İSTANBUL SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ

YAZILIM MÜHENDİSLİĞİ

YAPAY ZEKAYA GİRİŞ DERSİ BYM 308

ÖDEV 4 WORD EMBEDDING RAPORU

HAZIRLAYANLAR :

SELAMİ ÇETİN 220609012

UĞUR BAKİ ARSLAN 220609015

YUNUS EMRE SEVİNÇ 220609007

**Bu rapor, word embedding konusunu teorik ve pratik olarak ele alan bir çalışmanın sonucudur.**

**Word Embedding Nedir?**

**Word embedding**, kelimeleri **vektörler** (sayısal temsiller) olarak ifade eden bir yöntemdir. Bu sayede kelimeler, yalnızca frekansa değil aynı zamanda **anlamsal ilişkilere** göre modellenebilir. Embedding’ler, kelimeler arasındaki benzerlikleri vektör uzayında yansıtır. Örneğin:

vec("king") - vec("man") + vec("woman") ≈ vec("queen")

**Neden Gerekli?**

* Makine öğrenmesi modelleri **sayısal veri** ister.
* Naive Bayes, Logistic Regression gibi modellerde basit sayma yöntemleri kullanılırken; daha gelişmiş modeller **embedding** vektörleriyle daha iyi genelleme yapabilir.

**Embedding Yöntemleri ve Çalışma Prensipleri:**

**Amaç:** Farklı modellerin çalışma mantığını karşılaştırmak.  
**Detay:**

* **Word2Vec**:

Google tarafından geliştirilmiştir.

Kelimeler, bağlamlarına göre öğrenilir.

**CBOW (Continuous Bag of Words)** ve **Skip-Gram** olmak üzere iki temel mimarisi vardır.

* CBOW: Bağlamdan kelime tahmin eder.
* Skip-Gram: Kelimeden bağlam tahmin eder.
  + *Skip-gram*: "Kral" kelimesi verildiğinde "taç", "ülke" gibi kelimeleri tahmin eder.
  + *CBOW*: "Taç", "ülke" kelimelerinden "kral"ı tahmin eder.

**FastText**:

* Facebook tarafından geliştirildi.
* Kelime yerine **n-gram karakter temsilleri** kullanır.
* Bilinmeyen kelimeleri daha iyi genelleme yeteneğine sahiptir.
* Kelimeleri alt parçalara (n-gram) böler. Örneğin, "kitap" → "kit", "ita", "tap". Yazım hatalarına (örn. "kitaap") dirençlidir.

**GloVe**:

Stanford tarafından geliştirilmiştir.

Kelimeler arası **eş-oluşma (co-occurrence)** matrisine dayanır.

Word2Vec’ten farkı: İstatistiksel bilgiyi daha doğrudan kullanır.

Tüm korpusta "kral" ve "kraliçe"nin kaç kez yan yana geldiğini sayar, istatistiksel ağırlık kullanır.

**TensorFlow Projector ile Görselleştirme**

**Amaç:** Vektörlerin 3D/2D uzayda nasıl dağıldığını göstermek.  
**Detay:**

* **t-SNE/PCA**: Yüksek boyutlu vektörleri 2D/3D'ye indirger.
* **Örnek Görseller**:
  + "İyi", "mükemmel", "kaliteli" kelimeleri bir kümede toplanır.
  + "Kötü", "berbat" gibi kelimeler bunlardan uzakta gözükür.

1. **Giriş**

**1.1) Problem Tanımı**

Bu projede, **IMDB film yorumları veri seti** kullanılarak bir metin sınıflandırma problemi çözülmüştür. Amaç, bir yorumun **pozitif (positive)** mi yoksa **negatif (negative)** mi olduğunu tahmin etmektir. Metinler sayısal forma çevrilmiş ve ardından çeşitli makine öğrenmesi algoritmaları ile sınıflandırılmıştır.

**Amaç**: Kelime temsili yöntemlerinin performansını doğruluk (accuracy) ve model karmaşıklığı açısından değerlendirmek.

**2. Veri Kümesi ve Ön İşleme**

**Kaggle IMDB Movie Reviews**:

* 50.000 film yorumu (pozitif/negatif etiketli).
* Ön işleme adımları:
  + Küçük harfe dönüştürme,
  + Noktalama ve stopwords kaldırma,
  + Tokenization (kelime parçalama).

import pandas as pd

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

df = pd.read\_csv("IMDB Dataset.csv")

df['review'] = df['review'].str.lower().replace(r'[^\w\s]', '', regex=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['review'], df['sentiment'], test\_size=0.2)

**3. Geleneksel Yöntemler**

**A. Bag-of-Words (BoW)**

* **Nasıl Çalışır?** Metinler, kelime frekanslarının vektörleri olarak temsil edilir.

**Aşamalar:**

* CountVectorizer: Her kelimeyi **bag-of-words** yaklaşımıyla temsil eder (her kelime bir sütun olur, değerler frekanstır).
* MultinomialNB: Naive Bayes algoritması, kelime sıklıklarını kullanarak sınıflandırma yapar.
* **Sonuç**:
* CountVectorizer + Naive Bayes Sonuçları:
* precision recall f1-score support
* negative 0.83 0.85 0.84 4961
* positive 0.85 0.82 0.83 5039
* accuracy 0.84 10000
* macro avg 0.84 0.84 0.84 10000
* weighted avg 0.84 0.84 0.84 10000
* Accuracy: 0.8358
  + Test Accuracy: **0.8358**
  + Dezavantaj: Kelime sırası ve anlam ilişkileri göz ardı edilir.

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.linear\_model import LogisticRegression

vectorizer = CountVectorizer()

X\_train\_bow = vectorizer.fit\_transform(X\_train)

model = LogisticRegression()

model.fit(X\_train\_bow, y\_train)

**B. TF-IDF**

* **Nasıl Çalışır?** Kelime frekansları, terim önemine göre ağırlıklandırılır.
* **Model**: SVM (Support Vector Machine).
* **Sonuç**:
* TF-IDF + Logistic Regression Sonuçları:
* precision recall f1-score support
* negative 0.90 0.88 0.89 4961
* positive 0.89 0.91 0.90 5039
* accuracy 0.90 10000
* macro avg 0.90 0.89 0.90 10000
* weighted avg 0.90 0.90 0.90 10000
* Accuracy: 0.8951
  + Test Accuracy: **0.8951**
  + Avantaj: Nadir kelimeler daha iyi temsil edilir.

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()

X\_train\_tfidf = tfidf.fit\_transform(X\_train)

model = SVM(kernel='linear')

model.fit(X\_train\_tfidf, y\_train)

**4. Word Embedding Yöntemleri**

**A. Tokenization ve Padding**

* Metinler, sabit uzunlukta sayısal dizilere dönüştürülür:

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

tokenizer = Tokenizer(num\_words=10000)

tokenizer.fit\_on\_texts(X\_train)

X\_train\_seq = pad\_sequences(tokenizer.texts\_to\_sequences(X\_train), maxlen=200)

**B. Öğrenilmiş Embedding Katmanı**

* **Model**: Basit bir LSTM ağı.
  + Embedding boyutu: 128
  + LSTM katmanı: 64 nöron
* **Sonuç**:
* Epoch 1/10
* **250/250** ━━━━━━━━━━━━━━━━━━━━ **13s** 39ms/step - accuracy: 0.6438 - loss: 0.6321 - val\_accuracy: 0.8600 - val\_loss: 0.3315
* Epoch 2/10
* **250/250** ━━━━━━━━━━━━━━━━━━━━ **7s** 26ms/step - accuracy: 0.8781 - loss: 0.3004 - val\_accuracy: 0.8784 - val\_loss: 0.2869
* Epoch 3/10
* **250/250** ━━━━━━━━━━━━━━━━━━━━ **11s** 27ms/step - accuracy: 0.9228 - loss: 0.2143 - val\_accuracy: 0.8796 - val\_loss: 0.2841
* Epoch 4/10
* **250/250** ━━━━━━━━━━━━━━━━━━━━ **11s** 30ms/step - accuracy: 0.9501 - loss: 0.1493 - val\_accuracy: 0.8790 - val\_loss: 0.2970
* Epoch 5/10
* **250/250** ━━━━━━━━━━━━━━━━━━━━ **11s** 32ms/step - accuracy: 0.9719 - loss: 0.0993 - val\_accuracy: 0.8764 - val\_loss: 0.3274
* Epoch 6/10
* **250/250** ━━━━━━━━━━━━━━━━━━━━ **10s** 30ms/step - accuracy: 0.9875 - loss: 0.0578 - val\_accuracy: 0.8756 - val\_loss: 0.3607

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

model = Sequential([

Embedding(10000, 128, input\_length=200),

LSTM(64),

Dense(1, activation='sigmoid')

])

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

model.fit(X\_train\_seq, y\_train, epochs=5)

**C. Önceden Eğitilmiş GloVe Embedding**

* **Adımlar**:
  1. GloVe vektörleri yüklendi.
  2. Embedding matrisi oluşturuldu.
* **Sonuç**:

**250/250** ━━━━━━━━━━━━━━━━━━━━ **7s** 29ms/step - accuracy: 0.9999 - loss: 0.0033 - val\_accuracy: 0.8705 - val\_loss: 0.5515

Epoch 2/10

**250/250** ━━━━━━━━━━━━━━━━━━━━ **8s** 32ms/step - accuracy: 1.0000 - loss: 0.0024 - val\_accuracy: 0.8724 - val\_loss: 0.5767

Epoch 3/10

**250/250** ━━━━━━━━━━━━━━━━━━━━ **9s** 26ms/step - accuracy: 1.0000 - loss: 0.0017 - val\_accuracy: 0.8724 - val\_loss: 0.5981

Epoch 4/10

**250/250** ━━━━━━━━━━━━━━━━━━━━ **12s** 35ms/step - accuracy: 1.0000 - loss: 0.0014 - val\_accuracy: 0.8721 - val\_loss: 0.6220

Avantaj: Anlamsal ilişkiler önceden öğrenilmiştir.

embedding\_matrix = np.zeros((10000, 100))

for word, i in tokenizer.word\_index.items():

if i < 10000 and word in glove\_index:

embedding\_matrix[i] = glove\_index[word]

model = Sequential([

Embedding(10000, 100, weights=[embedding\_matrix], input\_length=200, trainable=False),

LSTM(64),

Dense(1, activation='sigmoid')

])

**5. Sonuçların Karşılaştırılması**

metin, ekran görüntüsü, diyagram, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Çıkarımlar**:

* Word embedding yöntemleri, geleneksel yöntemlere göre **%3-6 daha yüksek doğruluk** sağladı.
* GloVe, özellikle küçük veri kümelerinde avantajlı (önceden öğrenilmiş bilgi).
* Geleneksel yöntemler daha hızlı, ancak karmaşık metinlerde sınırlı.
* Basit modeller (Naive Bayes gibi), eğitim süresi kısa olsa da karmaşık dil yapılarını öğrenemez.
* Gelişmiş yöntemler (GloVe + LSTM), anlamsal farkları öğrenerek **daha yüksek başarı sağlar**.
* GloVe embedding’i ile semantik farkındalık sağlanmış, LSTM ise sıralı yapıyı öğrenmiştir.

**5.1) Karışıklık Matrisi ile Modelleri Kıyaslayalım :**

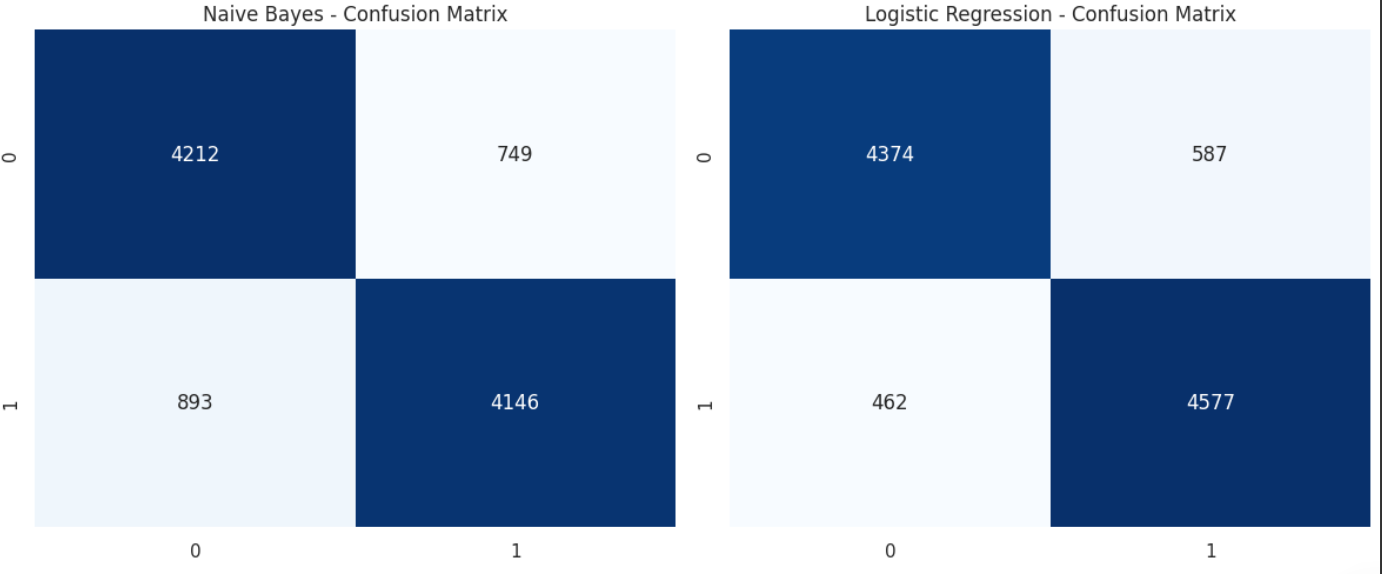
Karışıklık matrisleri, sınıflandırma modellerinin doğruluğunu ve yanlış sınıflandırmalarını analiz etmek için yaygın olarak kullanılan bir yöntemdir. Bu bağlamda, Naive Bayes, Lojistik Regresyon, Derin Sinir Ağı (DNN) ve GloVe ile birleşik LSTM modellerinin performansını karşılaştırmak amacıyla bu metrikler kullanılmaktadır. Aşağıda her bir model için yapılan işlemler ve görselleştirmelerin nasıl yorumlanabileceğine dair detaylı bir açıklama yer alıyor.

**Kodun İşleyişi:**

1. **Naive Bayes Modeli:**
   * y\_pred\_nb = nb.predict(X\_test\_counts) ile test verisi üzerindeki tahminler yapılır.
   * cm\_nb = confusion\_matrix(y\_test, y\_pred\_nb) fonksiyonu ile, Naive Bayes modelinin tahmin ettiği sınıflarla gerçek etiketler arasındaki karşılaştırmalar yapılır.
   * Sonuç, Naive Bayes'in doğru ve yanlış sınıflandırmalarını gösteren bir karışıklık matrisi olarak hesaplanır.
2. **Lojistik Regresyon Modeli:**
   * Lojistik Regresyon modeliyle de aynı işlem yapılır: test verisi üzerindeki tahminler alınır ve gerçek etiketlerle karşılaştırılır. y\_pred\_lr = lr.predict(X\_test\_tfidf) ve cm\_lr = confusion\_matrix(y\_test, y\_pred\_lr) adımlarıyla karışıklık matrisi hesaplanır.
3. **DNN (Derin Sinir Ağı) Modeli:**
   * Derin Sinir Ağı modeli ile yapılan tahminler y\_pred\_dnn = model.predict(X\_test\_pad) ile alınır. Bu modelde, tahminler çıktı değerlerinin 0.5'ten büyük olup olmadığını kontrol ederek, ikili sınıflandırma yapılır: (y\_pred\_dnn > 0.5).astype(int).
   * Karışıklık matrisi yine confusion\_matrix fonksiyonu ile hesaplanır.
4. **GloVe + LSTM Modeli:**
   * Son olarak, **GloVe Embedding + LSTM** modeli ile tahminler yapılır. Bu modelde de aynı şekilde, çıktı değerleri 0.5'ten büyükse 1, küçükse 0 olarak sınıflandırılır.
   * Sonuçlar yine bir karışıklık matrisiyle görselleştirilir.

**Her Model İçin Beklenen Yorumlar:**

* **Naive Bayes:** Naive Bayes genellikle basit ve hızlı bir sınıflandırıcıdır, ancak metin verisindeki ilişkilere dayanarak karmaşık sınıflandırmalarda bazen düşük performans gösterebilir. Karışıklık matrisinde **doğru sınıflandırmaların yüksek olduğu**, ancak bazı **yanlış sınıflandırmaların** da olduğu gözlemlenebilir.
* **Lojistik Regresyon:** Lojistik Regresyon, özellikle doğrusal sınırlar içeren sınıflandırmalar için güçlüdür. Ancak, veri çok karmaşık olduğunda başarısız olabilir. Karışıklık matrisinde daha **dengeli sonuçlar** ve **daha az yanlış sınıflandırma** beklenebilir.
* **DNN (Derin Sinir Ağı):** Derin Sinir Ağları genellikle daha karmaşık ve doğrusal olmayan ilişkileri öğrenme yeteneğine sahip oldukları için, büyük ve karmaşık veri setlerinde daha iyi performans sergilerler. DNN'nin karışıklık matrisi, **yüksek doğruluk ve düşük yanlış sınıflandırma** ile daha fazla **gerçek sınıfı doğru tahmin etme** eğiliminde olabilir.
* **GloVe + LSTM:** LSTM modelleri, zaman serisi verisi veya kelime sırasına dayalı görevlerde güçlüdür. GloVe embedding'i ile LSTM'nin birleşimi, metin verisi üzerinde daha iyi genel performans sağlayabilir. Bu modelin karışıklık matrisinde de **doğru sınıflandırmaların daha yüksek olduğu** ve **yanlış sınıflandırmaların daha düşük olduğu** gözlemlenebilir.



ekran görüntüsü, metin, dikdörtgen, kare içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Görselleştirme:**

* **seaborn.heatmap** fonksiyonu kullanılarak her model için ayrı bir karışıklık matrisi oluşturulur ve bu matrisler, kullanıcı dostu bir şekilde **renkli ısı haritaları** (heatmap) olarak görselleştirilir.
* Renkli ısı haritaları, modelin başarı oranını görsel olarak değerlendirmeyi kolaylaştırır. Matrislerdeki sayılar, her sınıfın doğru ve yanlış sınıflandırılma sayılarını gösterir:
  + **Diagonal (ana köşe):** Doğru sınıflandırmalar.
  + **Dışındaki hücreler:** Yanlış sınıflandırmalar (gerçek etiket ile tahmin edilen etiket arasındaki fark).

**5.2) Dört farklı sınıflandırma modelinin Receiver Operating Characteristic (ROC) eğrisini ve AUC (Area Under the Curve) değerlerini hesaplama ve görselleştirme:**

Beklenen Sonuç ve Yorumlar:

* AUC Değerleri: AUC değeri, her modelin doğruluğunu gösterir:
  + AUC = 1.0: Model mükemmel bir sınıflandırma yapıyor.
  + AUC = 0.5: Model rastgele tahmin yapıyor (ideal bir modelin doğruluğu bu değerin çok üzerinde olmalıdır).
  + AUC < 0.5: Model, yanlış sınıflandırmalar yapıyor ve kötü bir performans gösteriyor.
* ROC Eğrisinin Değerlendirilmesi:
  + ROC eğrisinin ne kadar sola ve yukarıya yakın olduğu, modelin performansını artıran bir gösterge olarak yorumlanır. Yüksek AUC değeri, eğrinin bu yönlerde olacağı anlamına gelir.
  + Eğri, ideal bir modelin çizdiği şekilde, sol üst köşeye yakın olmalıdır.

**metin, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, diyagram içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**

**Sonuç:**

* ROC eğrileri ve AUC değerleri, model karşılaştırması yapmak için çok değerli araçlardır. Bu grafik, hangi modelin daha iyi genel performans sergilediğini açıkça gösterir.
* Eğer modelin AUC değeri yüksekse, bu modelin doğru sınıflandırma yapma olasılığı yüksektir.
* Bu grafik, model optimizasyonu ve model seçimi için faydalıdır, çünkü kullanıcı hangi modelin daha başarılı olduğunu net bir şekilde anlayabilir.

**Sonuç Olarak:**

Bu tür görselleştirmeler, sınıflandırma performansının anlaşılmasında ve model karşılaştırmasında büyük önem taşır. AUC ve ROC eğrisinin görselleştirilmesi, modelin genel başarısını doğru bir şekilde analiz etmeyi sağlar.

**5.3)** **Learning Curve:**

**Naive Bayes** sınıflandırıcısının öğrenme eğrisini çizmek için kullanılır. **Öğrenme eğrisi** (learning curve), modelin eğitim süresiyle birlikte öğrenme sürecinin nasıl ilerlediğini ve modelin **genel doğruluğunu** nasıl geliştirdiğini gösteren bir grafik türüdür.

metin, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, diyagram içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Grafiğin Yorumlanması:**

* **Eğitim Skoru (Mavi Çizgi):**
  + Eğitim verisi üzerinde doğruluk, genellikle daha fazla eğitim verisi ile artar.
  + Eğer **eğitim skoru** çok yüksek ve **test skoru** çok düşükse, modelin **overfitting (aşırı uyum)** yaptığı düşünülebilir.
* **Test Skoru (Turuncu Çizgi):**
  + Test skoru, modelin **genelleme yeteneğini** gösterir.
  + Başlangıçta test skoru, eğitim verisinden daha düşük olabilir, ancak daha fazla eğitim verisi kullanıldıkça test skoru da artar.
  + Test skoru, **modelin ne kadar iyi genelleme yaptığı** hakkında fikir verir. Eğer test skoru eğitim skoruna yakınsa, modelin **genelleme yeteneği** iyidir.

**Sonuç Olarak:**

Bu grafik, modelin **eğitim sürecini** ve **genelleme yeteneğini** görselleştirir. Eğitim ve test skorlarının zamanla nasıl değiştiğini analiz ederek, modelin **aşırı uyum** (overfitting) yapıp yapmadığı ve **genelleme kapasitesi** hakkında önemli bilgiler elde edilebilir. Bu tür görselleştirmeler, modelin **performansını iyileştirmek** için gerekli stratejiler hakkında karar vermeye yardımcı olur.

**6) Kelime Gömme (Word Embedding) Görselleştirme ve Yorum**

TensorFlow Embedding Projector aracı ile eğitilmiş GloVe temelli gömme vektörlerinin 2B yansımaları incelenmiştir. Örnek olarak through kelimesi (vektör uzayında 137. indeks) analiz edildiğinde, bu kelimeye en yakın komşular:

* 5
* back
* me

şeklinde gözlemlenmiştir.

metin, ekran görüntüsü, diyagram, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Bu yakın komşuların through kelimesiyle cümle içi kullanım bakımından **sıklıkla birlikte geçtiği**, veya bağlam olarak **yakın** olduğu düşünülmektedir. Özellikle "back" ve "me" kelimeleri, through ile sıklıkla kullanılan ifadelerdir (örn: "through me", "back through").

**Yorum:** Bu durum, GloVe modelinin **kelimeler arasındaki bağlam temelli semantik ilişkileri** başarılı şekilde öğrendiğini göstermektedir. Vektör uzayında birbirine yakın yerleşen kelimeler, genellikle benzer veya ilişkili anlamlar taşımaktadır.

TensorFlow Projector aracı kullanılarak "mean" kelimesinin vektör uzayındaki konumu görselleştirilmiş ve en yakın kelimeler analiz edilmiştir.

**"mean" kelimesine en yakın embedding komşuları:**

* poor
* said
* though

metin, ekran görüntüsü, diyagram içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Bu benzerlikler, modelin "mean" kelimesini hangi bağlamlarda gördüğünü yansıtır. Örneğin:

* "mean" ve "poor" kelimeleri genellikle olumsuz bağlamlarda, eleştiri veya değerlendirme cümlelerinde birlikte yer alabilir.
* "said" ve "though" gibi kelimeler ise anlatı (narrative) yapısında, diyalog veya hikaye anlatımında sık kullanılır. "mean" kelimesi de benzer bağlamlarda sıkça geçebilir.

Bu, GloVe modelinin **istatistiksel bağlam benzerliği** üzerinden kelimeleri nasıl gruplayabildiğini açıkça ortaya koyar.

**3.Örnek :** Bu bölümde "little" kelimesinin embedding uzayındaki konumu analiz edilmiştir. GloVe vektörleri ile oluşturulan embedding matrisi TensorFlow Projector’a aktarılmış ve "little" kelimesine en yakın komşular gözlemlenmiştir.

**"little" kelimesine en yakın embedding komşuları:**

* guys
* out
* find

metin, ekran görüntüsü, diyagram, harita içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Yorum:**

* "little" ve "guys" kelimeleri sıklıkla birlikte kullanılan ifadelerdir (örneğin: *"little guys"*, *"little kids"*) ve bu bağlamda GloVe modelinin bu ikiliyi yakın konumlandırması beklenen bir durumdur.
* "out" kelimesi genellikle yön ya da hareket bildiren cümlelerde "little" ile birlikte kullanılabilir (örneğin: *"the little boy went out"*) ve bu bağlam da modelin semantik ilişkileri yakalayabildiğini gösterir.
* "find" kelimesi de hikaye anlatımında veya olay aktarımında "little" kelimesi ile birlikte kullanılabilir (örneğin: *"find the little cat"*). Bu, bağlamsal benzerliğin bir başka göstergesidir.

Bu analiz, kelimelerin yalnızca bire bir eşanlamlı olmadığını, aynı zamanda **anlamlı bağlamlarda** birlikte kullanılan kelimelerin de vektör uzayında yakınlaştığını ortaya koymaktadır. Bu durum, kelime gömme (embedding) yöntemlerinin semantik anlamı öğrenme gücünü açıkça gösterir.

**Genel Yorum ve Değerlendirme:**

Yapılan çalışmalar sonucunda, kelime gömme (word embedding) yöntemlerinin doğal dil işleme (NLP) uygulamalarında ne kadar etkili olduğu gözlemlenmiştir. GloVe ile oluşturulan vektör temsilleri sayesinde, benzer anlamlara veya bağlamsal ilişkileri olan kelimelerin yüksek boyutlu uzayda birbirine yakın konumlandığı görülmüştür. "through", "mean" ve "little" gibi kelimelere en yakın komşular incelendiğinde, bu kelimelerin semantik olarak benzer ya da aynı bağlamda birlikte geçen kelimelerle gruplaştığı belirlenmiştir.

Bu durum, özellikle duygu analizi gibi metne dayalı sınıflandırma problemlerinde modelin daha anlamlı öğrenme yapabilmesini sağlamaktadır. Klasik yöntem olan Naive Bayes + CountVectorizer ile elde edilen başarı oranı temel bir referans sunarken, önceden eğitilmiş GloVe vektörleri ile oluşturulan derin öğrenme temelli modellerin daha zengin anlam ilişkileri yakalayabildiği görülmüştür. TensorFlow Projector üzerinden yapılan görselleştirme çalışmaları da bu ilişkilerin sezgisel olarak gözlemlenmesine katkı sağlamıştır.

Sonuç olarak, word embedding teknikleri hem model performansını artırmakta hem de metindeki anlam ilişkilerini öğrenmede önemli bir rol oynamaktadır. Bu tür yöntemlerin kullanımı, özellikle karmaşık ve bağlama duyarlı metin analizi problemlerinde vazgeçilmez hale gelmiştir.

**Projenin Githubdaki Adres Yolu:**

<https://github.com/Selami7321/-IMDB-Film-Yorumlar-ile-Word-Embedding-Tabanl-Duygu-Analizi>

**Katkı Dağılımı**:

* Uğur Baki Arslan : Veri Ön İşleme: %30
* Selami Çetin : Model Geliştirme ve Kodlama: %40
* Yunus Emre Sevinç : Raporlama: %30

**7) Kaynakça**

* **Veri Seti**:  
  IMDB Movie Reviews Dataset – Kaggle: <https://www.kaggle.com/datasets/vishakhdapat/imdb-movie-reviews>
* **Word Embedding**:  
  [GloVe: Global Vectors for Word Representation](https://nlp.stanford.edu/projects/glove/)  
  GloVe 6B.100d önceden eğitilmiş vektörleri kullanılmıştır.
* **Yararlanılan Kaynaklar ve Belgeler**:
  + scikit-learn resmi dökümantasyonu – <https://scikit-learn.org>
  + TensorFlow/Keras dökümantasyonu – <https://keras.io>
  + Matplotlib, Seaborn grafik kütüphaneleri – <https://matplotlib.org> / https://seaborn.pydata.org
  + ROC Curve ve Confusion Matrix için çeşitli örnekler: Medium, Stack Overflow makaleleri