Ürün Kategorizasyonu için Makine Öğrenmesi Modelleri: Adım Adım Kılavuz

Giriş

Merhaba! Bu rapor, bir market veri setindeki ürün isimlerinden yola çıkarak bu ürünlerin hangi kategoriye ait olduğunu tahmin edebilen bilgisayar programları (makine öğrenmesi modelleri) oluşturma sürecini adım adım anlatmak için hazırlanmıştır. Amacımız, bu teknik süreci ve kullanılan terimleri, daha önce hiç makine öğrenmesi veya programlama ile ilgilenmemiş bir kişinin bile anlayabileceği sadelikte açıklamaktır.

Elimizde "market_data.csv" adında bir dosya bulunuyor. Bu dosya, çeşitli market ürünlerinin isimlerini ve bu ürünlerin ait olduğu kategorileri içeriyor. Görevimiz, bilgisayara bu veriyi öğreterek, yeni bir ürün ismi gördüğünde onun hangi kategoriye ait olduğunu otomatik olarak tahmin etmesini sağlamaktır. Bu işlem, örneğin bir e-ticaret sitesinde ürünleri doğru kategorilere yerleştirmek veya stok yönetimini kolaylaştırmak gibi birçok pratik fayda sağlayabilir.

Bu rapor boyunca şu adımları takip edeceğiz:

1. **Veriyi Keşfetme:** İlk olarak, elimizdeki veriyi daha yakından tanıyacağız. İçinde ne tür bilgiler var, kaç tane ürün ve kategori bulunuyor gibi soruların cevaplarını arayacağız.
2. **Veriyi Ön İşleme:** Bilgisayarların anlayabilmesi için veriyi özel bir formata dönüştürmemiz gerekiyor. Bu adımda metinleri temizleyecek ve sayısal değerlere çevireceğiz.
3. **Modelleme:** Farklı makine öğrenmesi algoritmaları kullanarak tahmin modelleri oluşturacağız. Bu algoritmaların nasıl çalıştığını ve neden seçildiklerini basitçe açıklayacağız.
4. **Değerlendirme:** Oluşturduğumuz modellerin ne kadar başarılı olduğunu ölçeceğiz. Accuracy, Precision, Recall gibi terimlerin ne anlama geldiğini ve sonuçları nasıl yorumlayacağımızı öğreneceğiz.
5. **Karşılaştırma ve Seçim:** Modelleri karşılaştırarak görevimiz için en iyi performansı gösteren modeli seçecek ve nedenini açıklayacağız.

Hazırsanız, bu öğretici yolculuğa başlayalım!

Adım 1: Veriyi Keşfetme - Elimizdeki Hazineyi Tanıyalım

Bir projeye başlarken ilk adım, elimizdeki malzemeyi, yani veriyi tanımaktır. Tıpkı bir aşçının yemek yapmadan önce malzemelerini kontrol etmesi gibi, biz de makine öğrenmesi modelimizi eğitmeden önce verimizi incelemeliyiz.

Veri keşfi için basit bir Python betiği kullandık (`explore_data.py`). Bu betik, popüler bir veri analiz kütüphanesi olan `pandas`'ı kullanarak `market_data.csv` dosyasını okudu ve bize bazı temel bilgiler verdi. İşte bulgularımız:

Veri Setinin İlk 5 Satırı:

```
                         item_name    category_name
0  sprite 1 lt limon aromali gazoz     gazli içecek
1                         toz seker        çay kahve
2             falim sakız 5li nane  sakız sekerleme
3            falim sakız 5li çilek  sakız sekerleme
4          f neffis toz seker 2 kg        çay kahve
```

- **Ne Görüyoruz?** Veri setimiz iki sütundan oluşuyor: `item_name` (ürün ismi) ve `category_name` (ürün kategorisi). Her satır bir ürünü temsil ediyor. Örneğin, ilk satırda "sprite 1 lt limon aromali gazoz" ürününün "gazli içecek" kategorisine ait olduğunu görüyoruz. (`df.head()` fonksiyonu bize bu ilk birkaç satırı gösterir.)

Veri Seti Bilgileri:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9019 entries, 0 to 9018
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   item_name      9019 non-null   object
 1   category_name  9019 non-null   object
dtypes: object(2)
memory usage: 141.0+ KB
```

- **Ne Anlama Geliyor?**
  - Toplam **9019 adet ürün (satır)** bilgisi var.
  - İki sütunumuz (`item_name` ve `category_name`) da `object` tipinde. Bu genellikle metin verileri için kullanılır.
  - `Non-Null Count` her iki sütun için de 9019. Bu harika bir haber! **Hiç eksik veri yok** anlamına geliyor. Yani her ürünün hem ismi hem de kategorisi belirtilmiş. (`df.info()` fonksiyonu bize bu genel yapıyı ve veri tiplerini özetler.)

İstatistiksel Özet:

```
                                item_name          category_name
count                                9019                   9019
unique                               9018                     63
top     wee silikon desenli emzik damakli  mutfak eşya gereçleri
freq                                    2                    668
```

- **Ne Öğrendik?**
  - `count`: Yine 9019 ürün olduğunu teyit ediyor.
  - `unique`: **9018 farklı ürün ismi** ve **63 farklı kategori** olduğunu gösteriyor. Neredeyse her ürün isminin benzersiz olması ilginç.
  - `top`: En sık görünen ürün isminin "wee silikon desenli emzik damakli"

ve en sık görünen kategorinin "mutfak eşya gereçleri" olduğunu belirtiyor.
  - `freq`: En sık görünen ürün isminin 2 kez, en sık görünen kategorinin ise 668 kez geçtiğini gösteriyor. (`df.describe(include='object')` fonksiyonu metin sütunları için bu tür özetleri verir.)

Eksik Değer Kontrolü:

```
item_name        0
category_name    0
dtype: int64
```

- **Sonuç:** Bu çıktı, `df.info()`'da gördüğümüzü teyit ediyor: Her iki sütunda da **0 eksik değer** var. (`df.isnull().sum()` her sütundaki eksik değer sayısını verir.)

Keşif Özeti: Elimizde yaklaşık 9000 ürünlük, temiz (eksik verisi olmayan) bir veri seti var. Ürün isimleri büyük ölçüde benzersiz ve bu ürünler 63 farklı kategoriye ayrılmış. Bu bilgilerle artık veriyi modelleme için hazırlama adımına geçebiliriz.

Teknik Not: Veri keşfi sırasında `pandas` kütüphanesinin kurulu olmadığını fark ettik ve `python3.11 -m pip install pandas` komutuyla yükledik. Ayrıca, başlangıçta dosya yolunu doğru belirtmediğimiz için bir hata aldık ve kodu `data_file = '/home/ubuntu/upload/market_data.csv'` şeklinde güncelleyerek bu sorunu çözdük. Bu tür küçük teknik sorunlar veri analizi süreçlerinde sıkça karşılaşılabilir.

Adım 2: Veriyi Ön İşleme - Bilgisayarın Anlayacağı Dile Çevirme

Bilgisayarlar, bizim gibi metinleri doğrudan anlayamazlar; sayılarla çalışmayı tercih ederler. Bu nedenle, `item_name` ve `category_name` sütunlarındaki metin verilerini makine öğrenmesi modellerimizin işleyebileceği sayısal bir formata dönüştürmemiz gerekiyor. Bu sürece **veri ön işleme** diyoruz.

Kullandığımız `data_preprocessing.py` betiği şu adımları gerçekleştirdi:

1. **Metin Temizleme:** Ürün isimlerindeki (`item_name`) gereksiz karakterleri temizledik.
   - **Neden?** "Sprite 1 LT" ile "sprite 1 lt" aynı üründür, ancak bilgisayar için farklıdır. Büyük/küçük harf farkları, noktalama işaretleri ve sayılar modelin kafasını karıştırabilir.
   - **Nasıl?**
     - Tüm harfleri küçük harfe çevirdik (`text.lower()`).
     - Türkçe harfler (ç, ğ, ı, ö, ş, ü) ve boşluklar dışındaki her şeyi (sayılar, noktalama işaretleri vb.) boşlukla değiştirdik (`re.sub(r'[^a-zçğıöşü\s]', ' ', text)`). `re` kütüphanesi metin içinde desen arama ve değiştirme için kullanılır.

  - Fazla boşlukları tek boşluğa indirdik (`re.sub(r'\s+', ' ', text).strip()`).
- **Örnek:** "Sprite 1 Lt Limon Aromalı Gazoz" -> "sprite lt limon aromali gazoz"

2. **Özellik Çıkarımı (TF-IDF):** Temizlenmiş ürün isimlerini sayısal vektörlere dönüştürdük.
   - **Neden?** Modellerin metinler arasındaki benzerlikleri ve farkları anlaması için sayısal bir temsile ihtiyaçları vardır.
   - **Nasıl (TF-IDF)?** TF-IDF (Term Frequency-Inverse Document Frequency), bir kelimenin bir üründeki önemini hesaplayan bir yöntemdir. Bir kelime bir üründe sık geçiyorsa (TF yüksek) ama tüm ürünlerde çok yaygın değilse (IDF yüksek), o kelime o ürün için daha ayırt edici kabul edilir.
     - `TfidfVectorizer` (Scikit-learn kütüphanesinden) bu işlemi yapar.
     - `max_features=5000`: Sadece en önemli (en sık geçen) 5000 kelimeyi/kelime grubunu dikkate almasını söyledik. Bu, işlem yükünü azaltır.
     - `ngram_range=(1, 2)`: Modelin sadece tekli kelimeleri değil ("limon", "gazoz"), aynı zamanda ardışık ikili kelime gruplarını da ("limon aromali") dikkate almasını sağladık. Bu, anlamı daha iyi yakalamaya yardımcı olabilir.
   - **Sonuç:** Her bir ürün ismi, 5000 sayıdan oluşan bir vektöre dönüştürüldü. Tüm ürünler için bu vektörler bir matris (tablo) oluşturdu: `(9019, 5000)` boyutunda bir matris elde ettik (9019 ürün, 5000 özellik/sayı).
3. **Hedef Değişkeni Kodlama (Label Encoding):** Kategori isimlerini (`category_name`) sayılara dönüştürdük.
   - **Neden?** Modelin tahmin etmesi gereken şey (hedef değişken) de sayısal olmalıdır.
   - **Nasıl?** `LabelEncoder` (Scikit-learn'den) her bir benzersiz kategori ismine 0'dan başlayarak bir numara atar.
   - **Örnek:** "gazli içecek" -> 14, "çay kahve" -> 59, "sakız sekerleme" -> 38 gibi.
   - **Sonuç:** 63 farklı kategori olduğu için 0'dan 62'ye kadar sayılar elde ettik.
4. **Nadir Kategorileri Filtreleme:** Çok az örneği olan (sadece 1 kez geçen) kategorileri veri setinden çıkardık.
   - **Neden?** Bir kategoriden sadece bir örnek varsa, modelin o kategoriyi öğrenmesi çok zordur ve genellikle modelin performansını düşürebilir.
   - **Nasıl?** `Counter` kullanarak her kategorinin kaç kez geçtiğini saydık ve en az 2 örneği olanları tuttuk.
   - **Sonuç:** Sadece 1 örneği olan 2 kategori bulundu ve bu 2 ürün veri setinden çıkarıldı. Veri setimiz 9017 ürüne düştü.
5. **Veriyi Eğitim ve Test Setlerine Ayırma:** Veriyi iki parçaya böldük.
   - **Neden?** Modeli verinin bir kısmıyla eğitip (%80), daha önce hiç

görmediği diğer kısmıyla (%20) test ederek modelin gerçek dünya performansını ölçebiliriz. Eğer tüm veriyle eğitirsek, model veriyi ezberleyebilir ve yeni ürünlerde başarısız olabilir.
- **Nasıl?** `train_test_split` (Scikit-learn'den) fonksiyonunu kullandık.
  - `test_size=0.2`: Verinin %20'sini test için ayır.
  - `random_state=42`: Bölme işlemini rastgele yapar, ancak bu sayıyı sabit tutarak her çalıştırdığımızda aynı bölme işleminin yapılmasını sağlarız (tekrarlanabilirlik için).
  - `stratify=y_filtered`: Bölme işlemi yapılırken, her kategorinin eğitim ve test setlerindeki oranının orijinal veri setindekiyle aynı olmasına özen gösterir. Bu, özellikle bazı kategoriler az sayıda örneğe sahipse önemlidir.
- **Sonuç:** 7213 ürün eğitim için, 1804 ürün test için ayrıldı.

6. **İşlenmiş Veri ve Nesneleri Kaydetme:** Ön işleme adımlarında oluşturulan eğitim/test verilerini, TF-IDF dönüştürücüsünü (`tfidf_vectorizer`) ve Kategori Kodlayıcısını (`label_encoder`) `joblib` kütüphanesi kullanarak diske kaydettik.
   - **Neden?** Bu sayede, modelleri eğitirken veya yeni tahminler yaparken ön işleme adımlarını tekrar tekrar çalıştırmak zorunda kalmayız. Kaydedilmiş nesneleri yükleyerek doğrudan kullanabiliriz.

Ön İşleme Özeti: Ham metin verilerimizi temizledik, ürün isimlerini TF-IDF ile sayısal vektörlere, kategori isimlerini ise basit sayılara dönüştürdük. Çok nadir kategorileri filtreledik ve modelin performansını doğru ölçebilmek için veriyi eğitim ve test setlerine ayırdık. Artık verimiz, makine öğrenmesi modellerini eğitmeye hazır!

Teknik Not: Bu adım için `scikit-learn` ve `joblib` kütüphanelerine ihtiyaç duyduk ve bunları `python3.11 -m pip install scikit-learn joblib` komutuyla yükledik.

Adım 3: Modelleme - Tahmin Yapacak Programları Eğitme

Verimizi hazırladığımıza göre, şimdi sıra geldi tahmin yapacak makine öğrenmesi modellerini oluşturmaya ve eğitmeye. Bu projede üç farklı popüler sınıflandırma algoritmasını denedik:

1. **Lojistik Regresyon (Logistic Regression):**
   - **Nasıl Çalışır?** İsmi regresyon olsa da, aslında bir sınıflandırma algoritmasıdır. Temelde, farklı kategorileri birbirinden ayıran bir çizgi (veya çok boyutlu uzayda bir düzlem/hiperdüzlem) bulmaya çalışır. Bir ürünün özelliklerine (TF-IDF vektörü) bakarak, o ürünün bu çizginin hangi tarafında kaldığına göre kategori tahmini yapar.
   - **Neden Seçildi?** Genellikle iyi bir başlangıç noktasıdır, yorumlanması diğerlerine göre daha kolay olabilir ve özellikle metin sınıflandırma gibi seyrek (çoğu değeri sıfır olan) verilerde iyi

performans gösterebilir.

- **Kodda Nasıl?** `sklearn.linear_model.LogisticRegression` sınıfını kullandık.
  - `solver='saga'`: Büyük veri setleri ve çok sayıda kategori için uygun bir çözücü algoritma.
  - `class_weight='balanced'`: Veri setimizde bazı kategoriler diğerlerinden daha fazla örneğe sahip olabilir (dengesizlik). Bu ayar, azınlıkta kalan kategorilere daha fazla önem vererek modelin daha adil öğrenmesini sağlar.
  - `max_iter=5000`: Modelin en iyi çizgiyi bulmak için yapacağı maksimum deneme sayısı. Bazen modelin optimum sonuca ulaşması için bu sayıyı artırmak gerekebilir.
  - `random_state=42`: Tekrarlanabilirlik için.
- **Eğitim:** `model.fit(X_train, y_train)` komutuyla model, eğitim verisindeki ürün isimleri (X_train) ile kategorileri (y_train) arasındaki ilişkiyi öğrendi.

2. **Naive Bayes (MultinomialNB):**
   - **Nasıl Çalışır?** Olasılık temelli bir yaklaşımdır. Bir ürün ismindeki kelimelere bakarak, o ürünün her bir kategoriye ait olma olasılığını hesaplar (Bayes Teoremi'ni kullanarak). En yüksek olasılığa sahip kategoriyi tahmin olarak verir. "Naive" (saf) denmesinin sebebi, kelimelerin birbirlerinden bağımsız olduğunu varsaymasıdır (örneğin, "limon" kelimesinin geçmesinin "gazoz" kelimesinin geçme olasılığını etkilemediğini varsayar, ki bu gerçekte tam doğru olmasa da pratikte iyi çalışır).
   - **Neden Seçildi?** Metin sınıflandırma görevlerinde oldukça hızlı ve etkili bir temel modeldir. Özellikle TF-IDF gibi kelime frekanslarına dayalı özelliklerle iyi çalışır ve hesaplama açısından verimlidir.
   - **Kodda Nasıl?** `sklearn.naive_bayes.MultinomialNB` sınıfını kullandık. Metin verileri için genellikle bu versiyonu tercih edilir.
     - `alpha=1.0`: Laplace düzeltmesi. Eğitim verisinde hiç geçmeyen bir kelime test verisinde karşımıza çıkarsa, olasılık sıfır olur ve bu tüm hesaplamayı bozabilir. Bu parametre, tüm olasılıklara küçük bir değer ekleyerek sıfır olasılıkları önler.
   - **Eğitim:** Yine `model.fit(X_train, y_train)` ile model, kelimelerin hangi kategorilerde daha sık geçtiğini olasılıksal olarak öğrendi.
3. **Karar Ağacı (Decision Tree):**
   - **Nasıl Çalışır?** Adından da anlaşılacağı gibi, bir dizi karar (soru) sorarak sonuca ulaşan ağaç benzeri bir yapı oluşturur. Ürün ismindeki belirli kelimelerin (veya TF-IDF değerlerinin) varlığına/yokluğuna veya değerine göre dallanarak ilerler ve sonunda bir kategori tahminine (yaprak düğüm) ulaşır. Örneğin, " 'limon' kelimesi geçiyor mu? Evet -> 'gazoz' kelimesi geçiyor mu? Evet -> Kategori: gazli içecek" gibi bir yol izleyebilir.
   - **Neden Seçildi?** Karar verme süreci görselleştirilebilir ve anlaşılması

kolaydır. Verideki doğrusal olmayan ilişkileri yakalayabilir.
- **Kodda Nasıl?** `sklearn.tree.DecisionTreeClassifier` sınıfını kullandık.
  - `class_weight='balanced'`: Lojistik Regresyon'daki gibi, dengesiz sınıflara karşı modeli daha adil hale getirir.
  - `random_state=42`: Tekrarlanabilirlik için.
- **Eğitim:** `model.fit(X_train, y_train)` ile model, veriyi en iyi şekilde ayıran soruları (kuralları) içeren ağaç yapısını oluşturdu.

Modelleme Özeti: Üç farklı algoritma kullanarak modellerimizi eğittik. Her model, eğitim verisindeki ürün isimleri ve kategoriler arasındaki deseni kendi yöntemleriyle öğrendi. Şimdi sıra, bu modellerin daha önce hiç görmedikleri test verisi üzerinde ne kadar başarılı olduklarını ölçmeye geldi.

Adım 4: Değerlendirme - Modellerimizin Karnesini İnceleme

Modellerimizi eğittik, peki ne kadar başarılılar? Bunu anlamak için modellerin test verisi (`X_test`) üzerindeki tahminlerini (`y_pred`), gerçek kategori bilgileriyle (`y_test`) karşılaştırdık. Bu karşılaştırmayı yapmak için bazı standart ölçüm metrikleri kullandık.

Temel Değerlendirme Metrikleri:

- **Accuracy (Doğruluk):** En basit metriktir. Modelin toplam tahminlerinin ne kadarının doğru olduğunu gösterir. (Doğru Tahmin Sayısı / Toplam Tahmin Sayısı). Örneğin, %90 doğruluk, modelin test verisindeki 100 üründen 90'ının kategorisini doğru tahmin ettiği anlamına gelir.
- **Precision (Kesinlik):** Modelin belirli bir kategori (örneğin, "süt") olarak tahmin ettiği ürünlerden kaçının gerçekten o kategoriye ait olduğunu ölçer. (Doğru Pozitif / (Doğru Pozitif + Yanlış Pozitif)). Yüksek kesinlik, modelin o kategoriye ait olmayan ürünleri yanlışlıkla o kategoriye atama (yanlış alarm) oranının düşük olduğunu gösterir.
- **Recall (Duyarlılık/Geri Çağırma):** Gerçekte belirli bir kategoriye (örneğin, "süt") ait olan ürünlerden kaçının model tarafından doğru bir şekilde o kategori olarak bulunduğunu ölçer. (Doğru Pozitif / (Doğru Pozitif + Yanlış Negatif)). Yüksek duyarlılık, modelin o kategoriye ait ürünleri gözden kaçırma (ıskalama) oranının düşük olduğunu gösterir.
- **F1-Skoru:** Kesinlik ve Duyarlılığın harmonik ortalamasıdır (`2 * (Precision * Recall) / (Precision + Recall)`). Bu iki metrik arasında bir denge kurar. Özellikle birini artırırken diğerinin çok düşmesini istemediğimiz durumlarda veya sınıf dağılımları dengesiz olduğunda kullanışlıdır.

Sınıflandırma Raporu (`classification_report`):

Scikit-learn kütüphanesinin bu fonksiyonu, her bir kategori için Precision, Recall ve F1-Skorunu hesaplar. Ayrıca şu özet metrikleri de sunar:

- `support`: Her kategorinin test setindeki gerçek örnek sayısı.
- `macro avg`: Metriklerin (Precision, Recall, F1) her kategori için hesaplanan değerlerinin basit aritmetik ortalaması. Her kategoriye eşit ağırlık verir.
- `weighted avg`: Metriklerin her kategori için hesaplanan değerlerinin, o kategorinin örnek sayısına (`support`) göre ağırlıklandırılmış ortalaması. Kalabalık kategorilerin performansa daha fazla etki etmesini sağlar.

Modellerin Sonuçları:

Her model için çalıştırılan Python betikleri (`logistic_regression_model.py`, `naive_bayes_model.py`, `decision_tree_model.py`) bu metrikleri hesapladı ve çıktı dosyalarına yazdı (`logistic_regression_output.txt`, `naive_bayes_output.txt`, `decision_tree_output.txt`). İşte özet sonuçlar:

1. Lojistik Regresyon:

- Accuracy: **0.9318** (%93.18)
- Weighted Avg Precision: 0.94
- Weighted Avg Recall: 0.93
- Weighted Avg F1-score: 0.93
- *Not:* Eğitim sırasında `ConvergenceWarning` alındı. Bu, modelin belirlenen `max_iter` (5000) deneme sayısında tam olarak en iyi çözüme ulaşamamış olabileceğini gösterir. Ancak elde edilen %93'lük başarı oranı yine de oldukça yüksektir.

2. Naive Bayes:

- Accuracy: **0.8525** (%85.25)
- Weighted Avg Precision: 0.86
- Weighted Avg Recall: 0.85
- Weighted Avg F1-score: 0.83

3. Karar Ağacı:

- Accuracy: **0.8725** (%87.25)
- Weighted Avg Precision: 0.89
- Weighted Avg Recall: 0.87
- Weighted Avg F1-score: 0.88

Değerlendirme Özeti: Üç model de test verisi üzerinde tahminler yaptı ve performanslarını ölçtük. Lojistik Regresyon, %93'ün üzerinde bir doğrulukla en başarılı görünen model oldu. Naive Bayes ve Karar Ağacı modelleri de sırasıyla %85 ve %87 doğruluk oranlarıyla fena olmayan sonuçlar verdiler. Şimdi bu sonuçları daha detaylı karşılaştırarak en iyi modeli seçeceğiz.

Adım 5: Karşılaştırma ve En İyi Modeli Seçme - Şampiyonu Belirliyoruz

Artık elimizde üç modelin de karnesi var. Peki, hangisi bu görev için en iyisi? Karar vermek için metrikleri yan yana koyalım:

Model	Accuracy	Weighted Avg Precision	Weighted Avg Recall	Weighted Avg F1-score
Lojistik Regresyon	**0.9318**	**0.94**	**0.93**	**0.93**
Naive Bayes	0.8525	0.86	0.85	0.83
Karar Ağacı	0.8725	0.89	0.87	0.88

Gözlemlerimiz:

- **Lojistik Regresyon açık ara önde:** Hem genel doğruluk (Accuracy) hem de sınıfların dengesizliğini dikkate alan ağırlıklı ortalama (Weighted Avg) metriklerde Lojistik Regresyon, diğer iki modelden belirgin şekilde daha iyi performans göstermiştir.
- **Karar Ağacı ikinci sırada:** Naive Bayes'e göre biraz daha iyi sonuçlar vermiştir.
- **Naive Bayes en düşük performansı sergiledi:** Ancak yine de %85'in üzerinde bir doğrulukla kabul edilebilir bir sonuç elde etmiştir.

Sonuç:

Elde edilen metriklere dayanarak, **Lojistik Regresyon modeli** bu ürün kategorizasyon görevi için test ettiğimiz modeller arasında **en başarılı olanıdır**. %93'ün üzerindeki doğruluk oranı, ürün isimlerinden kategorileri yüksek bir başarıyla tahmin edebildiğini göstermektedir.

Sonuç ve Sonraki Adımlar

Bu rapor boyunca, bir market veri setindeki ürün isimlerinden kategorilerini tahmin etmek için makine öğrenmesi modelleri oluşturma sürecini adım adım inceledik. Veriyi keşfettik, bilgisayarın anlayacağı formata getirmek için ön işlemlerden geçirdik, üç farklı model (Lojistik Regresyon, Naive Bayes, Karar Ağacı) eğittik, bu modellerin performanslarını çeşitli metriklerle değerlendirdik ve son olarak en iyi performansı gösteren Lojistik Regresyon modelini seçtik.

Bu süreç, makine öğrenmesinin temel adımlarını ve metin verileriyle nasıl çalışılabileceğini göstermektedir. Elde ettiğimiz Lojistik Regresyon modeli, %93 doğrulukla oldukça başarılı bir sonuç vermiştir ve pratik uygulamalarda kullanılabilir.

Olası İyileştirmeler ve Sonraki Adımlar:

- **Hiperparametre Optimizasyonu:** Modellerin performansını daha da artırmak için `max_iter`, `C` (Lojistik Regresyon için), `alpha` (Naive Bayes için), `max_depth` (Karar Ağacı için) gibi ayarları (hiperparametreler) sistematik olarak deneyerek en iyi kombinasyonu bulabiliriz.
- **Daha Gelişmiş Modeller:** Random Forest, Gradient Boosting (XGBoost, LightGBM) veya Derin Öğrenme (LSTM, Transformer) gibi daha karmaşık modelleri deneyerek potansiyel olarak daha yüksek doğruluk oranları elde edebiliriz.
- **Özellik Mühendisliği:** TF-IDF yerine kelime gömme (Word Embeddings) gibi daha gelişmiş metin temsil yöntemleri kullanabiliriz.
- **Veri Artırma:** Özellikle az örneği olan kategoriler için veri setini sentetik olarak artırma teknikleri denenebilir.

Umarım bu rapor, makine öğrenmesi dünyasına bir giriş yapmak ve bu teknolojinin nasıl çalıştığını anlamak açısından faydalı olmuştur. Eklerde, süreç boyunca kullanılan kodları ve elde edilen çıktıları bulabilirsiniz.