Orman (Random Forest)
- Lojistik Regresyon (LR)
- Karar Ağacı (Decision Tree)

Amaç, bu dokusal özellik setlerinin ve sınıflandırıcıların MNIST veri setindeki performansını karşılaştırmaktır. Böylece dokusal yaklaşımın el yazısı rakam tanımadaki potansiyeli ve sınırlılıkları değerlendirilmiştir.

## 2. Kullanılan Veri Seti

Bu çalışmada **MNIST** (Modified National Institute of Standards and Technology) el yazısı rakam veri seti kullanılmıştır. MNIST veri seti, makine öğrenimi ve bilgisayarla görme alanında standart bir benchmark olup şu özelliklere sahiptir:

- **Sınıflar:** 0'dan 9'a kadar toplam 10 farklı rakam sınıfı.
- **Görüntü Boyutu:** Her örnek 28x28 piksel, gri seviye formatında.
- **Eğitim Seti:** 60.000 örnek
- **Test Seti:** 10.000 örnek
- **Piksel Değerleri:** 0 (siyah) ile 255 (beyaz) arasında gri seviye değerleri.

**Veri Setinin Temini ve Formatı:** Bu proje kapsamında, Kaggle üzerinde bulunan CSV formatındaki MNIST veri seti kullanılmıştır (Kaynak: https://yann.lecun.com/exdb/mnist/). Bu formatta:

- **mnist_train.csv:** İlk sütun etiket (label), sonraki 784 sütun ise 28x28 pikselin satır-satır düzleştirilmiş formudur.
- **mnist_test.csv:** Benzer yapıdadır.

Bu dosyalarda her satır bir rakam örneğini temsil eder. İlk sütun 0-9 arasındaki sınıf etiketini, geri kalan sütunlar ise pikselleri içerir. Her piksel 0-255 arası tamsayı bir değerdir.

**Veri İşleme Adımları:**

1. **Veri Yükleme: pandas** kütüphanesi ile CSV dosyaları okunur.
2. **Etiket ve Piksel Ayrımı:** İlk sütun etiketlere, geri kalan sütunlar piksel matrisine dönüştürülür.
3. **Matris Şekillendirme:** Her satırdaki 784 piksel değeri (28x28=784) yeniden 28x28'lik bir matrise şekillendirilir. Bu sayede görüntü üzerinde dokusal özellikler (GLCM, LBP vb.) hesaplanabilir.
4. **Normalizasyon (Opsiyonel):** Piksel değerleri gerekirse [0,1] aralığına normalize edilebilir, ancak GLCM gibi yöntemler gri seviye bilgisine direkt ihtiyaç duyabildiğinden bu adım isteğe bağlı tutulmuştur.
5. **Eğitim/Test Ayrımı:** Veri setinin kendisi zaten eğitim (train) ve test (test) olarak ayrılmıştır. Bu ayrım korunmuştur.

### 3. Özellik Çıkarma Yöntemleri

Bu bölümde dokusal özellikleri (texture features) çıkarma süreçleri ayrıntılı olarak ele alınacaktır. Amacımız, ham piksel değerlerinden öte, görüntünün yapısını, desenini, gri seviye dağılımlarını ve tekrar eden örüntülerini istatistiksel olarak ifade eden öznitelikler elde etmektir. Böylece el yazısı rakam gibi benzer biçimli karakterlerin bile belli dokusal örüntülerle birbirinden ayrılabilirliği artırılmaya çalışılır.

Ele aldığımız dört temel dokusal özellik çıkarma yöntemi şunlardır:

1. Gray Level Co-occurrence Matrix (GLCM)
2. Local Binary Pattern (LBP)
3. Local Binary Gray Level Co-occurrence Matrix (LBGLCM)
4. Gray Level Run Length Matrix (GLRLM)

Aşağıda her bir yöntem teori, matematiksel temel ve kod örneği ile birlikte açıklanmaktadır.

### 3.1 Gray Level Co-occurrence Matrix (GLCM)

**Teorik Temel:**

GLCM, bir görüntüdeki piksel çiftlerinin belirli bir uzaklık (d) ve yön (θ) için gri seviye birlikteliklerini temsil eder. Gri seviye seviyeleri 0'dan L-1'e (L: gri seviye sayısı) kadar kodlanmış olsun. GLCM, C(i,j) ile ifade edilirse, i ve j gri seviyeleri olmak üzere:

$$C(i,j) = \sum_{x=1}^{M} \sum_{y=1}^{N} \begin{cases} 1 & \text{eğer } I(x,y) = i \text{ ve } I(x+\Delta x, y+\Delta y) = j \\ 0 & \text{aksi halde} \end{cases}$$

Burada (M,N) görüntü boyutlarıdır, (Δx,Δy) ise seçilen uzaklık ve yöndür. Örneğin d=1 ve θ=0° için Δx=0,Δy=1olabilir. Yani her piksel için sağındaki pikselle olan gri seviye ilişkisi incelenir.

GLCM'den elde edilebilecek önemli istatistiksel özellikler şunlardır (Haralick özellikleri olarak da bilinir):

- **Kontrast (Contrast):** Gri seviye farklarını ölçer. Yüksek kontrast değeri, komşu pikseller arasında büyük gri seviye farkları olduğunu gösterir.

$$\text{Contrast} = \sum_{i,j} (i-j)^2 C(i,j)$$

- **Enerji (Energy) veya Angular Second Moment (ASM):** Matrisin yoğunlaşma derecesini ölçer. Tek bir gri seviye çiftine yoğunlaşma varsa enerji yüksek olur.

$$\text{Energy} = \sum_{i,j} C(i,j)^2$$

- **Homojenlik (Homogeneity):** Diyagonale yakın elemanların yoğunluğunu ölçer. Pikseller arasındaki gri seviye farkları azaldıkça homojenlik artar.

$$\text{Energy} = \sum_{i,j} C(i,j)^2$$

- **Korelasyon (Correlation):** Gri seviyeler arasındaki doğrusal ilişkiyi ölçer.

$$\text{Correlation} = \frac{\sum_{i,j}(ij)C(i,j) - \mu_i\mu_j}{\sigma_i\sigma_j}$$

Burada μi,μj satır ve sütun ortalamaları, σi,σj ise standart sapmalardır.

**Kod Uygulaması:**

Aşağıdaki Python kodu (scikit-image kütüphanesi) ile bir 2D gri seviye görüntü için GLCM matrisini ve özelliklerini elde edebiliriz.

```
from skimage.feature import graycomatrix, graycoprops

def extract_glcm_features(image, distances=[1], angles=[0]):
    # GLCM hesaplama
    glcm = graycomatrix(image, distances=distances, angles=angles, levels=256,
                        symmetric=True, normed=True)
    # Özellikler
    contrast = graycoprops(glcm, 'contrast')[0, 0]
    dissimilarity = graycoprops(glcm, 'dissimilarity')[0, 0]
    homogeneity = graycoprops(glcm, 'homogeneity')[0, 0]
    energy = graycoprops(glcm, 'energy')[0, 0]
    correlation = graycoprops(glcm, 'correlation')[0, 0]
    asm = graycoprops(glcm, 'ASM')[0, 0]
    return [contrast, dissimilarity, homogeneity, energy, correlation, asm]
```

Bu fonksiyonla elde ettiğimiz 6 boyutlu öznitelik vektörü, görüntünün gri seviye ilişkilerini istatistiksel olarak temsil eder.

### 3.2 Local Binary Pattern (LBP)

**Teorik Temel:**

LBP, her pikselin etrafındaki belirli yarıçap ve nokta sayısına göre komşu piksellerinin parlaklık ilişkisini ikili (binary) bir kod olarak özetler. Temel formülasyon şu şekildedir:

Bir pikselin değeri Ic ve etrafındaki P adet komşu piksel Ip olsun (bu pikseller belirli bir yarıçap R ile örneklenir). LBP değeri:

$$LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(I_p - I_c)2^p$$

Burada s(x)=1 eğer x≥0 ise, aksi halde s(x)=0

Ortaya çıkan LBP değerleri bir histogramda toplanır. Bu histogram, görüntünün yerel dokusal örüntüsünü özetler. “uniform” LBP, histogramı daha anlamlı kılmak için sınıflandırma yapar ve benzer desenlere sahip LBP değerlerini tek gruplar altında toplar.

**Matematiksel Örnek:**

Diyelim ki merkez pikselin değeri 100 olsun. Etrafındaki 8 piksel (P=8) değerleri [102, 98, 110, 90, 120, 88, 99, 105] şeklinde olsun. Her komşu piksel, merkezden büyükse 1, küçükse 0 olarak kodlanır:

- 102 > 100 → 1
- 98 < 100 → 0
- 110 > 100 → 1
- 90 < 100 → 0
- 120 > 100 → 1
- 88 < 100 → 0
- 99 < 100 → 0
- 105 > 100 → 1

Binary kod: 10101001 (2^7 + 2^5 + 2^3 + 2^0 gibi hesaplanır)

Bu ikili sayı onluk sisteme çevrilerek pikselin LBP değeri elde edilir.

**Kod Uygulaması:**

```
from skimage.feature import local_binary_pattern

def extract_lbp_features(image, radius=3, n_points=24):
    lbp = local_binary_pattern(image, n_points, radius, method='uniform')
    # LBP histogramını çıkar
    (hist, _) = np.histogram(lbp.ravel(),
                             bins=np.arange(0, n_points + 3),
                             range=(0, n_points + 2))
    # Normalize et
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-6)
    return hist
```

Bu fonksiyon ile elde edilen histogram, görüntünün yerel dokusal desenlerinin istatistiksel dağılımını temsil eder.

### 3.3 Local Binary Gray Level Co-occurrence Matrix (LBGLCM)

**Teorik Temel:**

LBGLCM, LBP çıktısını kullanarak GLCM hesaplama fikrine dayanır. İlk adımda LBP ile elde edilen yerel dokusal bilgi, görüntünün her pikseline bir “desen değeri” atar. Daha sonra, bu LBP ile elde edilen yeni “gri seviye uzayında” GLCM hesaplanır. Böylece, sadece orijinal

gri seviyeler arasındaki ilişki değil, aynı zamanda yerel desenlerin birlikte bulunma istatistikleri de yakalanır.

Matematiksel olarak bakarsak, önce LBP ile her piksele LBPP,R(x,y) değeri atanır. Artık elimizde yeni bir "LBP görüntüsü" vardır. Bu LBP değerleri 0'dan belirli bir maximum değere kadar kodlanır (uniform LBP'de bu değer düşüktür). Ardından GLCM şu şekilde tanımlanır:

$$C_{LBP}(i,j) = \sum_{x,y} \begin{cases} 1 & LBP_{P,R}(x,y) = i \text{ ve } LBP_{P,R}(x+\Delta x, y+\Delta y) = j \\ 0 & \text{aksi} \end{cases}$$

Bu sayede dokusal desenlerin birlikte görülme olasılıkları çıkarılır.

LBGLCM'den genellikle GLCM'de olduğu gibi kontrast, enerji, homojenlik gibi özellikler hesaplanır. Bu, LBP'nin yerel desene duyarlılığı ile GLCM'nin ikili ilişkileri yakalama gücünü birleştirir.

**Kod Uygulaması:**

```
def extract_lbglcm_features(image, radius=3, n_points=24):
    # Önce LBP hesapla
    lbp = local_binary_pattern(image, n_points, radius,
method='uniform').astype(np.uint8)
    # LBP seviyeleri genellikle sınırlıdır, levels parametresi buna göre
ayarlanabilir
    glcm = graycomatrix(lbp, distances=[1], angles=[0], levels=256,
symmetric=True, normed=True)
    contrast = graycoprops(glcm, 'contrast')[0, 0]
    energy = graycoprops(glcm, 'energy')[0, 0]
    homogeneity = graycoprops(glcm, 'homogeneity')[0, 0]
    # Bu örnekte 3 temel özelliği geri döndürüyoruz
    return [contrast, energy, homogeneity]
```

Bu fonksiyon, LBP sonrası elde edilen desen görüntüsünden GLCM hesaplayarak dokusal bilgi kümemizi daha zengin hale getirir.

### 3.4 Gray Level Run Length Matrix (GLRLM)

**Teorik Temel:**

GLRLM, belirli bir yönde aynı gri seviyeye sahip ardışık piksel sayılarının (run) dağılımını ortaya koyar. Bir "run", aynı gri değere sahip bir dizi ardışık pikselden oluşur. GLRLM, her gri seviye değeri için run uzunluklarının frekansını kaydeder.

Resmi olarak, GLRLM R(i,r) matrisini şu şekilde tanımlayabiliriz:

- i: gri seviye değeri (0'dan L-1'e)

- r: run uzunluğu (1'den maksimum run uzunluğuna)

R(i,r)=seçilen yönde gri seviyesi i olan ve uzunluğu r piksel olan run sayısı

Bu matristen çeşitli istatistikler türetilebilir:

- **Short Run Emphasis (SRE):** Kısa run'ların vurgulanması.
- **Long Run Emphasis (LRE):** Uzun run'ların vurgulanması.
- **Gray Level Nonuniformity (GLN):** Gri seviyelerin nonuniform dağılımı.
- **Run Length Nonuniformity (RLN):** Run uzunluklarının dağılımı.
- **Run Percentage (RP)** vb.

Ancak bu çalışmada GLRLM tam kapsamıyla değil, basit bir run analiziyle sınırlı kalınmıştır. Örneğin sadece yatay yönde tarama yapılarak run uzunluklarının ortalaması, standart sapması veya entropisi gibi basit istatistikler çıkarılabilir.

**Kod Uygulaması (Basitleştirilmiş):**

Aşağıdaki örnekte tam bir GLRLM yerine, sadece yatay doğrultuda aynı gri değere sahip run'ların uzunlukları hesaplanıp bu uzunluklardan temel istatistikler çıkarılmaktadır.

```
from skimage.measure import shannon_entropy

def extract_run_length_features(image):
    # Bu basit yaklaşımda sadece yatay yönde run'lar hesaplanmaktadır.
    run_lengths = []
    for i in range(image.shape[0]):
        row = image[i, :]
        # Aynı gri seviye devam ettiği sürece run artar, değiştiğinde yeni run
başlar
        current_val = row[0]
        length = 1
        for j in range(1, len(row)):
            if row[j] == current_val:
                length += 1
            else:
                run_lengths.append(length)
                current_val = row[j]
                length = 1
        # Satırın sonunda kalan son run'ı da ekleyelim
        run_lengths.append(length)

    # Run length istatistikleri: ortalama, std ve entropi
    mean_rl = np.mean(run_lengths) if len(run_lengths) > 0 else 0
    std_rl = np.std(run_lengths) if len(run_lengths) > 0 else 0
    # Ek olarak tüm görüntünün entropisini de ekleyebiliriz (isteğe bağlı)
    image_entropy = shannon_entropy(image)
    return [mean_rl, std_rl, image_entropy]
```

Bu yaklaşım tam bir GLRLM analizi değildir, ancak fikir vermektedir. Tam GLRLM hesaplaması için skimage veya pyradiomics gibi kütüphaneler kullanılabilir. Daha gelişmiş GLRLM istatistikleri gelecekteki çalışmalar için bir genişleme alanıdır.

### 3.5 Tüm Özelliklerin Birleştirilmesi

Her yöntem farklı boyutta ve ölçekte özellik vektörleri üretir:

- GLCM: Örneğin 6 boyutlu bir özellik vektörü.
- LBP: LBP histogramı (örn. 24 nokta ve uniform ise yaklaşık 26 bin), normalize edilmiş histogram vektörü.
- LBGLCM: 3 boyutlu bir vektör (kontrast, enerji, homojenlik).
- GLRLM (basit): 3 boyutlu vektör (ortalama run, std, entropi).

Bu vektörler yatayda birleştirilerek tek bir uzun özellik vektörü oluşturulur. Ardından, bu özelliğin boyutu ve ölçekleri farklılık gösterebileceğinden standardizasyon gibi ön işlemler yapılabilir.

Kod:

```
def extract_features(image):
    glcm_f = extract_glcm_features(image)
    lbp_f = extract_lbp_features(image)
    lbglcm_f = extract_lbglcm_features(image)
    glrlm_f = extract_run_length_features(image)
    combined_features = np.hstack((glcm_f, lbp_f, glrlm_f, lbglcm_f))
    return combined_features
```

Bu fonksiyon, bir tek görüntü için GLCM, LBP, LBGLCM ve GLRLM özelliklerini hesaplayıp birleştirir ve sınıflandırıcılara verilecek nihai vektörü üretir.

Gerçekte, tüm veri seti için bu fonksiyon eğitim ve test görüntüleri üzerinde döngü kurularak çağrılır. Elde edilen büyük matris (num_samples x feature_dimension) makine öğrenimi modellerine girdi olarak kullanılır.

## 4. Deneysel Kurulum

**Veri Ayrımı:**

- Eğitim Verisi: 60.000 örnek
- Test Verisi: 10.000 örnek

Eğitim verisinden %20'lik bir kısım doğrulama (validation) seti olarak ayrılmıştır. Bu sayede, modellerin hiperparametre optimizasyonu ve performans karşılaştırması validation seti üzerinde yapılmıştır. En iyi performansı veren model, tüm eğitim verisiyle yeniden eğitilip test seti üzerinde değerlendirilmiştir.

**Özelliklerin Standardizasyonu:** Heterojen özelliklerin (GLCM istatistikleri, histogram değerleri, run-length istatistikleri) farklı ölçeklerde olması modelleri olumsuz

etkileyebileceğinden, **StandardScaler** kullanılarak her özellik ortalaması 0, standart sapması 1 olacak şekilde dönüştürülmüştür.

**Eğitim Süreci:** Her model **scikit-learn** kütüphanesi kullanılarak varsayılan veya basit ayarlarla eğitilmiştir. Örnek olarak SVM için **SVC(probability=True, random_state=42)** kullanılmış, Rastgele Orman için **RandomForestClassifier(random_state=42)** seçilmiştir.

**Değerlendirme Metrikleri:**

- **Doğruluk (Accuracy):** Doğru sınıflandırılan örneklerin oranı.
- **Karışıklık Matrisi (Confusion Matrix):** Hangi sınıfın hangi oranda başka bir sınıfa karıştırıldığını gösterir.
- **Precision, Recall, F1-Skoru:** Sınıf bazlı performansı detaylı inceler.
- **ROC Eğrisi ve AUC:** Sınıflandırma eşiği değiştikçe hassasiyet (TPR) ile FPR arasındaki ilişkiyi gösterir. AUC değeri, genel ayrıştırma gücünü ölçer.

## 5. Deney Sonuçları

**Validation Seti Üzerinde Modellerin Karşılaştırması:** Aşağıdaki tabloda her modelin doğrulama seti üzerindeki doğruluk değeri gösterilmektedir.S

| Model | Doğruluk (Validation) |
|---|---|
| Random Forest | ~0.6817 |
| SVM | ~0.7336 |
| KNN | ~0.6273 |
| Logistic Regression | ~0.6778 |
| Decision Tree | ~0.5391 |

En iyi performansı SVM sağlamıştır. Bu nedenle test aşamasında SVM modeli tercih edilmiştir.

**Test Seti Üzerinde SVM Performansı:** En iyi model olarak seçilen SVM, tüm eğitim seti (60.000 örnek) ile yeniden eğitildikten sonra test setinde %76 civarı doğruluk elde etmiştir. Sınıf bazlı metrikler aşağıdaki gibidir:

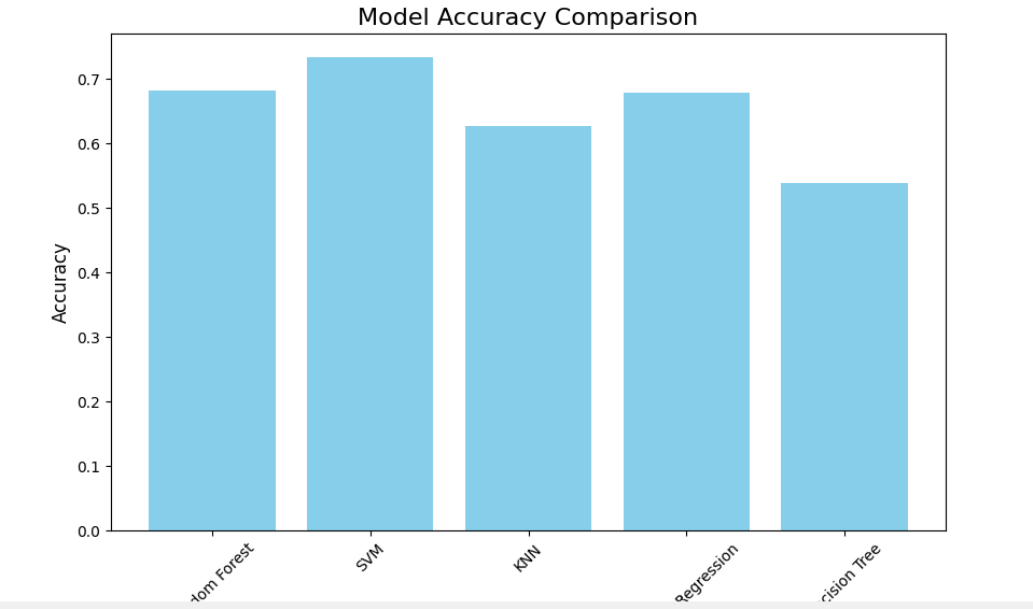
| Sınıf | Precision | Recall | F1-Skor | Destek (Örnek Sayısı) |
|---|---|---|---|---|
| 0 | 0.93 | 0.94 | 0.93 | 980 |
| 1 | 0.98 | 0.98 | 0.98 | 1135 |
| 2 | 0.56 | 0.50 | 0.53 | 1032 |
| 3 | 0.64 | 0.72 | 0.68 | 1010 |
| 4 | 0.81 | 0.87 | 0.84 | 982 |
| 5 | 0.61 | 0.60 | 0.60 | 892 |
| 6 | 0.68 | 0.63 | 0.65 | 958 |
| 7 | 0.79 | 0.79 | 0.79 | 1028 |
| 8 | 0.82 | 0.81 | 0.82 | 974 |
| 9 | 0.68 | 0.67 | 0.68 | 1009 |

Genel doğruluk: %76

Bu sonuçlar, bazı rakamların diğerlerine göre daha kolay tanındığını göstermektedir (örneğin “0” ve “1” oldukça başarılıyken “2” sınıfında başarı daha düşüktür). Bu, dokusal özellik setinin bazı rakamları ayırt etmekte daha başarılı olduğunu, bazılarında ise yetersiz kaldığını göstermektedir. Özellikle “2” ve “3” gibi benzer şekilli rakamlar arasında ayrıştırma zor olabilir. Bu durum, ileride daha spesifik dokusal özellikler veya farklı ön işleme adımlarıyla iyileştirilebilir.

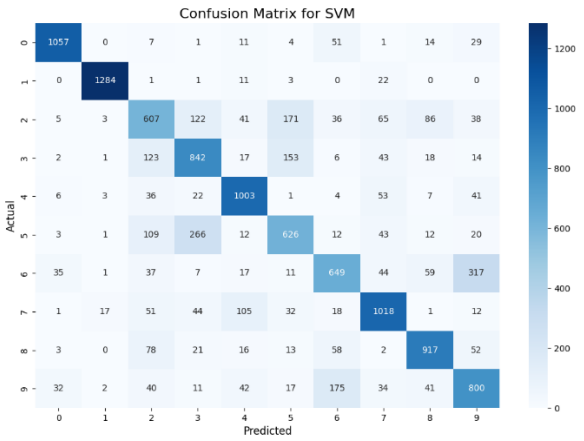
## 6. Görselleştirmeler ve Yorumlar

### Model Doğruluk Karşılaştırma Grafiği:




Bu çubuk grafikte farklı modellerin doğrulama setindeki doğrulukları gözlenebilir. SVM’nin diğer modellere nazaran daha yüksek olduğu açıkça görülmektedir. Rastgele Orman ve Lojistik Regresyon da orta seviyede performans sergilerken, Karar Ağacı en düşük performansta kalmıştır. Bu durum, dokusal özelliklerin doğrusal olmayan sınırlara sahip uzaylarda SVM gibi daha esnek modellerle daha iyi değerlendirildiğini düşündürmektedir.

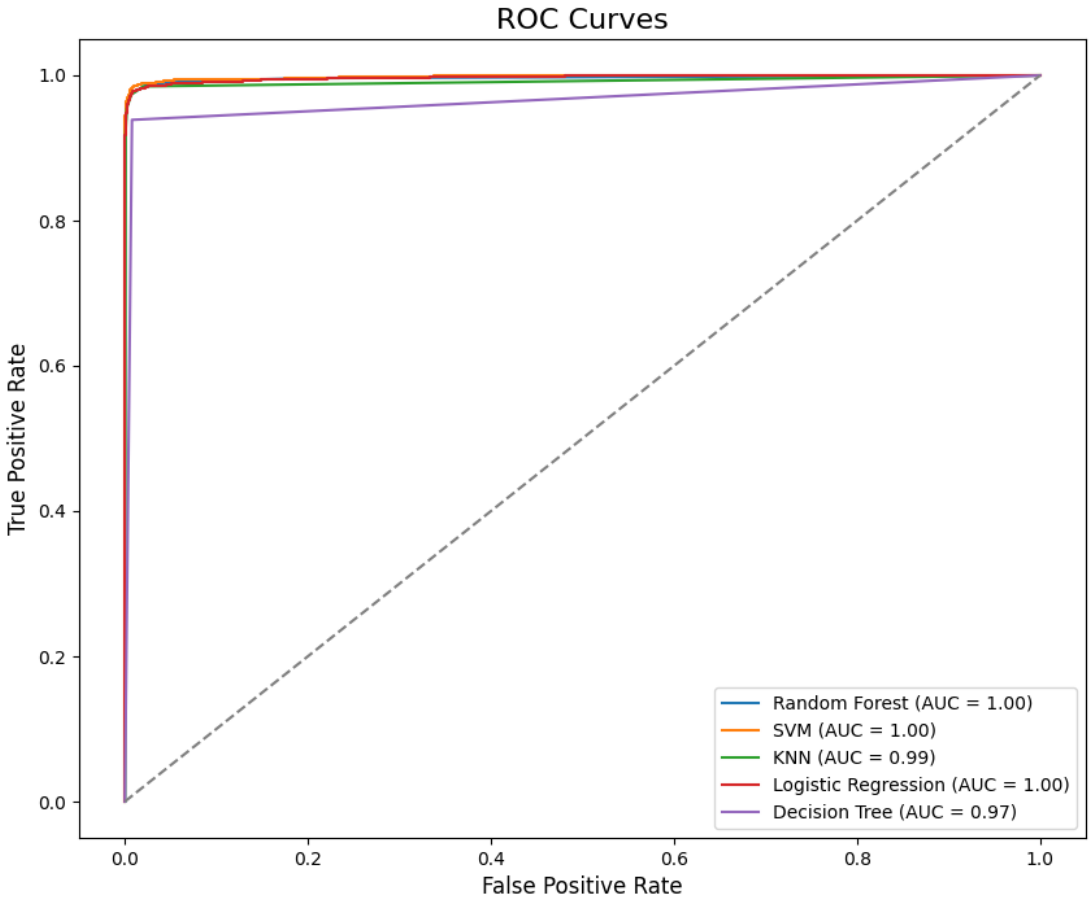
### SVM için Karışıklık Matrisi:



Karışıklık matrisi, gerçek etiketler ile modelin tahminlerini karşılaştırır. Diyagonal elemanlar doğru sınıflandırılan örneklerin sayısını ifade ederken, diğer hücreler hangi sınıfın hangi sınıfa karıştırıldığını gösterir. Bu matrise bakıldığında:

- "0" ve "1" sınıflarında yüksek doğru tanıma sayısı, bu rakamların dokusal özelliklerle kolay ayrıştırıldığını gösterir.
- "2" ve "3" sınıflarında ise görece daha fazla hata yapılmaktadır. Bu, bu rakamların dokusal açıdan daha fazla benzerlik taşıdığını veya kullanılan özelliklerin bu farkı yakalamakta yetersiz kaldığını gösterir.

**ROC Eğrileri ve AUC Değerleri:**




Farklı modellerin ROC eğrileri incelendiğinde SVM, Rastgele Orman ve Lojistik Regresyon gibi modellerin oldukça yüksek AUC değerlerine ulaştığı görülmektedir. ROC eğrisi, sınıf ayrım kabiliyetinin karar eşiği değiştikçe nasıl etkilendiğini gösterir. Eğriye yakın olan ve AUC değeri 1'e yakın olan modeller daha iyi ayrıştırma gücüne sahiptir. Bu, özellikle veri içindeki bazı sınıfların kolay ayırt edilebilir olduğunu ve modellerin genelde iyi bir ayrım yaptığı anlamına gelir. Ancak test doğruluğunda görülen %76'lık oran, bu modelin genelde iyi ayrım yaparken belli bazı sınıflarda sorun yaşadığını (bazı sınıflar için tutarlı ayrım zorluğu) göstermektedir.

## 7. Sonuçlar ve Tartışma

Bu çalışmada, el yazısı rakam tanıma problemi için dokusal tabanlı özellik çıkarma yöntemleri incelenmiş ve beş farklı sınıflandırıcı ile MNIST veri setinde test edilmiştir. Elde edilen sonuçlar aşağıdaki gibi özetlenebilir:

- Dokusal özellik çıkarma (GLCM, LBP, LBGLCM, GLRLM) ile ham piksel bazlı sınıflandırmaya kıyasla farklı bir bakış sağlanmıştır. Bu yöntemler, karakterlerin dokusal yapısını sayısal istatistiklerle ifade etmiştir.
- Test aşamasında en iyi performansı SVM modeli elde etmiştir (%76 doğruluk). Bu performans, saf piksel bilgisiyle eğitilen gelişmiş derin öğrenme modellerinin çok altında olsa da, amaç dokusal özellik tabanlı geleneksel makine öğrenimi yöntemlerini karşılaştırmak olduğundan elde edilen sonuçlar değerlidir.
- Bazı rakamlar (0 ve 1 gibi) dokusal özelliklerle daha kolay tanınırken, 2 ve 3 gibi benzer formlarda daha fazla hata yapılmıştır. Bu, gelecek çalışmalarda bu tip rakamlar için ek ön işleme, daha gelişmiş özellik setleri veya ek yöntemlerin denenmesini gerekli kılar.
- ROC eğrileri ve AUC değerleri, sınıflandırıcıların genellikle iyi ayrım gücüne sahip olduğunu, ancak doğruluk skorlarının bazı sınıflar için düşük kaldığını göstermiştir.

## 8. Kaynaklar


1. LeCun, Y., Cortes, C., & Burges, C. J. C. (2010). MNIST handwritten digit database.
2. Ojala, T., Pietikäinen, M., & Harwood, D. (1996). A Comparative Study of Texture Measures with Classification Based on Feature Distributions. Pattern Recognition.
3. Kaggle MNIST CSV: https://www.kaggle.com/datasets/oddrationale/mnist-in-csv
4. MNIST CSV DataSet: https://yann.lecun.com/exdb/mnist/
5. GLCM: https://medium.com/@girishajmera/feature-extraction-of-images-using-glcm-gray-level-cooccurrence-matrix-e4bda8729498


## 9. Ekler: Tam Kod

Aşağıda raporda bahsedilen tüm işlemleri (veri yükleme, özellik çıkarma, sınıflandırma, değerlendirme) içeren örnek kod yer almaktadır. Kod Python 3.12 ile test edilmiştir.

Kodu çalıştırmadan önce Dosya yolunu kendinize göre güncellemeyi **Unutmayınız**!

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import local_binary_pattern, graycomatrix, graycoprops
from skimage.measure import shannon_entropy
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix, roc_curve, auc
import seaborn as sns
from itertools import product

# Veri yolları
train_csv_path = r"C:\\Users\\FIRAT\\Desktop\\pattern-recognition\\Feature
Extraction of Images-analysis\\Data\\mnist_train.csv"
test_csv_path = r"C:\\Users\\FIRAT\\Desktop\\pattern-recognition\\Feature
Extraction of Images-analysis\\Data\\mnist_test.csv"

# Veri yükleme
def load_data(csv_path):
    df = pd.read_csv(csv_path)
    labels = df.iloc[:, 0].values
    images = df.iloc[:, 1:].values.reshape(-1, 28, 28).astype(np.uint8)
    return images, labels

train_images, train_labels = load_data(train_csv_path)
test_images, test_labels = load_data(test_csv_path)

# Özellik çıkarma fonksiyonları

def extract_glcm_features(image, distances=[1], angles=[0]):
    glcm = graycomatrix(image, distances=distances, angles=angles, levels=256,
symmetric=True, normed=True)
    features = []
    properties = ['contrast', 'dissimilarity', 'homogeneity', 'energy',
'correlation', 'ASM']
    for prop in properties:
        features.append(graycoprops(glcm, prop)[0, 0])
    return features

def extract_lbp_features(image, radius=3, n_points=24):
    lbp = local_binary_pattern(image, n_points, radius, method='uniform')
    (hist, _) = np.histogram(lbp.ravel(),
                             bins=np.arange(0, n_points + 3),
                             range=(0, n_points + 2))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-6)
    return hist

def extract_run_length_features(image):
    # Simplified GLRLM extraction
```

```
    run_lengths = []
    for i in range(image.shape[0]):
        row = image[i, :]
        run_length = np.diff(np.where(np.diff(row) != 0)[0])
        run_lengths.extend(run_length)
    return [np.mean(run_lengths), np.std(run_lengths), shannon_entropy(image)]

def extract_lbglcm_features(image):
    lbp = local_binary_pattern(image, 24, 3, method='uniform')
    glcm = graycomatrix(lbp.astype(np.uint8), distances=[1], angles=[0],
levels=256, symmetric=True, normed=True)
    contrast = graycoprops(glcm, 'contrast')[0, 0]
    energy = graycoprops(glcm, 'energy')[0, 0]
    homogeneity = graycoprops(glcm, 'homogeneity')[0, 0]
    return [contrast, energy, homogeneity]

def extract_features(images):
    features = []
    for img in images:
        glcm_f = extract_glcm_features(img)
        lbp_f = extract_lbp_features(img)
        glrlm_f = extract_run_length_features(img)
        lbglcm_f = extract_lbglcm_features(img)
        combined_features = np.hstack((glcm_f, lbp_f, glrlm_f, lbglcm_f))
        features.append(combined_features)
    return np.array(features)

# Özelliklerin çıkarılması
print("Extracting features for training data...")
train_features = extract_features(train_images)
print("Extracting features for test data...")
test_features = extract_features(test_images)

# Özelliklerin standardize edilmesi
scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)
test_features = scaler.transform(test_features)

# Veri bölünmesi
X_train, X_val, y_train, y_val = train_test_split(train_features,
train_labels, test_size=0.2, random_state=42)

# Modellerin tanımlanması
models = {
    "Random Forest": RandomForestClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "KNN": KNeighborsClassifier(),
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42)
```

```
}

# Modellerin eğitilmesi ve değerlendirilmesi
results = {}
confusion_matrices = {}
roc_curves = {}

for name, model in models.items():
    print(f"Training {name}...")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    y_proba = model.predict_proba(X_val) if hasattr(model, "predict_proba")
else None
    accuracy = accuracy_score(y_val, y_pred)
    results[name] = accuracy
    confusion_matrices[name] = confusion_matrix(y_val, y_pred)
    if y_proba is not None:
        fpr, tpr, _ = roc_curve(y_val, y_proba[:, 1], pos_label=1)
        roc_curves[name] = (fpr, tpr)
    print(f"{name} Accuracy: {accuracy}")

# En iyi modeli seçme
best_model_name = max(results, key=results.get)
best_model = models[best_model_name]
print(f"Best Model: {best_model_name}")

# Test setinde en iyi modelin performansı
best_model.fit(train_features, train_labels)
test_predictions = best_model.predict(test_features)
print("Test Data Classification Report:")
print(classification_report(test_labels, test_predictions))

# Sonuçları görselleştirme
# Model başarılarının bar grafiği
plt.figure(figsize=(10, 6))
plt.bar(results.keys(), results.values(), color='skyblue')
plt.title('Model Accuracy Comparison', fontsize=16)
plt.xlabel('Models', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.show()

# Confusion Matrix görselleştirme
plt.figure(figsize=(12, 8))
sns.heatmap(confusion_matrices[best_model_name], annot=True, fmt='d',
cmap='Blues', xticklabels=np.unique(test_labels),
yticklabels=np.unique(test_labels))
plt.title(f'Confusion Matrix for {best_model_name}', fontsize=16)
plt.xlabel('Predicted', fontsize=12)
```

```
plt.ylabel('Actual', fontsize=12)
plt.show()

# ROC AUC Curve
plt.figure(figsize=(10, 8))
for name, (fpr, tpr) in roc_curves.items():
    plt.plot(fpr, tpr, label=f"{name} (AUC = {auc(fpr, tpr):.2f})")
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title('ROC Curves', fontsize=16)
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.legend()
plt.show()
```