Data leakage can ruin our models in subdle ways. So, this is one of the most important concepts for practicing data scientists.

What is a data leakage?

Data leakage (or leakage) happens when your training data contains information about the target, but similar data will not be available when the model is used for prediction.

This leads to high performance on the training set (and possibly even the validation data), but the model will perform poorly in production.

In other words, leakage causes a model to look accurate until you start making decisions with the model, and then the model becomes very inaccurate.

Data leakage can happen at any step in the project life cycle of a data science project. Two of them being,

1. Target Leakage
2. train-test contamination

1. What is Target Leakage?

Target leakage occurs when your predictors include data that will not be available at the time you make predictions or some future data.

How to prevent it?

It is important to think about target leakage in terms of the timing or chronological order that data becomes available, not merely whether a feature helps make good predictions.

An example will be helpful. Imagine you want to predict who will get sick with pneumonia. The top few rows of your raw data look like this:

In [ ]:
```
got_pneumonia age weight male took_antibiotic_medicine...
False 65 100 False False...
False 72 130 True False...
True 58 100 False True...
```

The model would see that anyone who has a value of False for took_antibiotic_medicine didn't have pneumonia. Since validation data comes from the same source as training data, the pattern will repeat itself in validation, and the model will have great validation (or cross-validation) scores.

But the model will be very inaccurate when subsequently deployed in the real world, because even patients who will get pneumonia won't have received antibiotics yet when we need to make predictions about their future health.

At the time of prediction, we won't be knowing if the patient will likely to have pneumonia or not. So, "took_antibiotic_medicine" will not make any sense at the time of prediction as the patient has not yet been diagnoised with pneumonia yet. Or, the patient has not become yet a patient at the time of prediction for the likelihood of the disease.

Using "took_antibiotic_medicine" will be equivalent to using future data to predict the present

2. Train-Test Contamination

Recall that validation is meant to be a measure of how the model does on data that it hasn't considered before. You can corrupt this process in subtle ways if the validation data affects the preprocessing behavior.

This is sometimes called train-test contamination.

For example, imagine you run preprocessing (like fitting an imputer for missing values) before calling train_test_split(). The end result? Your model may get good validation scores, giving you great confidence in it, but perform poorly when you deploy it to make decisions.

Another example would be performing scaling like StandardScalar on our dataset before splitting it. Then feature scaling will carry some information of test set into the training set or we can say possibility of data leakage

If your validation is based on a simple train-test split, exclude the validation data from any type of fitting, including the fitting of preprocessing steps. This is easier if you use scikit-learn pipelines. When using cross-validation, it's even more critical that you do your preprocessing inside the pipeline!

In [1]:
```
import pandas as pd

# Read the data
data = pd.read_csv('datasets/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
```

Out[1]:

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

In [5]:
```
# cross-validating

from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
```

```
my_pipeline = make_pipeline(RandomForestClassifier(n_estimators=100))

# my_pipeline = Pipeline(
#     steps=[
#         ('model', 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
```

you'll find that it's very rare to find models that are accurate 98% of the time. It happens, but it's uncommon enough that we should inspect the data more closely for target leakage.

Here is a summary of the data, which you can also find under the data tab:

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

A few variables look suspicious. For example, does expenditure mean expenditure on this card or on cards used before appying?

At this point, basic data comparisons can be very helpful:

In [6]:

```
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
```

As shown above, everyone who did not receive a card had no expenditures, while only 2% of those who received a card had no expenditures. It's not surprising that our model appeared to have a high accuracy.

But this also seems to be a case of target leakage, where expenditures probably means expenditures on the card they applied for.

Since share is partially determined by expenditure, it should be excluded too. The variables active and majorcards are a little less clear, but from the description, they sound concerning. In most situations, it's better to be safe than sorry if you can't track down the people who created the data to find out more.

In [7]:

```
# Drop predictors from dataset which might lead to data leakage
potential_leaks = ['expenditure', 'share', 'active', 'majorcards']
X2 = X.drop(potential_leaks, axis=1)

# Evaluate the model without suspicious predictors
cv_scores = cross_val_score(my_pipeline, X2, y,
                            cv=5,
                            scoring='accuracy')

print("Cross-val accuracy: %f" % cv_scores.mean())
```

```
Cross-val accuracy: 0.830165
```

We can find that the accuracy is very much lower than before which might seem disappointing. But we can expect it to be right about 80% of the time when used on new applications.

Whereas, the previous model might do much worse than that on new data. (Inspite of high cross-validation scores).

Points to keep in mind

1. Data Leakage can be a huge mistake in many data science applications.
1. Separation of training and validation data carefully before doing any pre-processing on the data can prevent train-test contamination. (Pipelines can help in implementing this separation).

   ```
   2.1. Always fit on training data and use the frozen parameter to
   transform the test data


   2.2. Don't fit on test data.
   ```

1. Caution, common sense, domain expertise and EDA can help identify target leakage.

Some Great Examples from Kaggle

The Data Science of Shoelaces

Nike has hired you as a data science consultant to help them save money on shoe materials. Your first assignment is to review a model one of their employees built to predict how many shoelaces they'll need each month. The features going into the machine learning model include:

1. The current month (January, February, etc)
1. Advertising expenditures in the previous month
1. Various macroeconomic features (like the unemployment rate) as of the beginning of the current month
1. The amount of leather they ended up using in the current month

The results show the model is almost perfectly accurate if you include the feature about how much leather they used. But it is only moderately accurate if you leave that feature out. You realize this is because the amount of leather they use is a perfect indicator of how many shoes they produce, which in turn tells you how many shoelaces they need.

Do you think the leather used feature constitutes a source of data leakage? If your answer is "it depends," what does it depend on?

Answer:

This is tricky, and it depends on details of how data is collected (which is common when thinking about leakage). Would you at the beginning of the month decide how much leather will be used that month? If so, this is ok. But if that is determined during the month, you would not have access to it when you make the prediction. If you have a guess at the beginning of the month, and it is subsequently changed during the month, the actual amount used during the month cannot be used as a feature (because it causes leakage).

Return of the Shoelaces

You have a new idea. You could use the amount of leather Nike ordered (rather than the amount they actually used) leading up to a given month as a predictor in your shoelace model.

Does this change your answer about whether there is a leakage problem? If you answer "it depends," what does it depend on?

Answer:

This could be fine, but it depends on whether they order shoelaces first or leather first. If they order shoelaces first, you won't know how much leather they've ordered when you predict their shoelace needs. If they order leather first, then you'll have that number available when you place your shoelace order, and you should be ok.

Getting Rich With Cryptocurrencies?

You saved Nike so much money that they gave you a bonus. Congratulations.

Your friend, who is also a data scientist, says he has built a model that will let you turn your bonus into millions of dollars. Specifically, his model predicts the price of a new cryptocurrency (like Bitcoin, but a newer one) one day ahead of the moment of prediction. His plan is to

purchase the cryptocurrency whenever the model says the price of the currency (in dollars) is about to go up.

The most important features in his model are:

1. Current price of the currency
2. Amount of the currency sold in the last 24 hours
3. Change in the currency price in the last 24 hours
4. Change in the currency price in the last 1 hour
5. Number of new tweets in the last 24 hours that mention the currency

The value of the cryptocurrency in dollars has fluctuated up and down by over 100 dollars in the last year, and yet his model's average error is less than 1 dollar. He says this is proof his model is accurate, and you should invest with him, buying the currency whenever the model says it is about to go up.

Is he right? If there is a problem with his model, what is it?

Answer:

There is no source of leakage here. These features should be available at the moment you want to make a predition, and they're unlikely to be changed in the training data after the prediction target is determined. But, the way he describes accuracy could be misleading if you aren't careful. If the price moves gradually, today's price will be an accurate predictor of tomorrow's price, but it may not tell you whether it's a good time to invest. For instance, if it is 100 dollars today, a price of 100 dollars tomorrow may seem accurate, even if it can't tell you whether the price is going up or down from the current price. A better prediction target would be the change in price over the next day. If you can consistently predict whether the price is about to go up or down (and by how much), you may have a winning investment opportunity.

Preventing Infections

An agency that provides healthcare wants to predict which patients from a rare surgery are at risk of infection, so it can alert the nurses to be especially careful when following up with those patients.

You want to build a model. Each row in the modeling dataset will be a single patient who received the surgery, and the prediction target will be whether they got an infection.

Some surgeons may do the procedure in a manner that raises or lowers the risk of infection. But how can you best incorporate the surgeon information into the model?

You have a clever idea.

Take all surgeries by each surgeon and calculate the infection rate among those surgeons. For each patient in the data, find out who the surgeon was and plug in that surgeon's average infection rate as a feature. Does this pose any target leakage issues? Does it pose any train-test contamination issues?

Answer:

This poses a risk of both target leakage and train-test contamination (though you may be able to avoid both if you are careful).

You have target leakage if a given patient's outcome contributes to the infection rate for his surgeon, which is then plugged back into the prediction model for whether that patient becomes infected. You can avoid target leakage if you calculate the surgeon's infection rate by using only the surgeries before the patient we are predicting for. Calculating this for each surgery in your training data may be a little tricky.

You also have a train-test contamination problem if you calculate this using all surgeries a surgeon performed, including those from the test-set. The result would be that your model could look very accurate on the test set, even if it wouldn't generalize well to new patients after the model is deployed. This would happen because the surgeon-risk feature accounts for data in the test set. Test sets exist to estimate how the model will do when seeing new data. So this contamination defeats the purpose of the test set.

Housing Prices

You will build a model to predict housing prices. The model will be deployed on an ongoing basis, to predict the price of a new house when a description is added to a website. Here are four features that could be used as predictors.

1. Size of the house (in square meters)
2. Average sales price of homes in the same neighborhood
3. Latitude and longitude of the house
4. Whether the house has a basement

You have historic data to train and validate the model.

Which of the features is most likely to be a source of leakage?

Hint: Which of these features might be updated in a database after the house is sold? That's the one to worry about.

Answer:

2 is the source of target leakage. Here is an analysis for each feature:

1. The size of a house is unlikely to be changed after it is sold (though technically it's possible). But typically this will be available when we need to make a prediction, and the data won't be modified after the home is sold. So it is pretty safe.

1. We don't know the rules for when this is updated. If the field is updated in the raw data after a home was sold, and the home's sale is used to calculate the average, this constitutes a case of target leakage. At an extreme, if only one home is sold in the neighborhood, and it is the home we are trying to predict, then the average will be exactly equal to the value we are trying to predict. In general, for neighborhoods with few sales, the model will perform very well on the training data. But when you apply the model, the home you are predicting won't have been sold yet, so this feature won't work the same as it did in the training data.

1. These don't change, and will be available at the time we want to make a prediction. So there's no risk of target leakage here.

1. This also doesn't change, and it is available at the time we want to make a prediction. So there's no risk of target leakage here.

In [ ]:

Leakage is a hard and subtle issue. You should be proud if you picked up on the issues in these examples.

Now you have the tools to make highly accurate models, and pick up on the most difficult practical problems that arise with applying these models to solve real problems.

In [ ]:

Author: Piyush Kumar

Github

In [ ]: