Veri Madenciliği ile Sınıflandırma

Proje Raporu

Hazırlayan

Fırat Kaan Bitmez

Öğrenci Numarası

23281855

Dersin Hocası

Prof.Dr. Erdal Kılıç

Giriş

Bu rapor, Veri Madenciliği dersinde verilen Sınıflandırma projesi için hazırlanmıştır. Proje, Türkçe köşe yazarlarının yazılarını sınıflandırmayı amaçlamaktadır. Beş farklı köşe yazarının en az 20 köşe yazısı kullanılarak bir model oluşturulmuş ve bu model, yeni köşe yazılarının hangi yazarlara ait olduğunu tahmin etmek için kullanılmıştır.

Amacımız

Bu projenin temel amacı, metin sınıflandırma algoritmalarını kullanarak Türkçe köşe yazarlarının yazılarını sınıflandırmaktır. Bunun için, veri madenciliği teknikleri ve doğal dil işleme yöntemleri kullanılmıştır. Türkçe köşe yazarlarının yazılarını sınıflandırmak, metinlerin içeriğini analiz etmek ve farklı yazarların üsluplarını belirlemek için önemlidir. Bu tür bir sınıflandırma, haber siteleri veya medya kuruluşları gibi platformlarda içerik yönetimi ve kullanıcı deneyimini iyileştirmek için kullanılabilir.

Kullanılan Kütüphaneler, Araçlar ve Modüller

Proje **Python** programlama dili ile kodlanmıştır. Projede kullanılan kütüphane ve modüller aşağıda listelenmiştir. Her birinin kullanım amacı ve projede ne şekilde kullanıldığına dair kısa açıklamalar eklenmiştir:

collections: Veri yapıları için kullanılmıştır.

os: Dosya işlemleri için kullanılmıştır.

re: Regular expressions için kullanılmıştır. (pip install regex)

numpy: Sayısal işlemler için kullanılmıştır. (pip install numpy)

snowballstemmer: Türkçe kök bulma işlemleri için kullanılmıştır. (pip install snowballstemmer)

sklearn: Model oluşturma, metrikler ve sınıflandırma algoritmaları için kullanılmıştır. (pip install scikit-learn)

seaborn: Görselleştirme için kullanılmıştır.

matplotlib.pyplot: Görselleştirme için kullanılmıştır. (pip install matplotlib)

Kullanılan Algoritma ve Araçlar

Projede kullanılan ana algoritma **Destek Vektör Makineleri (Support Vector Machines - SVM)** algoritmasıdır. SVM, sınıflandırma problemleri için kullanılan bir makine öğrenimi algoritmasıdır. Veri noktalarını sınıflandırmak için bir hiperdüzlem oluşturarak çalışır. SVM, belirli bir hiperdüzlemi seçerken sınıflar arasındaki marjı maksimize etmeye çalışır, bu da genelleme yeteneğini artırır.

Kodda, **sklearn.svm.SVC** sınıfı kullanılarak SVM modeli oluşturulmuştur. Ayrıca GridSearchCV ile en iyi parametrelerin bulunması sağlanmıştır. **GridSearchCV**, bir hiperparametre alanının belirli bir değer aralığında en iyi sonuçları veren parametre değerlerini bulmak için kullanılır. Bu kodda, **SVM** için "**C**" (ceza parametresi), "**gamma**" (RBF çekirdek fonksiyonunun genişliği) ve "**kernel**" (çekirdek fonksiyonu) parametrelerinin en iyi değerlerini bulmak için kullanılmıştır.

Son olarak, en iyi model **GridSearchCV** ile belirlenmiş ve eğitim verisi üzerinde eğitilmiştir. Model performansı, sınıflandırma raporu ve karışıklık matrisi ile değerlendirilmiştir. **Confusion matrisi(Karışıklık Matrisi)**, gerçek ve tahmin edilen sınıflar arasındaki ilişkiyi görselleştiren bir tablodur. Bu değerlendirmeler, modelin doğruluğunu ve performansını değerlendirmek için kullanılmıştır.

Tasarım Adımları

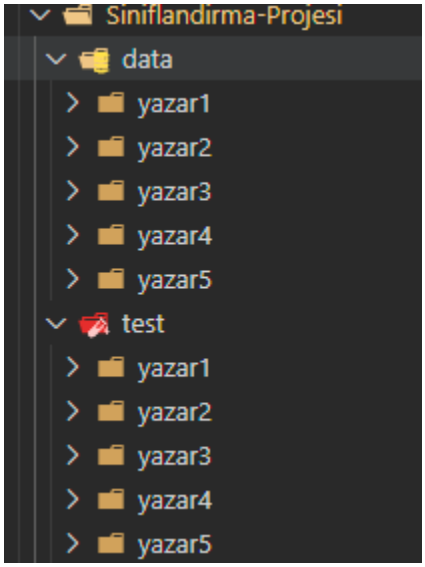
1. Veri Toplama ve Düzenleme

Projedeki Veri Kümesi, beş farklı köşe yazarının 24'er Köşe yazısı alınarak **Toplamda 120 Köşe yazısından** oluşan bir Data oluşturulmuştur. Her bir yazarın Eğitimi için 20, Testi için 4'er köşe yazısı ayrılmıştır. Bu Projede **Eğitim için 100 Köşe yazısı test için ise 20 köşe yazısı** kullanılmıştır.

Eğitim için ayrılan Köşe yazıları Data klasörü içinde yazar1, yazar2, yazar3, yazar4, yazar5 klasörleri içerisinde **20'şer** tane olmak üzere **100** köşe yazısı toplanmış ve düzenlenmiştir.

Test için ayrılan köşe yazıları ise **test** klasörü içerinde yazar isimleri ile toplanmıştır.

Her yazarın köşe yazıları doc(document) kısaltmasıyla adlandırılmış ve 1'den 20 ye kadar numaralandırılarak Veri Setimiz kategorize ve düzenli bir hale getirilmiştir.



Veri Setimizde Eğitim ve Test için kullanacağımız yazarların Kimlikleri şu şekildedir.

Yazar1: **Uğur Dündar**

Yazar2: **Soner Yalçın**

Yazar3: **Murat Muratoğlu**

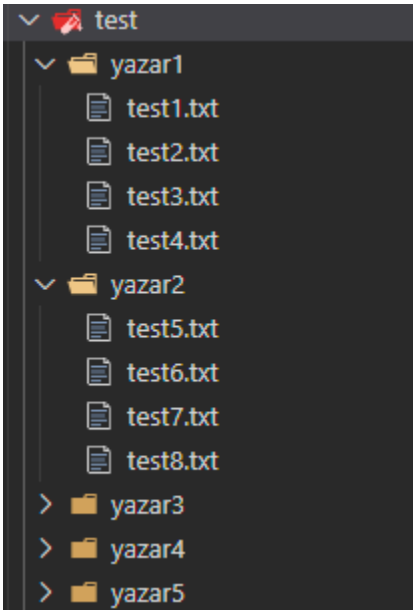
Yazar4: **Ege Cansen**

Yazar5: **Rahmi Turan**

Test için ayrılan 20 köşe yazısı ise yine aynı şekilde 1'den 20'ye kadar olacak şekilde sıralanmış ve **test** klasörü içerisinde toplanmıştır. Ve her yazarın kendine ait klasöre göre ayrılmıştır. Buradaki test isimli text dosyalarını 4'er 4'er olacak şekilde yazar1, yazar2, yazar3, yazar4, yazar5 diye adlandırılmış yazarlara aittir.

Yani test klasöründeki test(sayi).txt dosyalarının:

- **1-4** Arası köşe yazılarının yazarı **Yazar1** (Uğur Dündar),
- **5-8** Arası köşe yazılarının yazarı **Yazar2** (Soner Yalçın),
- **9-12** Arası köşe yazılarının yazarı **Yazar3** (Murat Muratoğlu),
- **13-16** Arası köşe yazılarının yazarı **Yazar4** (Ege Cansen),
- **17-20** Arası köşe yazılarının yazarı **Yazar5** (Rahmin Turandır.



Veri Setimizi oluşturduktan sonra **Önişleme** ve **Temizleme** adımları için bir **Stopword** Türkçe Kelime listesi oluşturdum ve Proje içerisine dahil ettim. Bu Kelime Listesi **Zemberek** Doğal Dil İşleme Kütüphanesi içerisinden alınmıştır.

stopword.txt diye adlandırdığım ve proje içerisine dahil etmek için projede şöyle bir kodlama yapıldı.

```
stopwords_path = "C:/Users/FIRAT/Desktop/myProject/veri-
madenciligi/Siniflandirma-Projesi/stopword.txt"

# Stop word dosyasını okuyarak, metinlerdeki gereksiz kelimeleri filtrelemek için
bir stop word listesi
with open(stopwords_path, "r", encoding="utf-8") as stopwords_file:
    stop_words = stopwords_file.read().splitlines()
```

Veri Toplama ve Düzenlemede Karşılaşılan Zorluklar

- Veri bir metin olduğu ve burada bahsedilen yazarların zamanla duygu ve düşünce ya da eğitim düzeyi değiştiği için yazılar arasındaki Süre çok artarsa model şaşırıp farklı tahminlerde bulunabiliyor
- Alıntı yapma ise başka bir sorun Yazarların kendi yazılarında diğer yazarlardan alıntı yapması Modelin sanki o yazarın yazısıymış gibi algılamasına sebep olabiliyor.
- Yazarların Kendince bir üslup ve açıklama mantıkları var bu da yazıların uzun yada kısa olmasına sebep oluyor. Fakat böyle bir durumda da bir yazar bir konuyu iki sayfada açıklarken diğer bir yazar yarım sayfada açıklayabiliyor. Buda bizim veri setimizde dengesiz bir ortam oluşturuyor. Bu yüzden bende veri setini dengelemek için şöyle bir yöntem buldum.

```
# Eğitim verilerini dengeli hale getirme
# Her yazardan eşit sayıda örnek alın
min_samples = min(len(texts) for texts in training_data.values())
for author, texts in training_data.items():
    training_data[author] = texts[:min_samples]
```

Bunu yapmamım sebebi yazarlara ait vektörler oluşturduğımda bir yazarın büyük yada küçük vektörü olması modelin tahmin etmesinde zorluklar ortaya çıkarabiliyordu. Fazla Eğitilen bir model her şeyi x yazarı gibi tahmin ederken az eğitilen bir yazarda ise hiç tahmin edilemiyordu.

2. Veri Temizleme ve Önişleme

Veri temizleme ve ön işleme adımı, metin verilerinin analiz ve modelleme sürecine hazırlanmasını sağlayan önemli bir aşamadır. Bu adımda, veri setindeki metinlerin anlamlı bilgilerini çıkarmak ve gereksiz bilgilerden arındırmak için çeşitli işlemler uygulanır. Normalde bu tarz işlemler için hazır kütüphaneler kullanılabilir (Zemberek gibi.) Fakat yaptığım denemelerde kütüphaneler belirli bir düzende ve linux platformda düzgün çalışıyordu bende kendime göre bir önişleme ve temizleme aşaması yarattım.

İşte bu adımların detaylıca açıklaması:

- **Özel Karakterlerin Kaldırılması**

Metin verilerinde genellikle noktalama işaretleri, parantezler, tireler gibi özel karakterler bulunur. Bu karakterler metin analizinde genellikle gereksizdir ve model performansını olumsuz etkileyebilir. Bu nedenle, özel karakterler genellikle kaldırılır.

```
cleaned_text = re.sub(r'\W', ' ', text)   # Noktalama işaretlerinin kaldırılması
```

- **Sayıların Kaldırılması**

Metin verilerinde sayılar bulunabilir, ancak metin sınıflandırma için genellikle anlamsızdır. Bu nedenle, metin verilerinden sayılar genellikle kaldırılır.

```
cleaned_text = re.sub(r'\d+', ' ', cleaned_text)  # Sayıların kaldırılması
```

- **Küçük Harfe Dönüştürme**

Metin verilerinin tümünü küçük harflere dönüştürmek, büyük ve küçük harf farklılıklarını ortadan kaldırarak modele daha tutarlı bir şekilde erişmesini sağlar. Örneğin, "Kitap" ve "kitap" kelimeleri aynı kelime olarak kabul edilir.

```
cleaned_text = cleaned_text.lower()   # Metnin küçük harflere dönüştürülmesi
```

- **Stop Words'lerin Kaldırılması**

Stop words, metinlerde sıklıkla görülen ancak genellikle anlam taşımayan kelimelerdir (örneğin, "ve", "ama", "veya" gibi). Bu kelimeler genellikle modelin performansını düşürür ve bu nedenle genellikle kaldırılır.

```
cleaned_text = ' '.join([word for word in cleaned_text.split() if word not in
stop_words])  # Stop words'lerin kaldırılması
```

- **Türkçe Kök Bulma İşlemi**

Türkçe kök bulma işlemi, kelimelerin köklerini bulmayı amaçlar. Örneğin, "geliyorum", "geliyorsun", "geliyorlar" kelimelerinin kökü "gel"dir. Bu işlem, modelin daha genelleştirilmiş ve anlamlı özelliklerle çalışmasını sağlar.

```
  # Türkçe kök bulma işlemi

stemmer = TurkishStemmer()
cleaned_text = ' '.join([stemmer.stemWord(word) for word in
cleaned_text.split()])
```

Bu adımların uygulanması, metin verilerinin temizlenmesini ve modele hazır hale getirilmesini sağlar. Bu sayede, modelin daha iyi performans göstermesi ve doğru sonuçlar üretmesi sağlanır.

```
# Eğitim verisine temizlenmiş metni ekleyin
training_data[author_name].append((cleaned_text, file_name))  # Dosya ismiyle birlikte metni ekliyoruz
```

Veri Önişleme ve Temizleme için Türkçe Doğal Dil İşlemenin bir kütüphanesi olan **Zemberek** kullanmak istedim. Fakat Java ile yazıldığı için ve Windows Platformlarda çok fazla sorun ile karşılaştığım için Kendimce yöntemler ile bu adımı tamamladım.

3. TF-IDF Vektörleme

TF-IDF (Term Frequency-Inverse Document Frequency), metin verilerini sayısal vektörlere dönüştürmek için yaygın olarak kullanılan bir tekniktir. Bu teknik, metin verilerindeki her bir kelimenin önem derecesini belirlemek için kullanılır.

```
# Model oluşturma
model = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('classifier', SVC())
])
```

Burada, Pipeline kullanılarak TfidfVectorizer() sınıfı TF-IDF vektörleme işlemi için kullanılmıştır. Bu şekilde, tfidf adı altında vektörleme işlemi, diğer sınıflandırma adımı olan classifier adı altında ise SVC (Destek Vektör Makineleri) sınıflandırma algoritması ile bir boru hattı (pipeline) oluşturulmuştur.

İlk olarak, TF (Term Frequency) ve IDF (Inverse Document Frequency) kavramlarını ayrı ayrı açıklayarak başlayalım:

- **Term Frequency (TF - Kelime Sıklığı)**

Bir belgedeki bir kelimenin ne kadar sıklıkta geçtiğini gösteren bir metrik. Genellikle, bir belgedeki bir kelimenin sıklığı, o kelimenin belgedeki toplam kelime sayısına oranı olarak hesaplanır. Ancak bazen kelime frekansı doğrudan kelimenin belgedeki toplam sayısını da temsil edebilir. Örneğin, "kitap" kelimesinin bir belgedeki sıklığı, o belgede "kitap" kelimesinin geçtiği toplam sayıdır.

- **Inverse Document Frequency (IDF - Ters Belge Frekansı)**

Bir kelimenin belgedeki nadirliğini ölçen bir metrik. IDF, bir kelimenin ne kadar nadir olduğunu belirler. Nadir kelimeler, belgedeki genel içeriği daha iyi temsil edebilir ve bu nedenle daha yüksek bir öneme sahip olabilirler. IDF, bir kelimenin belgedeki tüm belgelerde ne kadar nadir olduğunu hesaplar ve bu nadirlik derecesine göre bir ağırlık verir. Nadir kelimelerin IDF değerleri daha yüksektir.

TF-IDF vektörleme işlemi, TF ve IDF değerlerinin çarpımıyla elde edilir ve bu işlem her bir kelimenin her bir belge için bir özellik vektörü oluşturur. TF-IDF vektörleme adımları şu şekilde gerçekleşir:

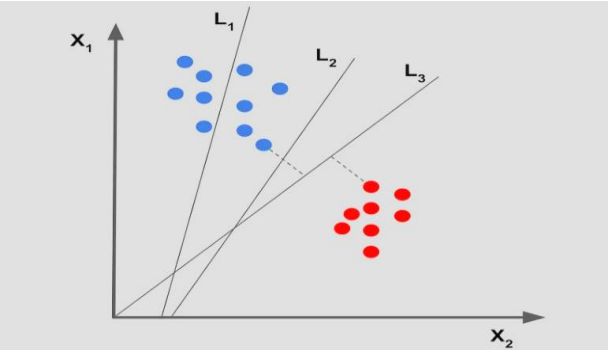
- **TF-IDF Vektörleme**

TF ve IDF değerleri, her bir kelimenin her bir belgedeki önemini belirlemek için çarpılır. Böylece, her bir belge için bir TF-IDF vektörü oluşturulur.

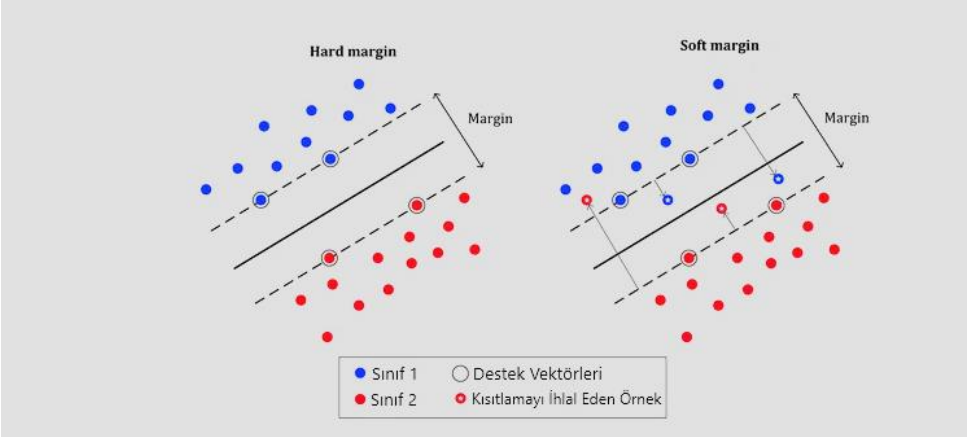
Bu işlem sonucunda, her bir belge TF-IDF vektörlerinin oluşturulmasıyla sayısal bir temsile dönüştürülür. Bu sayısal temsil, metin verilerinin modellemeye uygun hale getirilmesini sağlar ve metin sınıflandırma gibi görevler için kullanılabilir. TF-IDF vektörleme, metin verilerindeki kelime önemini vurgulayarak modelin daha iyi performans göstermesini sağlar.

4. Destek Vektör Makineleri (SVM) Algoritması ile SVC (Support Vector Classifier) Sınıflandırma

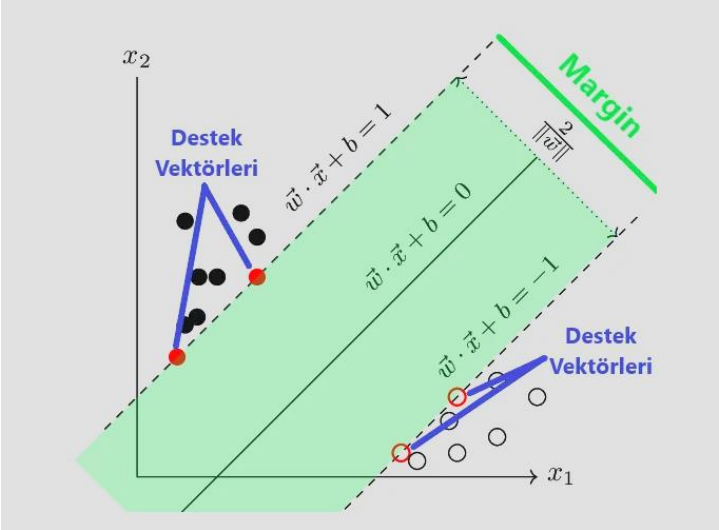
Support Vector Machine (Destek Vektör Makineleri diğer adıyla DVM ya da SVM) sınıflandırma, regresyon ve aykırı değerleri bulmak için kullanılan denetimli (superwised) bir öğrenme tekniğidir. SVM algoritması classification kavramı adı altında gelişen ve diğer classification türlerinden farklı olan bir algoritmadır. SVM algoritması bir düzlem üzerine yerleştirilmiş 2 veya daha fazla nokta kümelerini ayırmak için doğrular çizer. 2 veri kümesi düşünüldüğünde bu doğrunun, iki kümenin noktaları için de maksimum uzaklıkta olmasını amaçlar. Karmaşık ama küçük ve orta ölçekteki veri setleri için uygundur.



Görüldüğü üzere iki veri kümesi grafik üzerinde görülmekte ve 3 doğrusal çizgi. Buradaki öncelikli amacımız veri kümelerinin ayıran bir doğru çizmek. Görüldüğü üzere L_1 doğrusu bu iki kümeyi ayırmakta en başarısız olanı. Diğer iki doğruya bakacak olursak L_2 mavi renkli noktaların oluşturduğu kümeye daha yakınken kırmızı veri kümesine daha uzak kalır. Buradaki dengesizlik yeni tahmin edilmesi istenen verinin daha hatalı bir tahminle sonuçlanmasına neden olacaktır. Bundan dolayı algoritma 2 veri kümesindeki birbirlerine en yakın olan verilerin (kırmızı ve mavi verilerin) arasındaki en fazla aralığı yakaladığı doğruyu çizmeyi tercih edecektir. Bu arada sınıflar arası çizilebilecek sonsuz adet doğru vardır ve bu çizilebilecek doğrulara karar doğruları adı verilir.



Bu iki sınıfı en iyi ayıran doğrunun (L_3) bölgeler arasında kalan alana margin denir. Margin ne kadar geniş olursa sınıflar okadar iyi ayrıştırılır. Bazı durumlarda veirler margin bölgesine girebilir. Bu duruma 'Soft Margin' denir. Hard Margin ise veri doğrusal olarak ayrılabiliyorsa çalışır.



Burada görüldüğü gibi her iki veri kümesine destek vektörleri çizilmiştir. Bu destek vektörleri referans alınarak SVM çizilebilecek en iyi doğruyu çizer (w.x+b=0). SVM lerde sınıflar +1 veya -1 olarak etiketlenir. Bundan dolayı karar doğrusu (hiper düzlem) üstünde kalan doğruya wx+b=1 altında kalan doğruya ise wx+b=-1 olarak yazılır.

Kısaca Özetlemek gerekirse SVM, diğer yöntemlere göre iki temel avantaja sahiptir. Yüksek boyutlu uzaylarda işlem yapabilme yeteneğine sahiptir ve bu, genellikle birçok özelliğin kullanılması durumunda yararlıdır. Ayrıca, aşırı uymaya karşı dayanıklıdır, yani genelleme yeteneği daha iyidir.

Bu kodlamada Destek Vektör Makineleri (SVC) sınıflandırması için aşağıdaki kodlar bulunmaktadır:

- **SVM Algoritmasıyla SVC'nin oluşturulması**

```
model = Pipeline([
('tfidf', TfidfVectorizer()),
 ('classifier', SVC())
])
```

- **En iyi parametrelerin bulunması için GridSearchCV kullanımı ve en iyi modelin seçilmesi**

```
# GridSearchCV ile en iyi parametreleri bulma
param_grid = {
    'classifier__C': [0.1, 1, 10, 100],
    'classifier__gamma': [1, 0.1, 0.01, 0.001],
    'classifier__kernel': ['rbf', 'linear', 'poly', 'sigmoid']
}

grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X, y)

best_model = grid_search.best_estimator_

# Eğitim ve doğrulama verisi için modeli eğitin
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)
best_model.fit(X_train, y_train)

# Model performansını değerlendirme
y_pred = best_model.predict(X_val)
print("Sınıflandırma Raporu:")
print(classification_report(y_val, y_pred))
```

5.Modelin Eğitilmesi ve Değerlendirilmesi

Modelin eğitilmesi ve değerlendirilmesi, makine öğrenimi projelerinde kritik bir aşamadır. Bu aşama, modelin performansının anlaşılmasına ve geliştirilmesine olanak sağlar. İşte bu adımların detaylı bir açıklaması:

- **Eğitim Aşaması**

Model eğitimi, öncelikle belirlenen algoritmaya ve parametrelere göre gerçekleştirilir. Eğitim veri seti kullanılarak model oluşturulur ve bu model, veri setindeki örüntüleri öğrenir.

Destek Vektör Makineleri (SVM) gibi algoritmalar için, eğitim aşamasında hiperdüzlem belirlenir ve modelin parametreleri ayarlanır.

```
# Eğitim ve doğrulama verisi için modeli eğitin
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)
best_model.fit(X_train, y_train)
```

- **Doğrulama Aşaması**

Model oluşturulduktan sonra, genellikle ayrılmış olan doğrulama veri seti kullanılarak modelin performansı değerlendirilir. Bu aşamada, modelin ne kadar iyi genelleştirildiği ve yeni veri noktaları üzerinde ne kadar başarılı olduğu ölçülür.

Doğrulama veri seti üzerinde modelin performansını değerlendirmek için çeşitli metrikler kullanılabilir. Bunlar arasında doğruluk, hassasiyet, geri çağırma ve F1-score gibi değerler bulunur.

```
# Model performansını değerlendirme
y_pred = best_model.predict(X_val)
print("Sınıflandırma Raporu:")
print(classification_report(y_val, y_pred))
```

- **Test Aşaması**

Modelin eğitim ve doğrulama aşamalarını tamamladıktan sonra, genellikle ayrılmış olan test veri seti kullanılarak modelin performansı kesin olarak değerlendirilir.

Test veri seti, genellikle modelin daha önce görmediği verilerden oluşur ve modelin gerçek dünya performansını yansıtır.

Modelin test veri seti üzerindeki performansı, eğitim ve doğrulama aşamalarında elde edilen sonuçlarla karşılaştırılarak değerlendirilir.

```
# Test verisindeki her bir yazar için
for author_folder in os.listdir(test_data_folder):
    if os.path.isdir(os.path.join(test_data_folder, author_folder)):
        author_name = author_folder
        # Yazar klasöründeki her bir dosya için
        for file_name in os.listdir(os.path.join(test_data_folder,
author_folder)):
            file_path = os.path.join(test_data_folder, author_folder, file_name)
            with open(file_path, "r", encoding="utf-8") as file:
                text = file.read()
                # Metni temizleme ve kök bulma işlemleri
                cleaned_text = re.sub(r'\W', ' ', text)
                cleaned_text = re.sub(r'\d+', ' ', cleaned_text)
```

```
                cleaned_text = cleaned_text.lower()
                cleaned_text = ' '.join([word for word in cleaned_text.split() if
word not in stop_words])
                stemmer = TurkishStemmer()
                cleaned_text = ' '.join([stemmer.stemWord(word) for word in
cleaned_text.split()])
                # Gerçek yazarı kaydet
                true_values.append(author_name)
                # Model ile tahmin yapma
                prediction = best_model.predict([cleaned_text])[0]
                predicted_values.append(prediction)
                # Dosya ismi ve tahmin edilen yazarı terminalde gösterme
                print("Dosya:", file_name, "- Gerçek Yazar:", author_name, "-
Tahmin Edilen Yazar:", prediction)
```

- **Performans Değerlendirme**

Modelin performansı, doğrulama ve test aşamalarında elde edilen sonuçlar kullanılarak değerlendirilir.

Doğruluk, hassasiyet, geri çağırma, F1-score gibi metriklerin yanı sıra ROC eğrisi, confusion matrix gibi görsel araçlar da kullanılarak modelin performansı analiz edilir.

Bu analiz sonucunda, modelin hangi sınıflarda daha iyi veya daha kötü performans gösterdiği belirlenir ve gerektiğinde modelin iyileştirilmesi için önlemler alınır.

Modelin eğitilmesi ve değerlendirilmesi aşamaları, makine öğrenimi projelerinde modelin başarısını belirlemek için kritik öneme sahiptir. Bu aşamaların titizlikle yürütülmesi, projenin başarısını ve modelin gerçek dünya performansını doğrudan etkiler.

```
# Confusion matrix hesaplama
conf_matrix = confusion_matrix(true_values, predicted_values)

# Confusion matrixi görselleştirme
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d")
plt.xlabel('Tahmin Edilen')
plt.ylabel('Gerçek Değer')
plt.title('5x5 Confusion Matrix')
plt.show()

# Confusion matrixin terminalden çıktı olarak verilmesi
print("5x5 Confusion Matrix:")
```

```
print(conf_matrix)
```

Yaptığımız Test Aşamasından sonra aşağıdaki şekilde bir Confusion matrisi elde ediyoruz. Bunu Yorumlamamız gerekirse:

Normalde Her yazara ait 4'er köşe yazımız vardı yani bizim **mükemmel sonuç** diyebileceğimiz sonuç şöyle bir şey olmadıydı:

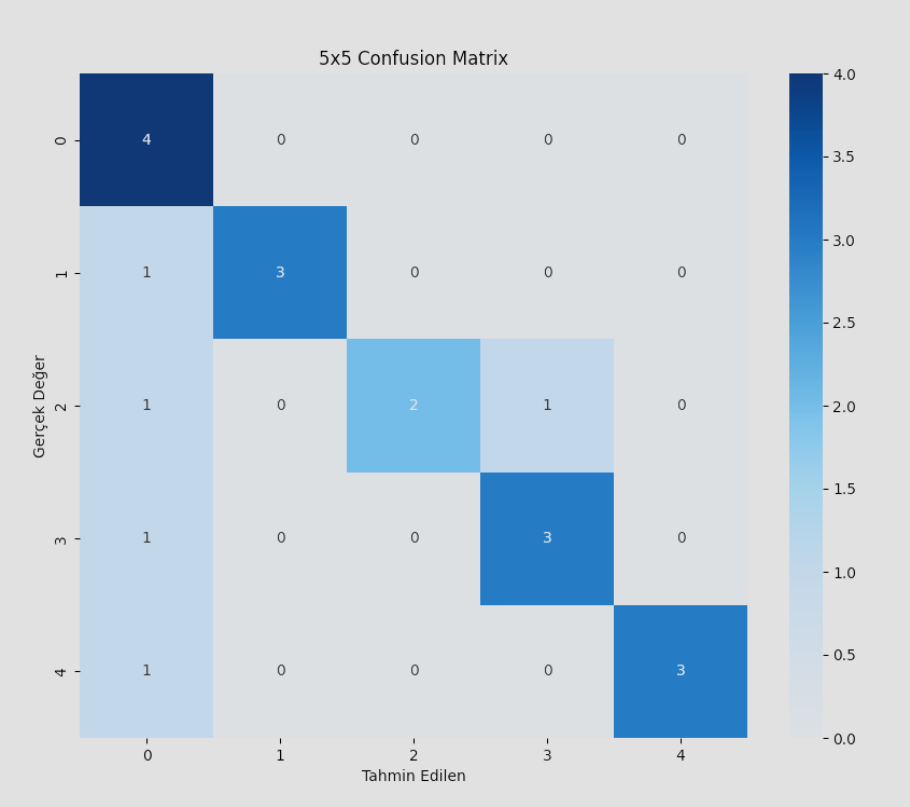
[[4 0 0 0 0]

[0 4 0 0 0]

[0 0 4 0 0]

[0 0 0 4 0]

[0 0 0 0 4]]



Fakat burada görüldüğü üzere modelimiz %75'lik bir doğru tahmin yaparak 20 köşe yazısın 15'in yazarını doğru tahmin etmiştir.

Burada **Yazar1'e** dikkatlice bakarsak bütün yazarların bir yazısını Modelimiz Yazar1 olarak tahmin etmiş bunun aslında bir nedeni var bunlar yukarda Veri oluşturma bölümünde

bahsettiğim sorundan kaynaklanıyor. Çoğu yazar kendi yazısında başka yazarlardan alıntı yaparak yazı yazıyor bu da tahminde yanılma gibi sorunlara yol açıyor. Manuel olarak bu sorunu çok basit bir şekilde çözebilirdik aslında fakat hem Veriyi elle bozmamak adına hem de Model eğitirken Manuel müdahale etmek istemediğim için Sonuca müdahale etmedim.

```
Dosya: test1.txt - Gerçek Yazar: yazar1 - Tahmin Edilen Yazar: yazar1
Dosya: test2.txt - Gerçek Yazar: yazar1 - Tahmin Edilen Yazar: yazar1
Dosya: test3.txt - Gerçek Yazar: yazar1 - Tahmin Edilen Yazar: yazar1
Dosya: test4.txt - Gerçek Yazar: yazar1 - Tahmin Edilen Yazar: yazar1
Dosya: test5.txt - Gerçek Yazar: yazar2 - Tahmin Edilen Yazar: yazar1
Dosya: test6.txt - Gerçek Yazar: yazar2 - Tahmin Edilen Yazar: yazar2
Dosya: test7.txt - Gerçek Yazar: yazar2 - Tahmin Edilen Yazar: yazar2
Dosya: test8.txt - Gerçek Yazar: yazar2 - Tahmin Edilen Yazar: yazar2
Dosya: test10.txt - Gerçek Yazar: yazar3 - Tahmin Edilen Yazar: yazar3
Dosya: test11.txt - Gerçek Yazar: yazar3 - Tahmin Edilen Yazar: yazar4
Dosya: test12.txt - Gerçek Yazar: yazar3 - Tahmin Edilen Yazar: yazar1
Dosya: test9.txt - Gerçek Yazar: yazar3 - Tahmin Edilen Yazar: yazar3
Dosya: test13.txt - Gerçek Yazar: yazar4 - Tahmin Edilen Yazar: yazar1
Dosya: test14.txt - Gerçek Yazar: yazar4 - Tahmin Edilen Yazar: yazar4
Dosya: test15.txt - Gerçek Yazar: yazar4 - Tahmin Edilen Yazar: yazar4
Dosya: test16.txt - Gerçek Yazar: yazar4 - Tahmin Edilen Yazar: yazar4
Dosya: test17.txt - Gerçek Yazar: yazar5 - Tahmin Edilen Yazar: yazar1
Dosya: test18.txt - Gerçek Yazar: yazar5 - Tahmin Edilen Yazar: yazar5
Dosya: test19.txt - Gerçek Yazar: yazar5 - Tahmin Edilen Yazar: yazar5
Dosya: test20.txt - Gerçek Yazar: yazar5 - Tahmin Edilen Yazar: yazar5
```

Zaten Terminal çıktısındada hangi yazısının doğru tahmin edilip hangisinin yanlış tahmin edildiğini gözlemleyebiliyoruz.

```
Sınıflandırma Raporu:
              precision    recall  f1-score   support

      yazar1       1.00      0.60      0.75         5
      yazar2       1.00      0.83      0.91         6
      yazar3       0.75      1.00      0.86         3
      yazar4       1.00      0.75      0.86         4
      yazar5       0.40      1.00      0.57         2

    accuracy                           0.80        20
   macro avg       0.83      0.84      0.79        20
weighted avg       0.90      0.80      0.82        20
```

Terminal çıktısında sınıflandırma modelinin performansını değerlendirmek için bir çıktı alıyoruz. Bu çıktıda, her bir sınıf için "precision", "recall" ve "f1-score" olmak üzere üç ana ölçümü içerir. Ayrıca, modelin genel performansını görmek için "accuracy" (doğruluk) ölçümü de sağlanmıştır.

Rapordaki temel terimlerin açıklamaları

Precision (Kesinlik): Bir sınıfa ait tahmin edilen örneklerin ne kadarının gerçekten o sınıfa ait olduğunu gösterir. Kesinlik, yanlış pozitiflerin (yanlış alarm) oranını ölçer. Yüksek kesinlik

değeri, modelin o sınıfı doğru bir şekilde tanımlama yeteneğini gösterir. Formülü: TP / (TP + FP), TP = True Positive (Gerçek pozitif), FP = False Positive (Yanlış pozitif).

Recall (Duyarlılık): Bir sınıfa ait tüm gerçek örneklerin ne kadarının doğru bir şekilde tahmin edildiğini gösterir. Recall, yanlış negatiflerin (kaçırılanlar) oranını ölçer. Yüksek recall değeri, modelin o sınıfı kaçırmadan ne kadar iyi tanıdığını gösterir. Formülü: TP / (TP + FN), FN = False Negative (Yanlış negatif).

F1-score: Precision ve recall'in harmonik ortalamasıdır. F1-score, hem kesinlik hem de duyarlılığın bir dengeyi temsil ettiği bir ölçüdür. Dengesiz veri kümelerinde, sınıflandırma performansını daha doğru bir şekilde değerlendirmek için kullanılır.

Support (Destek): Her bir sınıfa ait gerçek örneklerin sayısını temsil eder.

Accuracy (Doğruluk): Modelin doğru tahmin ettiği toplam örneklerin oranını gösterir. Tüm sınıfların doğru tahmin edilme oranını gösterir. Ancak, dengesiz veri kümelerinde yanıltıcı olabilir.

Macro Avg (Makro Ortalama): Her sınıf için ölçülen metriklerin aritmetik ortalamasıdır. Tüm sınıflara eşit ağırlık verir.

Weighted Avg (Ağırlıklı Ortalama): Her sınıfın destek (support) değerine göre ağırlıklandırılmış metriklerin ortalamasıdır. Bu, dengesiz veri kümelerinde daha güvenilir bir performans ölçüsü sağlar.

***Bu rapora göre**, modelin "yazar1" ve "yazar5" sınıflarını tanıma yeteneği diğerlerine göre daha düşüktür. "Yazar5" sınıfında özellikle düşük precision dikkat çekmektedir, yani model "yazar5" olarak etiketlediği örneklerin büyük bir kısmı yanlış alarm olabilir. Öte yandan, "yazar2" ve "yazar3" sınıflarını tanıma yeteneği oldukça yüksektir. Ancak, **tüm sınıfların doğruluk oranı %80**'dir, bu da modelin **genel olarak iyi performans** gösterdiğini gösterir.

Sonuç

Eğitim verileri temizlenmiş ve kök bulma işlemleri uygulanmıştır. Model, GridSearchCV ile en iyi parametrelerin bulunmasıyla oluşturulmuştur. Son olarak, model test verisi üzerinde değerlendirilmiştir ve başarı metrikleri kullanılarak performansı değerlendirilmiştir.

Projede elde edilen sonuçlar oldukça başarılıdır. Model, test verisi üzerinde %80 civarında bir doğruluk elde etmiştir. Confusion matrisi ve sınıflandırma raporu ile modelin performansı detaylı bir şekilde incelenmiştir. Proje, Türkçe köşe yazarlarının metinlerini sınıflandırmak için etkili bir çözüm sunmuştur.

Kaynaklar

https://github.com/firatkaanbitmez/veri-madenciligi

https://mfatihto.medium.com/support-vector-machine-algoritması-makine-öğrenmesi-8020176898d8