Bu eğitimde, **veri sızıntısının (data leakage)** ne olduğunu ve nasıl önleneceğini öğreneceksiniz. Nasıl önleyeceğinizi bilmezseniz, veri sızıntısı sıklıkla karşınıza çıkacak ve modellerinizi sinsi ve tehlikeli şekillerde mahvedecektir. Bu nedenle, bu konu pratik yapan veri bilimcileri için en önemli kavramlardan biridir.

Introduction

**Veri sızıntısı (veya sızıntı)**, eğitim verinizin hedef (target) hakkında bilgi içermesi, ancak model tahmin yapmak için kullanıldığında benzer verilerin mevcut olmaması durumunda meydana gelir. Bu durum, eğitim setinde (ve muhtemelen doğrulama verilerinde bile) yüksek performansa yol açar, ancak model gerçek kullanımda (production) kötü performans gösterir.

Başka bir deyişle, sızıntı, bir modelin onunla kararlar almaya başlayana kadar doğru görünmesine, ancak daha sonra çok hatalı hale gelmesine neden olur.

İki ana sızıntı türü vardır: **hedef sızıntısı (target leakage)** ve **eğitim-test kirliliği (train-test contamination)**.

### **Target leakage**

**Hedef sızıntısı (Target leakage)**, tahmincilerinizin, tahminleri yaptığınız sırada mevcut olmayacak verileri içermesiyle meydana gelir. Hedef sızıntısını, bir özelliğin iyi tahminler yapmaya yardımcı olup olmadığına göre değil, verilerin ne zaman veya **kronolojik olarak hangi sırayla** erişilebilir hale geldiğine göre düşünmek önemlidir.

Bir örnek bu konuyu aydınlatacaktır. Pnömoniye yakalanacak kişiyi tahmin etmek istediğinizi düşünün. Ham verilerinizin ilk birkaç satırı şöyle görünüyor:

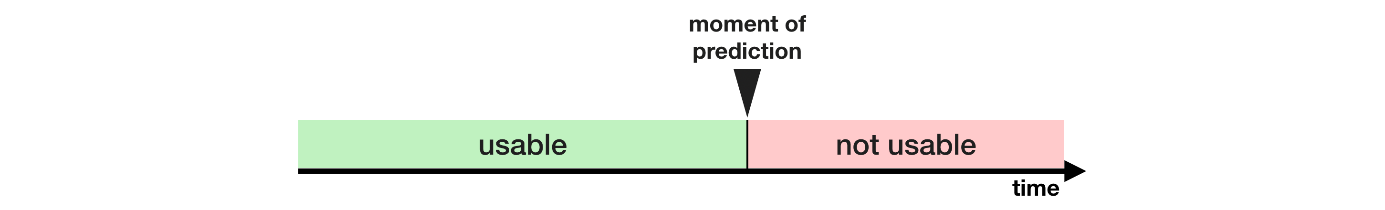
| got\_pneumonia | age | weight | male | took\_antibiotic\_medicine | ... |
| --- | --- | --- | --- | --- | --- |
| False | 65 | 100 | False | False | ... |
| False | 72 | 130 | True | False | ... |
| True | 58 | 100 | False | True | ... |

İnsanlar zatürreye yakalandıktan **sonra** iyileşmek için antibiyotik ilaçlar alırlar. Ham veriler, bu sütunlar arasında güçlü bir ilişki gösterir, ancak took\_antibiotic\_medicine (antibiyotik ilaç aldı) değeri, got\_pneumonia (zatürre oldu) değerinin belirlenmesinden **sonra** sıklıkla değişir. Bu bir hedef sızıntısıdır.

Model, took\_antibiotic\_medicine değeri False olan hiç kimsenin zatürre olmadığını görecektir. Doğrulama verileri, eğitim verileriyle aynı kaynaktan geldiği için, bu örüntü doğrulamada da tekrarlanacak ve model çok iyi doğrulama (veya çapraz doğrulama) skorlarına sahip olacaktır.

Ancak, model daha sonra gerçek dünyada kullanıldığında çok hatalı olacaktır, çünkü zatürreye yakalanacak hastalar bile, gelecekteki sağlık durumları hakkında tahmin yapmamız gerektiğinde henüz antibiyotik almamış olacaklardır.

Bu tür bir veri sızıntısını önlemek için, hedef değeri gerçekleştiğinden **sonra** güncellenen (veya oluşturulan) herhangi bir değişken dışlanmalıdır.



### **Train-Test Contamination**

Doğru. Veri sızıntısının farklı bir türü, eğitim verilerini doğrulama verilerinden ayırmaya dikkat etmediğinizde ortaya çıkar.

Doğrulama, modelin daha önce görmediği veriler üzerinde nasıl performans gösterdiğini ölçmek için tasarlanmıştır. Doğrulama verileri, ön işleme davranışını etkilerse bu süreci fark edilmeyen şekillerde bozabilirsiniz. Bu duruma bazen **eğitim-test kirliliği (train-test contamination)** denir.

Örneğin, train\_test\_split()'i çağırmadan önce (eksik değerler için bir atama aracı [imputer] eğitmek gibi) bir ön işleme çalıştırdığınızı düşünün. Sonuç ne olur? Modeliniz iyi doğrulama skorları alarak size büyük bir güven verebilir, ancak onu karar vermek için kullandığınızda performansı düşük kalır.

Sonuçta, tahminlerinizi yapma şeklinize doğrulama veya test verilerinden gelen bilgileri dahil ettiniz, bu nedenle model, yeni verilere genelleme yapamasa bile bu belirli veriler üzerinde iyi sonuç verebilir. Bu sorun, daha karmaşık özellik mühendisliği (feature engineering) yaptığınızda daha da sinsi (ve daha tehlikeli) hale gelir.

Doğrulamanız basit bir eğitim-test ayrımına dayanıyorsa, doğrulama verilerini, ön işleme adımlarının eğitimini de dahil olmak üzere, her türlü **eğitimden** dışarıda bırakın. Scikit-learn işlem hatları (pipelines) kullanırsanız bu daha kolaydır. Çapraz doğrulama kullanırken ise ön işlemenizi işlem hattının içine yapmanız daha da kritiktir!

# Example

Bu örnekte, hedef sızıntısını (target leakage) tespit etmenin ve gidermenin bir yolunu öğreneceksiniz.

Kredi kartı başvurularıyla ilgili bir veri seti kullanacağız ve temel veri seti kurulum kodunu atlayacağız. Sonuç olarak, her bir kredi kartı başvurusuyla ilgili bilgiler bir DataFrame olan **X**'te saklanacak. Bu veriyi, hangi başvuruların kabul edildiğini bir Series olan **y**'de tahmin etmek için kullanacağız.

import pandas as pd

*# Read the data*

data = pd.read\_csv('../input/aer-credit-card-data/AER\_credit\_card\_data.csv',

true\_values = ['yes'], false\_values = ['no'])

*# Select target*

y = data.card

*# Select predictors*

X = data.drop(['card'], axis=1)

print("Number of rows in the dataset:", X.shape[0])

X.head()

Number of rows in the dataset: 1319

|  | reports | age | income | share | expenditure | owner | selfemp | dependents | months | majorcards | active |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 37.66667 | 4.5200 | 0.033270 | 124.983300 | True | False | 3 | 54 | 1 | 12 |
| 1 | 0 | 33.25000 | 2.4200 | 0.005217 | 9.854167 | False | False | 3 | 34 | 1 | 13 |
| 2 | 0 | 33.66667 | 4.5000 | 0.004156 | 15.000000 | True | False | 4 | 58 | 1 | 5 |
| 3 | 0 | 30.50000 | 2.5400 | 0.065214 | 137.869200 | False | False | 0 | 25 | 1 | 7 |
| 4 | 0 | 32.16667 | 9.7867 | 0.067051 | 546.503300 | True | False | 2 | 64 | 1 | 5 |

Bu küçük bir veri seti olduğundan, model kalitesinin doğru ölçümlerini sağlamak için çapraz doğrulama kullanacağız.

from sklearn.pipeline import make\_pipeline

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import cross\_val\_score

*# Since there is no preprocessing, we don't need a pipeline (used anyway as best practice!)*

my\_pipeline = make\_pipeline(RandomForestClassifier(n\_estimators=100))

cv\_scores = cross\_val\_score(my\_pipeline, X, y,

cv=5,

scoring='accuracy')

print("Cross-validation accuracy: **%f**" % cv\_scores.mean())

Cross-validation accuracy: 0.981052

Deneyim kazandıkça, %98 oranında doğru sonuç veren modeller bulmanın çok nadir olduğunu göreceksiniz. Bu olur, ancak hedef sızıntısı olup olmadığını anlamak için verileri daha yakından incelememiz gerektiği kadar nadirdir.

İşte veri sekmesi altında da bulabileceğiniz verilerin bir özeti:

* **card**: 1 if credit card application accepted, 0 if not
* **reports**: Number of major derogatory reports
* **age**: Age n years plus twelfths of a year
* **income**: Yearly income (divided by 10,000)
* **share**: Ratio of monthly credit card expenditure to yearly income
* **expenditure**: Average monthly credit card expenditure
* **owner**: 1 if owns home, 0 if rents
* **selfempl**: 1 if self-employed, 0 if not
* **dependents**: 1 + number of dependents
* **months**: Months living at current address
* **majorcards**: Number of major credit cards held
* **active**: Number of active credit accounts

Şüpheli görünen birkaç değişken var. Örneğin, **expenditure** (harcama) bu karttaki harcamayı mı, yoksa başvuru öncesi kullanılan kartlardaki harcamayı mı ifade ediyor?

Bu noktada, temel veri karşılaştırmaları çok faydalı olabilir:

expenditures\_cardholders = X.expenditure[y]

expenditures\_noncardholders = X.expenditure[~y]

print('Fraction of those who did not receive a card and had no expenditures: **%.2f**' \

%((expenditures\_noncardholders == 0).mean()))

print('Fraction of those who received a card and had no expenditures: **%.2f**' \

%(( expenditures\_cardholders == 0).mean()))

Fraction of those who did not receive a card and had no expenditures: 1.00

Fraction of those who received a card and had no expenditures: 0.02

Yukarıda görüldüğü gibi, kart almayan herkesin harcaması yokken, kart alanların sadece %2'sinin harcaması yoktu. Modelimizin yüksek bir doğruluğa sahip görünmesi şaşırtıcı değil. Ancak bu, harcamaların muhtemelen **başvurdukları karttaki harcamaları** ifade ettiği bir hedef sızıntısı durumu gibi görünüyor.

**share** (pay) kısmen **expenditure** (harcama) tarafından belirlendiği için, o da dışlanmalıdır. **active** (aktif) ve **majorcards** (ana kartlar) değişkenleri biraz daha az açık, ancak açıklamalardan endişe verici duruyorlar. Çoğu durumda, daha fazlasını öğrenmek için verileri oluşturan kişileri bulamıyorsanız, tedbiri elden bırakmamak daha iyidir.

Hedef sızıntısı olmayan bir modeli şu şekilde çalıştırırdık:

*# Drop leaky predictors from dataset*

potential\_leaks = ['expenditure', 'share', 'active', 'majorcards']

X2 = X.drop(potential\_leaks, axis=1)

*# Evaluate the model with leaky predictors removed*

cv\_scores = cross\_val\_score(my\_pipeline, X2, y,

cv=5,

scoring='accuracy')

print("Cross-val accuracy: **%f**" % cv\_scores.mean())

Cross-val accuracy: 0.830919

Bu doğruluk oranı oldukça düşük, bu da hayal kırıklığı yaratabilir. Ancak, yeni başvurularda kullanıldığında %80 oranında doğru olmasını bekleyebiliriz; oysa sızıntı içeren modelin, çapraz doğrulamadaki görünürdeki yüksek skoruna rağmen, gerçekte bundan çok daha kötü performans göstermesi muhtemeldir.

# Conclusion

Veri sızıntısı, birçok veri bilimi uygulamasında milyonlarca dolarlık bir hataya yol açabilir. Eğitim ve doğrulama verilerinin dikkatli bir şekilde ayrılması, eğitim-test kirlenmesini önleyebilir ve veri hatları bu ayrımın uygulanmasına yardımcı olabilir. Aynı şekilde, dikkat, sağduyu ve veri araştırmasının birleşimi, hedef sızıntının belirlenmesine yardımcı olabilir.