1. **Standardizasyon ve Normalizasyon: Veri Ön İşlemede İki Temel Kavram**

**Standardizasyon** ve **normalizasyon**, makine öğrenmesi ve istatistik gibi alanlarda sıklıkla kullanılan, verileri analiz etmeye hazır hale getirmek için uygulanan iki önemli veri ön işleme tekniğidir. Her ne kadar benzer amaçları olsa da, bu iki yöntem arasında önemli farklılıklar bulunmaktadır.

**Standardizasyon Nedir?**

Standardizasyon, bir veri setindeki tüm değerlerin ortalaması 0 ve standart sapması 1 olacak şekilde yeniden ölçeklendirilmesidir. Bu işlem, farklı birimlerde veya ölçeklerde olan verileri karşılaştırılabilir hale getirmek için kullanılır. Standardizasyon, özellikle normal dağılıma sahip veriler üzerinde etkilidir.

* **Amaç:** Verileri aynı ölçeğe getirerek, farklı değişkenlerin model üzerindeki etkisini eşitlemek.
* **Formül:**
* z = (x - μ) / σ
  + z: Standardize edilmiş değer
  + x: Orjinal değer
  + μ: Ortalama
  + σ: Standart sapma

**Normalizasyon Nedir?**

Normalizasyon ise, verilerin belirli bir aralıkta (genellikle 0 ile 1 arasında) sıkıştırılması işlemidir. Bu yöntem, farklı aralıklarda olan verilerin etkisini dengelemek ve bazı algoritmaların daha iyi çalışmasını sağlamak için kullanılır.

* **Amaç:** Verileri belirli bir aralığa sıkıştırarak, farklı değişkenlerin ağırlıklarını eşitlemek.
* **Formül (Min-Max Normalizasyon):**
* x' = (x - min(x)) / (max(x) - min(x))
  + x': Normalize edilmiş değer
  + x: Orjinal değer
  + min(x): Verinin minimum değeri
  + max(x): Verinin maksimum değeri

**Standardizasyon ve Normalizasyon Arasındaki Farklar**

|  |  |  |
| --- | --- | --- |
| Özellik | Standardizasyon | Normalizasyon |
| **Amaç** | Verileri aynı ölçeğe getirme | Verileri belirli bir aralığa sıkıştırma |
| **Dağılım** | Normal dağılıma uygun | Her türlü dağılıma uygun |
| **Ölçek** | Ortalama 0, standart sapma 1 | 0 ile 1 arasında |
| **Aykırı değerler** | Aykırı değerlerin etkisini azaltabilir | Aykırı değerlerin etkisini azaltamayabilir |

**Neden Standardizasyon ve Normalizasyon Kullanılır?**

* **Farklı ölçekteki değişkenlerin etkilerini eşitlemek:** Örneğin, bir veri setinde yaş ve gelir gibi farklı birimlerdeki değişkenler varsa, standardizasyon veya normalizasyon bu değişkenlerin model üzerindeki etkisini dengelemeye yardımcı olur.
* **Algoritmaların performansını artırmak:** Bazı algoritmalar (örneğin, k-means kümeleme, yapay sinir ağları) ölçeklenmiş verilere daha iyi yanıt verir.
* **Hesaplama hızını artırmak:** Büyük veri setlerinde, standardizasyon veya normalizasyon hesaplama süresini kısaltmaya yardımcı olabilir.
* **Daha iyi görselleştirme:** Standardize edilmiş veya normalize edilmiş veriler, daha iyi görselleştirme sağlar.

**Önemi**

Standardizasyon ve normalizasyon, makine öğrenmesi ve istatistik gibi alanlarda verilerin analiz edilmesi ve modellenmesi için kritik öneme sahiptir. Bu teknikler, verilerin daha anlamlı hale getirilmesini, farklı değişkenlerin karşılaştırılmasını ve modelin performansının artırılmasını sağlar.

**Hangi yöntemi kullanmalıyım?**

* **Normal dağılım:** Verileriniz normal dağılıma yakınsa, standardizasyon genellikle daha iyi bir seçenektir.
* **Aykırı değerler:** Aykırı değerleriniz varsa, standardizasyon aykırı değerlerin etkisini azaltmaya yardımcı olabilir.
* **Algoritma:** Kullandığınız algoritmanın gereksinimlerine göre seçim yapmanız önemlidir. Örneğin, bazı algoritmalar belirli bir aralıktaki verilere daha iyi yanıt verir.

**Sonuç olarak,** standardizasyon ve normalizasyon, veri ön işleme sürecinde sıklıkla kullanılan ve verilerin analiz edilebilir hale getirilmesinde önemli rol oynayan iki temel tekniktir. Veri setinizin özelliklerine ve kullanacağınız algoritmaya göre uygun yöntemi seçerek daha doğru ve güvenilir sonuçlar elde edebilirsiniz.

**Ek Not:** Veri ön işleme sadece standardizasyon ve normalizasyonla sınırlı değildir. Eksik değerlerin doldurulması, aykırı değerlerin tespiti ve çıkarılması gibi diğer teknikler de veri ön işleme sürecinde kullanılır.

1. Veri Bölme

**70-15-15 Oranının Belirlenmesi ve Veri Bölümünün Temelleri**

Harika bir soru! 70-15-15 oranının neden bu kadar yaygın kullanıldığı ve veri setinin nasıl bölündüğü, makine öğrenmesinde sıkça merak edilen konulardandır.

**70-15-15 Oranının Belirlenmesi**

Bu oran, genel olarak makine öğrenmesi modellerinin performansını değerlendirmek için ampirik olarak belirlenmiş bir kıstasdır. Ancak, bu oran kesin bir kural değildir ve veri setinizin büyüklüğü, karmaşıklığı ve problemin doğasına göre değişebilir.

* **Deneyim ve En İyi Uygulamalar:** Yıllar içinde yapılan birçok araştırma ve deney, bu oranın genellikle iyi sonuçlar verdiğini göstermiştir.
* **Modelin Karmaşıklığı:** Çok karmaşık modeller için daha fazla eğitim verisi gerekebilirken, daha basit modeller için daha az veri yeterli olabilir.
* **Veri Setinin Büyüklüğü:** Büyük veri setlerinde daha küçük bir test seti bile anlamlı sonuçlar verebilir. Küçük veri setlerinde ise daha dikkatli olunması gerekebilir.
* **Hesaplama Gücü:** Büyük veri setlerinde tüm veriyi eğitim için kullanmak hesaplama açısından zor olabilir. Bu durumda, daha küçük bir eğitim seti kullanmak daha pratik olabilir.

**Veri Bölümünün Temelleri**

Veri seti, genellikle şu kriterlere göre bölünür:

* **Rastgelelik:** Veri örnekleri, herhangi bir bias (önyargı) yaratmamak için rastgele seçilir.
* **Temsiliyet:** Her bir alt küme (train, validation, test), orijinal veri setinin özelliklerini temsil etmelidir. Örneğin, eğer veri setinde sınıf dengesizliği varsa, her alt kümede de aynı oranda sınıf dağılımı olması hedeflenir.
* **Bağımsızlık:** Test seti, modelin eğitimi sırasında hiç görülmemiş verilerden oluşmalıdır. Bu, modelin genelleme yeteneğini daha doğru bir şekilde ölçmemizi sağlar.

**Veri Bölme Yöntemleri**

* **Hold-out Method:** Veri seti, doğrudan train, validation ve test setlerine bölünür.
* **K-Fold Cross-Validation:** Veri seti, k eşit parçaya bölünür ve her bir parça sırayla test seti olarak kullanılırken diğer parçalar eğitim seti olarak kullanılır. Bu yöntem, özellikle küçük veri setleri için daha güvenilir sonuçlar verir.
* **Stratified K-Fold Cross-Validation:** Eğer veri setinde sınıf dengesizliği varsa, her katmanda sınıfların dağılımı korunarak k-fold cross-validation uygulanır.

**Özetle**

70-15-15 oranı, makine öğrenmesi modellerinin değerlendirilmesi için yaygın olarak kullanılan bir kıstas olsa da, kesin bir kural değildir. Veri setinizin özelliklerine ve probleminize göre bu oranı değiştirebilirsiniz. Veri bölümü yaparken rastgelelik, temsiliyet ve bağımsızlık gibi temel prensiplere dikkat etmek önemlidir.

**Örnek Kod (Python, scikit-learn kullanarak)**

from sklearn.model\_selection import train\_test\_split

# X: Görüntü verileri

# y: Etiketler (nesne sınıfları)

# İlk olarak train ve test setlerini ayırıyoruz

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Daha sonra train setini train ve validation setlerine ayırıyoruz

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.1765, random\_state=42) # Yaklaşık %15 validation

2.2) **Pratikte Veri Seti Bölümü ve Dağılım Probleminin Çözümü**

**Veri setini pratikte nasıl böleriz ve dağılım sorununu nasıl çözeriz?** Bu, makine öğrenmesi projelerinde sıkça karşılaşılan bir sorudur. İşte size bu konuda bazı pratik öneriler ve yöntemler daha önce de bahsettiğimz:

**1. Scikit-learn Kütüphanesini Kullanma:**

* **train\_test\_split:** Bu fonksiyon, veri setini eğitim ve test setlerine basitçe böler. Stratifikasyon (stratification) parametresi ile sınıf dağılımını koruyarak bölme yapabilirsiniz.
* **KFold:** K-katlamalı çapraz doğrulama için kullanılır. Veri seti k eşit parçaya bölünür ve her parça sırayla test seti olarak kullanılırken diğer parçalar eğitim seti olur. Bu yöntem, modelin performansını daha güvenilir bir şekilde değerlendirmeye yardımcı olur.
* **StratifiedKFold:** Stratifikasyonlu k-katlamalı çapraz doğrulama, sınıf dağılımını koruyarak k-fold cross-validation yapar.

**Örnek Kod:**

Python

from sklearn.model\_selection import train\_test\_split, StratifiedKFold

# X: Özellik matrisi

# y: Hedef değişken

# Stratifikasyonlu eğitim ve test seti bölümü

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=42)

# Stratifikasyonlu 5-katlamalı çapraz doğrulama

skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

for train\_index, test\_index in skf.split(X, y):

X\_train, X\_test = X[train\_index], X[test\_index]

y\_train, y\_test = y[train\_index], y[test\_index]

# Model eğitimi ve değerlendirmesi

**2. Dağılım Probleminin Çözümü:**

* **Stratifikasyon:** Sınıf dağılımını koruyarak veriyi bölmek, özellikle dengesiz veri setlerinde önemlidir.
* **Oversampling:** Azınlık sınıfındaki örnekleri çoğaltarak sınıf dengesini sağlama.
* **Undersampling:** Çoğunluk sınıfındaki örnekleri azaltarak sınıf dengesini sağlama.
* **SMOTE (Synthetic Minority Over-sampling Technique):** Azınlık sınıfına yeni sentetik örnekler oluşturarak oversampling yapmak.
* **Class Weighting:** Model eğitimi sırasında sınıfların ağırlıklarını ayarlayarak, azınlık sınıfına daha fazla önem vermek.

**3. Diğer Önemli Noktalar:**

* **Rastgelelik:** Veri bölümü işlemi rastgele yapılmalıdır. Bu, modelin farklı bölümlerdeki verilere maruz kalmasını sağlar ve genelleme yeteneğini artırır.
* **Temsiliyet:** Her bir alt küme (eğitim, doğrulama, test), orijinal veri setinin özelliklerini temsil etmelidir.
* **Bağımsızlık:** Test seti, modelin eğitimi sırasında hiç görülmemiş verilerden oluşmalıdır.
* **Hiperparametre Tuning:** Validation seti üzerinde farklı hiperparametre kombinasyonlarını deneyerek en iyi performansı veren modeli seçebilirsiniz.

**4. Özel Durumlar:**

* **Zaman Serisi Verileri:** Geçmiş verilerden geleceği tahmin etmek için kullanılan verilerde, zaman sırasına göre bölme yapılmalıdır.
* **Küçük Veri Setleri:** K-fold cross-validation gibi yöntemler, küçük veri setlerinde daha iyi sonuçlar verebilir.

**Sonuç olarak,** veri setini bölme ve dağılım sorununu çözme konusunda birçok yöntem bulunmaktadır. Hangi yöntemin kullanılacağı, veri setinizin özelliklerine, probleminize ve modelinize bağlıdır. Bu konuda deneyim kazanmak ve farklı yöntemleri denemek önemlidir.

**Ek Bilgiler:**

* **Imbalanced Learn:** Python'da dengesiz veri setleri için birçok yöntem sunan bir kütüphane.
* **imblearn.over\_sampling:** Oversampling teknikleri için.
* **imblearn.under\_sampling:** Undersampling teknikleri için.

1. Projedeki tek nöron vs 2 nörnon problemi: binary vs categorical multiclass pratik deneme ve notları:

**Kod:**

from sklearn.metrics import precision\_score, recall\_score, f1\_score

# Farklı optimizers ve batch\_size için test fonksiyonu

def test\_with\_different\_params(model\_type='binary', optimizer='adam', batch\_size=32):

# Model seçimi

if model\_type == 'binary':

model = create\_amazon\_review\_model() # Binary sınıflandırma modeli

y\_train\_labels = y\_train

y\_test\_labels = y\_test

loss = 'binary\_crossentropy'

else:

model = create\_multiclass\_amazon\_review\_model() # Multiclass sınıflandırma modeli

y\_train\_labels = y\_train\_multiclass

y\_test\_labels = y\_test\_multiclass

loss = 'categorical\_crossentropy'

# Model derleme (farklı optimizer ve loss ile)

model.compile(loss=loss, optimizer=optimizer, metrics=['accuracy'])

# Modeli eğit

history = model.fit(X\_train\_pad, y\_train\_labels, epochs=5, batch\_size=batch\_size, validation\_split=0.2, verbose=1)

# Tahminler ve değerlendirme

if model\_type == 'binary':

y\_pred = (model.predict(X\_test\_pad) > 0.5).astype('int32')

y\_true = y\_test

else:

y\_pred = np.argmax(model.predict(X\_test\_pad), axis=1)

y\_true = np.argmax(y\_test\_multiclass.values, axis=1)

# Performans metrikleri

precision = precision\_score(y\_true, y\_pred, average='binary' if model\_type == 'binary' else 'macro')

recall = recall\_score(y\_true, y\_pred, average='binary' if model\_type == 'binary' else 'macro')

f1 = f1\_score(y\_true, y\_pred, average='binary' if model\_type == 'binary' else 'macro')

print(f"{model\_type.capitalize()} - Optimizer: {optimizer}, Batch Size: {batch\_size}")

print(f"Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}\n")

# Farklı batch size ve optimizer'larla test

test\_with\_different\_params(model\_type='binary', optimizer='adam', batch\_size=32)

test\_with\_different\_params(model\_type='binary', optimizer='sgd', batch\_size=64)

test\_with\_different\_params(model\_type='multiclass', optimizer='adam', batch\_size=32)

test\_with\_different\_params(model\_type='multiclass', optimizer='sgd', batch\_size=64)

**Output:**

Epoch 1/5

/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input\_length` is deprecated. Just remove it.

warnings.warn(

**3889/3889** ━━━━━━━━━━━━━━━━━━━━ **76s** 19ms/step - accuracy: 0.8983 - loss: 0.2650 - val\_accuracy: 0.9164 - val\_loss: 0.2116

Epoch 2/5

**3889/3889** ━━━━━━━━━━━━━━━━━━━━ **74s** 19ms/step - accuracy: 0.9388 - loss: 0.1536 - val\_accuracy: 0.9123 - val\_loss: 0.2443

Epoch 3/5

**3889/3889** ━━━━━━━━━━━━━━━━━━━━ **73s** 19ms/step - accuracy: 0.9771 - loss: 0.0627 - val\_accuracy: 0.9070 - val\_loss: 0.3688

Epoch 4/5

**3889/3889** ━━━━━━━━━━━━━━━━━━━━ **76s** 19ms/step - accuracy: 0.9891 - loss: 0.0306 - val\_accuracy: 0.8934 - val\_loss: 0.4334

Epoch 5/5

**3889/3889** ━━━━━━━━━━━━━━━━━━━━ **75s** 19ms/step - accuracy: 0.9939 - loss: 0.0183 - val\_accuracy: 0.8916 - val\_loss: 0.5995

**1216/1216** ━━━━━━━━━━━━━━━━━━━━ **5s** 4ms/step

Binary - Optimizer: adam, Batch Size: 32

Precision: 0.94, Recall: 0.94, F1 Score: 0.94

/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input\_length` is deprecated. Just remove it.

warnings.warn(

Epoch 1/5

**1945/1945** ━━━━━━━━━━━━━━━━━━━━ **36s** 18ms/step - accuracy: 0.8743 - loss: 0.4150 - val\_accuracy: 0.8760 - val\_loss: 0.3759

Epoch 2/5

**1945/1945** ━━━━━━━━━━━━━━━━━━━━ **36s** 19ms/step - accuracy: 0.8739 - loss: 0.3787 - val\_accuracy: 0.8760 - val\_loss: 0.3733

Epoch 3/5

**1945/1945** ━━━━━━━━━━━━━━━━━━━━ **35s** 18ms/step - accuracy: 0.8751 - loss: 0.3745 - val\_accuracy: 0.8760 - val\_loss: 0.3716

Epoch 4/5

**1945/1945** ━━━━━━━━━━━━━━━━━━━━ **36s** 18ms/step - accuracy: 0.8751 - loss: 0.3721 - val\_accuracy: 0.8760 - val\_loss: 0.3690

Epoch 5/5

**1945/1945** ━━━━━━━━━━━━━━━━━━━━ **36s** 18ms/step - accuracy: 0.8751 - loss: 0.3680 - val\_accuracy: 0.8760 - val\_loss: 0.3614

**1216/1216** ━━━━━━━━━━━━━━━━━━━━ **7s** 5ms/step

Binary - Optimizer: sgd, Batch Size: 64

Precision: 0.87, Recall: 1.00, F1 Score: 0.93

/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input\_length` is deprecated. Just remove it.

warnings.warn(

Epoch 1/5

**3889/3889** ━━━━━━━━━━━━━━━━━━━━ **76s** 19ms/step - accuracy: 0.8978 - loss: 0.2630 - val\_accuracy: 0.9116 - val\_loss: 0.2169

Epoch 2/5

**3889/3889** ━━━━━━━━━━━━━━━━━━━━ **75s** 19ms/step - accuracy: 0.9379 - loss: 0.1577 - val\_accuracy: 0.9121 - val\_loss: 0.2342

Epoch 3/5

**3889/3889** ━━━━━━━━━━━━━━━━━━━━ **75s** 19ms/step - accuracy: 0.9753 - loss: 0.0673 - val\_accuracy: 0.9011 - val\_loss: 0.3251

Epoch 4/5

**3889/3889** ━━━━━━━━━━━━━━━━━━━━ **75s** 19ms/step - accuracy: 0.9892 - loss: 0.0306 - val\_accuracy: 0.9020 - val\_loss: 0.4612

Epoch 5/5

**3889/3889** ━━━━━━━━━━━━━━━━━━━━ **74s** 19ms/step - accuracy: 0.9934 - loss: 0.0188 - val\_accuracy: 0.8986 - val\_loss: 0.5490

**1216/1216** ━━━━━━━━━━━━━━━━━━━━ **4s** 4ms/step

Multiclass - Optimizer: adam, Batch Size: 32

Precision: 0.76, Recall: 0.74, F1 Score: 0.75

/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input\_length` is deprecated. Just remove it.

warnings.warn(

Epoch 1/5

**1945/1945** ━━━━━━━━━━━━━━━━━━━━ **36s** 18ms/step - accuracy: 0.8752 - loss: 0.3986 - val\_accuracy: 0.8760 - val\_loss: 0.3734

Epoch 2/5

**1945/1945** ━━━━━━━━━━━━━━━━━━━━ **36s** 18ms/step - accuracy: 0.8738 - loss: 0.3768 - val\_accuracy: 0.8760 - val\_loss: 0.3706

Epoch 3/5

**1945/1945** ━━━━━━━━━━━━━━━━━━━━ **35s** 18ms/step - accuracy: 0.8732 - loss: 0.3741 - val\_accuracy: 0.8760 - val\_loss: 0.3645

Epoch 4/5

**1945/1945** ━━━━━━━━━━━━━━━━━━━━ **34s** 18ms/step - accuracy: 0.8761 - loss: 0.3583 - val\_accuracy: 0.8760 - val\_loss: 0.3420

Epoch 5/5

**1945/1945** ━━━━━━━━━━━━━━━━━━━━ **34s** 18ms/step - accuracy: 0.8750 - loss: 0.3319 - val\_accuracy: 0.8772 - val\_loss: 0.3066

**1216/1216** ━━━━━━━━━━━━━━━━━━━━ **5s** 4ms/step

Multiclass - Optimizer: sgd, Batch Size: 64

Precision: 0.82, Recall: 0.51, F1 Score: 0.48

**1. Modelin Embedding Shape'i Hakkında Detaylı Teknik Bilgi**

Kodunda kullanılan embedding katmanı şu şekilde tanımlanmış:

model.add(Embedding(input\_dim=max\_words, output\_dim=128, input\_length=max\_len))

Bu embedding katmanının detaylarını teknik olarak inceleyelim:

* **input\_dim (max\_words)**: Bu, modelin kelime dağarcığı boyutudur. Burada max\_words = 5000 olarak belirlenmiş. Bu demek oluyor ki, model 5000 kelimeden oluşan bir kelime dağarcığı oluşturacak ve her kelimeye bir vektör karşılığı atayacak. Yani, tokenize edilmiş metinler bu 5000 kelimelik vektör uzayında temsil edilecek.
* **output\_dim**: Bu parametre, her kelimenin embedding vektörünün boyutunu belirler. Burada output\_dim = 128 olarak seçilmiş. Yani her kelime, 128 boyutlu bir vektörle temsil edilecek. Bu 128 boyut, kelimeler arası anlamsal ilişkileri öğrenmek için kullanılan bir latent (gizli) uzaydır.
* **input\_length (max\_len)**: Bu, modelin her girdinin uzunluğunun sabit olduğu varsayımıdır. Burada max\_len = 200 olarak belirlenmiş. Yani, her inceleme metni en fazla 200 kelimeden oluşacak şekilde padding uygulanmış. Eğer bir inceleme 200 kelimeden kısa ise, geri kalan kısımlar sıfır ile doldurulacak; uzun ise kesilecek.

**Embedding Katmanının Output Shape'i:**

* **Girdi Shape'i**: (batch\_size, max\_len)
  + batch\_size her iterasyonda modele verilen örnek sayısını belirtir (bu örnekte batch\_size = 32).
  + max\_len ise her inceleme metninin uzunluğunu (200) ifade eder.
* **Çıktı Shape'i**: (batch\_size, max\_len, output\_dim)
  + batch\_size 32 olarak belirlenmiş.
  + max\_len = 200 olduğu için her inceleme metni 200 kelime olacak.
  + output\_dim = 128 olduğundan, her kelime 128 boyutlu bir vektörle temsil edilecek.

Özetle, **Embedding katmanının çıktısı** (batch\_size, max\_len, output\_dim) yani (32, 200, 128) şeklindedir. Bu da her incelemede 200 kelimenin 128 boyutlu vektörlerle temsil edildiğini gösterir.

**2. 5. Katmandaki Dense Layer Nedir ve 1 Yerine 2 Olsaydı Ne Olurdu?**

Kodda 5. katmanda kullanılan Dense katmanı şu şekildedir:

model.add(Dense(1, activation='sigmoid'))

Bu katman, sentiment (duygu analizi) yapmak için kullanılıyor ve binary sınıflandırma (olumlu/olumsuz) yapıyor. Şimdi bu katmanın detaylarına bakalım:

* **Dense(1)**: Bu katman, 1 adet çıkış birimi (nöron) içeriyor. Bu nöron, sınıflandırma sonucunu (0 veya 1) temsil ediyor.
* **activation='sigmoid'**: Sigmoid aktivasyon fonksiyonu, çıkışı 0 ile 1 arasında bir olasılık olarak verir. 0'a yakın değerler "negatif sınıfı" (olumsuz inceleme), 1'e yakın değerler "pozitif sınıfı" (olumlu inceleme) gösterir.

**Eğer Dense(1) Yerine Dense(2) Olsaydı Ne Değişirdi?**

Eğer 1 yerine 2 nöron olsaydı, bu modelin çıkışı iki sınıf arasında olasılık dağılımı yapacak şekilde olurdu, yani model **multiclass** sınıflandırma yapmaya başlardı. Bu durumda, aktivasyon fonksiyonu olarak sigmoid yerine genellikle softmax kullanılır.

* **Binary Sınıflandırma (Dense(1))**:
  + Tek bir nöron, sigmoid ile çalışır ve çıkışı iki sınıftan biri (0 veya 1) olarak verir.
  + Loss fonksiyonu binary\_crossentropy olur.
* **Multiclass Sınıflandırma (Dense(2))**:
  + İki nöron olur, her biri bir sınıf için çalışır ve çıkışı softmax ile iki sınıfın olasılık dağılımı şeklinde verir.
  + Bu durumda, loss fonksiyonu **categorical\_crossentropy** olur.

**Loss Fonksiyonu: Binary Crossentropy Nedir?**

binary\_crossentropy, ikili sınıflandırma problemleri için kullanılan bir loss (kayıp) fonksiyonudur. Modelin yaptığı tahmin ile gerçek etiket arasındaki farkı ölçer. Cross-entropy'nin amacı, modelin tahmin ettiği olasılık dağılımını, gerçek dağılımla kıyaslayarak modele hata sinyali vermektir.

Formül olarak:

* gerçek etiket (0 veya 1),
* \hat{y\_i} modelin tahmin ettiği olasılık (sigmoid fonksiyonunun sonucu).

Eğer Dense(2) olsaydı ve çok sınıflı bir problem olsaydı, bu durumda categorical\_crossentropy kullanılırdı. Bu, çoklu sınıflar arasındaki farkı ölçmek için kullanılır.

**Matematiksel İfade:**

* y\_i: i. sınıfın gerçek olasılığı (one-hot kodlamasında 0 veya 1)
* p\_i: i. sınıfın tahmin edilen olasılığı

Kategorik çapraz entropi kaybı şu şekilde hesaplanır:

Kayıp = - Σ(y\_i \* log(p\_i))

Toplam, tüm sınıflar üzerinde alınır.

**Neden Kategorik Çapraz Entropi Kullanılır?**

* **Tahmin Hatasını Doğrudan Ölçer:** Tahmin edilen olasılık dağılımı ile gerçek dağılım arasındaki farkı doğrudan hesaplar.
* **Çoklu Sınıf Sınıflandırmaya Uygun:** Birden fazla sınıf içeren problemlerde kullanılabilir.
* **Türevlenebilir:** Gradyan tabanlı optimizasyon yöntemleri (örneğin, geri yayılım) ile kullanılabilir.

**Örnek:**

3 sınıflı bir sınıflandırma problemi düşünelim. Gerçek sınıf ikinci sınıf olsun ve model şu olasılıkları tahmin etsin:

* Sınıf 1: 0.2
* Sınıf 2: 0.3
* Sınıf 3: 0.5

One-hot kodlanmış gerçek etiket: [0, 1, 0]

Çapraz entropi kaybı:

Kayıp = -(0 \* log(0.2) + 1 \* log(0.3) + 0 \* log(0.5))

Model bu kaybı en aza indirerek daha doğru tahminler yapmayı öğrenir.

**Özetle:** Kategorik çapraz entropi kaybı, derin öğrenme modellerinin sınıflandırma problemlerinde başarılı bir şekilde eğitilmesini sağlayan önemli bir araçtır. Modelin tahmin ettiği olasılık dağılımı ile gerçek dağılım arasındaki farkı ölçerek, modelin öğrenme sürecini yönlendirir.

Bu proje, **binary** (ikili) sınıflandırma için hazırlanmış, yani bir incelemenin olumlu ya da olumsuz olarak sınıflandırılması amaçlanıyor. Bu nedenle **Dense(1)** kullanılmış ve aktivasyon fonksiyonu olarak **sigmoid** seçilmiş. Eğer problem iki sınıf yerine birden fazla sınıf içeriyor olsaydı, model **multiclass** sınıflandırma yapacak şekilde ayarlanır ve **Dense(2)** veya daha fazla nöron kullanılarak her sınıf için olasılıklar tahmin edilirdi. Şimdi senin için hem binary hem de multiclass senaryolarını pratik bir şekilde kıyaslayacağız ve performans metriklerini hesaplayacağız.

**Plan:**

1. **Binary Sınıflandırma** ile mevcut yapı üzerinden sonuçları inceleyeceğiz.
2. **Multiclass Sınıflandırma** için modeli düzenleyip, performans farkını analiz edeceğiz.
3. **Precision, Recall, F1 Score** hesaplayarak, sonuçları karşılaştıracağız.
4. **Batch Size ve Optimizer'ları** değiştirerek etkilerini göreceğiz.

**Adım Adım Pratik Çalışma**

**Adım 1: Binary Sınıflandırma (Mevcut Model)**

from sklearn.metrics import precision\_score, recall\_score, f1\_score

# Binary sınıflandırma için model eğitildi (mevcut hali)

y\_pred\_binary = (model.predict(X\_test\_pad) > 0.5).astype('int32')

# Precision, Recall, F1 hesaplama

precision\_binary = precision\_score(y\_test, y\_pred\_binary)

recall\_binary = recall\_score(y\_test, y\_pred\_binary)

f1\_binary = f1\_score(y\_test, y\_pred\_binary)

print(f"Binary Sınıflandırma - Precision: {precision\_binary:.2f}, Recall: {recall\_binary:.2f}, F1 Score: {f1\_binary:.2f}")

Bu bölümde, mevcut **binary sınıflandırma** yapısını kullanıyoruz. **Sigmoid aktivasyonu** ile tek bir nöron çıkışı 0 veya 1 (olumlu/olumsuz) olarak sonuçlanıyor. Precision, recall ve F1 score bu çıkışa göre hesaplanıyor.

**Adım 2: Multiclass Sınıflandırma**

Şimdi modelde **Dense(2)** kullanarak iki sınıf arasında olasılık dağılımı yapacağız (örnek olarak 3'ten düşük derecelendirmeler negative sınıfı, 3 ve üzeri positive sınıfı olacak şekilde).

**Modelin yeniden yapılandırılması:**

from tensorflow.keras.layers import Softmax

def create\_multiclass\_amazon\_review\_model():

model = Sequential()

# 1. Katman: Embedding

model.add(Embedding(input\_dim=max\_words, output\_dim=128, input\_length=max\_len))

# 2. Katman: Ara Katman 1

model.add(Flatten())

# 3. Katman: Ara Katman 2

model.add(Dense(64, activation='relu'))

# 4. Katman: Ara Katman 3

model.add(Dense(32, activation='relu'))

# 5. Katman: Multiclass çıkış için Softmax

model.add(Dense(2, activation='softmax'))

# Modeli derleyelim

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

# Yeni modeli oluştur ve eğit

multiclass\_model = create\_multiclass\_amazon\_review\_model()

# Etiketleri yeniden düzenleyelim (2 sınıfa uygun hale getirme)

y\_train\_multiclass = pd.get\_dummies(y\_train)

y\_test\_multiclass = pd.get\_dummies(y\_test)

# Yeni modeli eğitelim

history\_multiclass = multiclass\_model.fit(X\_train\_pad, y\_train\_multiclass, epochs=5, batch\_size=32, validation\_split=0.2, verbose=1)

# Multiclass sınıflandırma tahminleri

y\_pred\_multiclass = multiclass\_model.predict(X\_test\_pad)

y\_pred\_multiclass\_labels = np.argmax(y\_pred\_multiclass, axis=1)

# Precision, Recall, F1 hesaplama

precision\_multiclass = precision\_score(y\_test, y\_pred\_multiclass\_labels)

recall\_multiclass = recall\_score(y\_test, y\_pred\_multiclass\_labels)

f1\_multiclass = f1\_score(y\_test, y\_pred\_multiclass\_labels)

print(f"Multiclass Sınıflandırma - Precision: {precision\_multiclass:.2f}, Recall: {recall\_multiclass:.2f}, F1 Score: {f1\_multiclass:.2f}")

* **Categorical Crossentropy**: Multiclass sınıflandırmada kullanılan bir loss fonksiyonudur. Modelin, bir örneği birden fazla sınıf arasında doğru şekilde ayırt etmesine yardımcı olur.
* **Softmax aktivasyonu**: Her sınıfın olasılığını verir ve toplamı 1 olacak şekilde normalize eder.

**Adım 3: Farklı Batch Size ve Optimizer'larla Deneme**

Şimdi batch size ve optimizer değişikliklerinin performansa nasıl etki ettiğini inceleyelim.

**Deneme: Batch Size ve Optimizer'ı Değiştirme**

# Batch size 64 ve optimizer 'SGD' olarak değiştirelim

model.compile(loss='binary\_crossentropy', optimizer='SGD', metrics=['accuracy'])

history\_sgd = model.fit(X\_train\_pad, y\_train, epochs=5, batch\_size=64, validation\_split=0.2, verbose=1)

# Tahminler yap ve performansı ölç

y\_pred\_sgd = (model.predict(X\_test\_pad) > 0.5).astype('int32')

accuracy\_sgd = accuracy\_score(y\_test, y\_pred\_sgd)

# Precision, Recall, F1 hesaplama

precision\_sgd = precision\_score(y\_test, y\_pred\_sgd)

recall\_sgd = recall\_score(y\_test, y\_pred\_sgd)

f1\_sgd = f1\_score(y\_test, y\_pred\_sgd)

print(f"Batch Size 64 ve SGD Optimizer - Accuracy: {accuracy\_sgd:.2f}")

print(f"Precision: {precision\_sgd:.2f}, Recall: {recall\_sgd:.2f}, F1 Score: {f1\_sgd:.2f}")

* **Optimizer Değişikliği**: SGD yerine adam kullanıldığında model genellikle daha hızlı öğrenir çünkü adam adaptif öğrenme hızına sahiptir.
* **Batch Size Değişikliği**: Batch size 32 yerine 64 yapılması, her iterasyonda daha fazla veri işlenmesine neden olur, bu da eğitim hızını etkileyebilir ancak daha fazla bellek kullanır.

**Bu incelemenin Sonuçları:**

1. **Binary Sınıflandırma** (Tek nöron) kullanıldığında, precision, recall ve F1 score metrikleri iki sınıflı bir problemde olasılık tahmin eder.
2. **Multiclass Sınıflandırma** (İki nöron) kullanıldığında, model iki farklı sınıf arasında olasılık dağılımı yapar ve genellikle softmax ve categorical\_crossentropy kullanılır.
3. **Optimizer ve Batch Size** değişiklikleri performansı doğrudan etkileyebilir, her model ve veri seti için en iyi kombinasyonu bulmak için çeşitli denemeler yapılmalıdır.

**Bu Sonuçların Detaylı Açıklaması ve Sebepleri**

**Binary Sınıflandırmada Adam Optimizatörünün Daha Hızlı ve Doğru Olmasının Sebepleri:**

1. **Adam'ın Öğrenme Oranı Uyarlamaları**: Adam, öğrenme oranını adaptif olarak günceller. Bu özellik, özellikle binary sınıflandırma gibi hızlı öğrenmeyi gerektiren durumlarda eğitim sürecini hızlandırır ve daha hızlı yakınsama sağlar.
2. **Binary Cross-Entropy ile Uyum**: Binary sınıflandırmada her tahmin ya “0” ya da “1” olacak şekilde yapılır. Adam optimizasyon algoritması, bu gibi iki sınıf arasındaki büyük farklılıklarda daha etkin bir şekilde çalışır.
3. **Küçük Batch Size ile Verimli Öğrenme**: Daha küçük batch size, her iterasyonda modele daha fazla güncelleme yapılmasını sağlar. Bu, binary sınıflandırmada modelin eğitimi hızlandırır ve daha doğru sonuçlara ulaşmayı kolaylaştırır. Küçük batch size ayrıca modelin veri değişimlerine daha hızlı yanıt vermesine olanak tanır.

**Multiclass Sınıflandırmada Adam ve Categorical Cross-Entropy İle Daha Stabil Performans Sağlanmasının Sebepleri:**

1. **Softmax Çıkışı ile Çoklu Olasılık Tahmini**: Multiclass sınıflandırmada softmax, her sınıf için olasılık hesaplaması yapar. Adam optimizasyon algoritması, softmax ile birleştiğinde her sınıf için ağırlıkları hızlı ve doğru bir şekilde günceller, böylece her sınıfın doğru bir şekilde öğrenilmesine olanak tanır.
2. **Categorical Cross-Entropy'nin Çok Sınıflı Yapıya Uygunluğu**: Bu loss fonksiyonu, her sınıf için olasılık farklarını minimize ederek tüm sınıfların dengeli bir şekilde öğrenilmesini sağlar. Bu nedenle, özellikle birden fazla sınıfın olduğu durumlarda modelin stabil ve dengeli bir performans göstermesine yardımcı olur.
3. **SGD ile Eğitimde Doğruluğun Düşmesi**: SGD (Stochastic Gradient Descent), her iterasyonda yalnızca tek bir örneği güncelleyerek ağırlıkları günceller. Bu, çok sınıflı sorunlarda daha uzun sürede öğrenmeye yol açar ve doğruluğu düşürebilir çünkü ağırlık güncellemeleri yavaş ve daha rastlantısal olur. Sonuç olarak, Adam gibi adaptif bir optimizör bu tip durumlarda daha iyi sonuç verir.

**Genel Sonuç:**

Bu yapı, Adam optimizörü ve batch size parametrelerinin, sınıflandırma türüne göre nasıl bir etki yaptığını net bir şekilde gösterir:

* **Binary sınıflandırmada**: Adam ile hızlı ve doğru sonuç alınır; küçük batch size doğruluğu artırır.
* **Multiclass sınıflandırmada**: Adam optimizörü ve categorical cross-entropy kombinasyonu, SGD’ye kıyasla daha hızlı öğrenme ve stabil bir doğruluk sağlar.