

WEEK 2 - Part 2

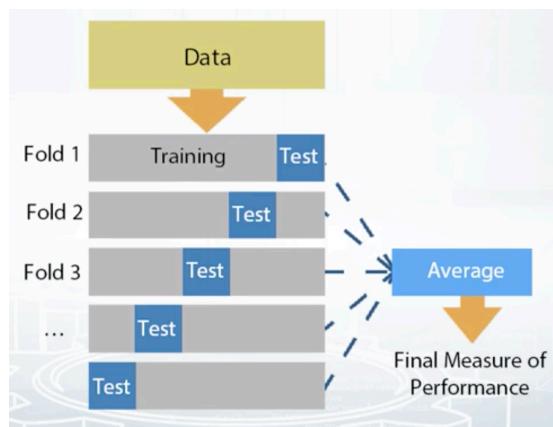
Validation Strategies

- Hold out (Just one validation set.)
- K-Fold
- Leave One Out = K-fold when k=n.

HoldOut: `sklearn.model_selection.ShuffleSplit`

K-Fold : `sklearn.model_selection.Kfold`

LOO: `sklearn.model_selection.LeaveOneOut`



Stratification:

We usually use holdout or K-fold on shuffle data. By shuffling data we are trying to reproduce random trained validation split. But sometimes, especially if you do not have enough samples for some class, a random split can fail.

Let's consider following example. We have binary classification tests and a small data set with eight samples. Four of class 0, and four of class 1. Let's split the data into four folds. Done, but notice, we are not always getting 0 and 1 in the same problem. If we'll use the second fold for validation, we'll get an average value of the target in the train of 2/3 instead of 0.5. This can drastically change predictions of our model. What we need here to handle this problem is stratification. It is just the way to ensure we'll get similar target distribution over different folds. If we split data into four folds with stratification, the average of each false target values will be 0.5.

It is easy to guess significance of this problem is higher, first for small data sets, like in this example, second for unbalanced data sets. And for binary classification, that could be, if target average were very close to 0 or vice versa, very close to 1. And third, for multi-class classification tasks with huge amount of classes. For good classification datasets, stratification split will be quite similar to a simple shuffle split, i.e. to a random split.

Samples and their target values

0	1	0	0	1	1	1	0
0	1	0	0	1	1	1	0
0.5		0		1			0.5
0	1	0	0	1	1	1	0
0.5	0.5	0.5	0.5	0.5	0.5	0.5	

Stratification is useful for:

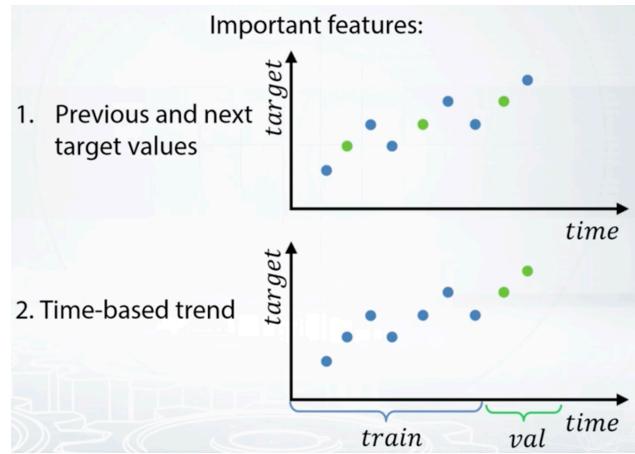
- Small datasets
- Unbalanced datasets
- Multiclass classification

Stratification preserves the same target distribution over different folds.

[Read More.](#)

Data Splitting Strategies

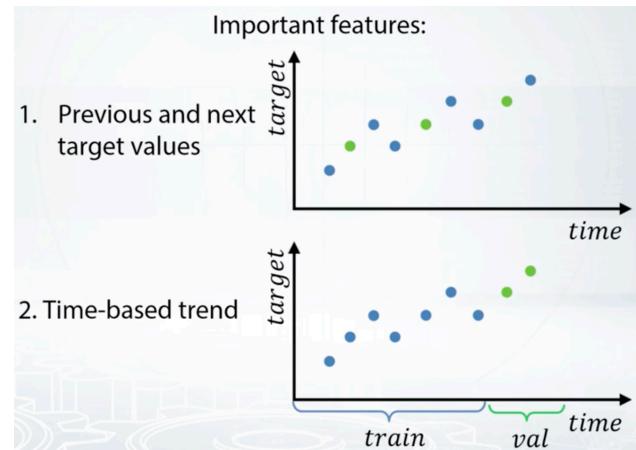
Since we already know the main strategies for validation, we can move to more concrete examples. Let's imagine, we're solving a competition with a time series prediction, namely, we are to predict a number of customers for a shop for which they're due in next month. How should we divide the data into train and validation here? Basically, we have two possibilities. Having data frame first, we can take random rows in validation and second, we can make a time-based split, take everything before some date as a train and everything out there as a validation.



In the first plot, we can just interpolate between the previous and the next value to get our predictions. Very easy, but wait. Do we really have future information about the number of customers in the real world? Well, probably not. But does this mean that this validation is useless? Again, it doesn't. What it does mean is that if we make train/validation split different from train/test split, then we are going to create a useless model.

And here, we get to the main rule of making a reliable validation. We should, if possible, set up validation to mimic train/test split, but that's a little later.

On the second picture, for most of test point, we have neither the next value nor the previous one. Now, let's imagine we have a pool of different models trained on different features, and we selected the best model for each type of validation. Now, the question, will these models differ? And if they will, how significantly? Well, it is certain that if you want to predict what will happen a few points later, then the model which favor features like previous and next target values will perform poorly. It happens because in this case, we just don't have such observations for the test data. But we have to give the model something in the feature value, and it probably will be NaN or missing values. How much experience that model have with these type of situations? Not much. The model just won't expect that and quality will suffer. Now, let's remember the second case. Actually, here we need to rely more on the time trend. And so, the features, which is the model really we need here, are more like what was the trend in the last couple of months or weeks? So, that shows that the model selected as the best model for the first type of validation will perform poorly for the second type of validation.

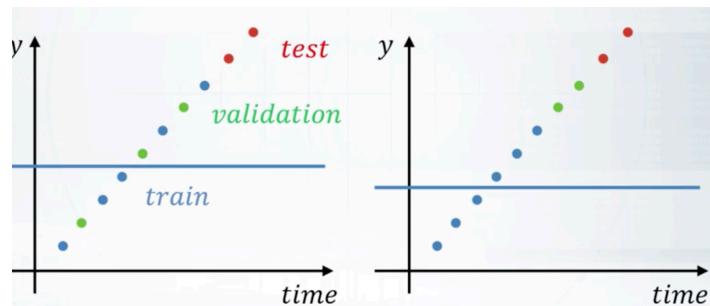


On the opposite, the best model for the second type of validation was trained to predict many points ahead, and it will not use adjacent

target values. So, to conclude this comparison, these models indeed differ significantly, including the fact that most useful features for one model are useless for another.

But, the generated features are not the only problem here. Consider that actual train/test split is time-based, here is the question. If we carefully generate features that are drawing attention to time-based patterns, we'll get a reliable validation with a random-based split? Let me say this again in another words. If we'll create features which are useful for a time-based split and are useless for a random split, will be correct to use a random split to select the model? It's a tough question. Let's take a moment and think about it.

Okay, now let's answer this. Consider the case when target follows a linear



trade. In the first plot, we see the exact case of randomly chosen validation. In the second, we see the same time-based split as we considered before. First, let's notice that in general, model predictions will be close to targets mean value calculated using train data. So in the first plot, if the validation points will be closer to this mean value compared to test points, we'll get a better score in validation than on test. But in the second case, the validation points are roughly as far as the test points from target mean value. And so, in the second case, validation score will be more similar to the test score.

Great, as we just found out, in the case of incorrect validation, not only features, but the value target can lead to unrealistic estimation of the score.

Different splitting strategies can differ significantly, namely:

1. In generated features,
2. In the way the model will rely on that features, and
3. In some kind of target leak.

That means, to be able to find smart ideas for feature generation and to consistently improve our model, we absolutely want to identify

train/test split made by organizers, including the competition, and reproduce it.

Let's now categorize most of these splitting strategies and competitions, and discuss examples for them.

- Random, row-wise
- Time Wise
- By ID
- Combined

Let's start with the most basic one, the **random split**. The most common way of making a train/test split is to split data randomly by rows. This usually means that the rows are independent of each other. For example, we have a test of predicting if a client will pay off a loan. Each row represents a person, and these rows are fairly independent of each other. Now, let's consider that there is some dependency, for example, between family members or people which work in the same company. If a husband can pay a credit probably, his wife can do it too. That means if by some misfortune, a husband will be present in the train data and his wife will be present in the test data. We probably can exploit this and devise a special feature for that case. For such possibilities, and realizing that kind of features is really interesting.

- **Time base split**

We already discussed the vivid example of the split in the beginning of this video. In that case, we generally have everything before a particular date as a training data, and everything after date as a test data. This can be a signal to use special approach to feature generation, especially to make useful features based on the target. For example, if we are to predict a number of customers for the shop for each day in the next week, we can come up with something like the number of customers for the same day in the previous week, or the average number of customers for the past month.

A special case of validation for the time-based split is a moving window validation.

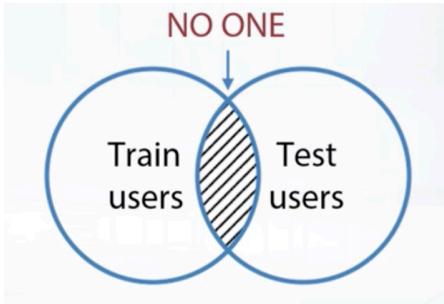
In the previous example, we can move the date which divides train and validation. Successively using week after week as a validation set, just like on this picture.

Moving window validation

week1	week2	week3	week4	week5	week6
	train		validation		
	train			validation	
		train			validation

Let's discuss the **ID-based** split.

For example, let's imagine we have to solve a task of music recommendations for completely new users. That means, we have different sets of users in train and test. If so, we probably can make a conclusion that features based on user's history, for example, how many songs user listened in the last week, will not help for completely new users. As an example of ID-based split, I want to tell you a bit about the Caterpillar to pricing competition. In that competition, train/test split was done on some category ID, namely, tube ID. There is an interesting case when we should employ the ID-based split, but IDs are hidden from us. Here, I want to mention two examples of competitions with hidden ID-based split. These include Intel and MumbaiODT Cervical Cancer Screening competition, and The Nature Conservancy fisheries monitoring competition. In the first competition, we had to classify patients into 3 classes, and for each patient, we had several photos. Indeed, photos of one patient belong to the same class. Again, sets of patients from train and test did not overlap. And we should also ensure these in the training regulations split.



As another example, in The Nature Conservancy fisheries monitoring competition, there were photos of fish from several different fishing boats. Again, fishing boats and train and test did not overlap. So one could easily overfit if you would ignore this and make a random split.

Because the IDs were not given, competitors had to derive these IDs by themselves. In both these competitions, it could be done by clustering pictures.

The easiest case was when pictures were taken just one after another, so the images were quite similar. You can find more details of such clustering in the kernels of these competitions.

Now, having in these two main standalone methods, we also need to know that they sometimes may be **combined**. For example, if we have a task of predicting sales in a shop, we can choose a split in date for each shop independently, instead of using one date for every shop in the data. Or another example, if we have search queries from multiple users, is using several search engines, we can split the data by a combination of user ID and search engine ID. Examples of competitions with combined splits include the Western Australia Rental Prices competition by Deloitte and their qualification phase of data science game 2017. In the first competition, train/test was split by a single date, but the public/private split was made by different dates for different geographic areas.

In the second competition, participants had to predict whether a user of online music service will listen to the song. The train/test split was made in the following way. For each user, the last song he listened to was placed in the test set, while all other songs were placed in the train set. These were the main splitting strategies employed in the competitions.

Again, the main idea I want you to take away from this lesson is that your validation should always mimic train/test split made by organizers. It could be something non-trivial. For example, in the Home Depot Product Search Relevance competition, participants were asked to estimate search relevancy. In general, data consisted of search terms and search results for those terms, but test set contained completely new search terms. So, we couldn't use either a random split or a search term-based split for validation. First split favored more complicated models, which led to overfitting while second split, conversely, to under-fitting. So, in order to select optimal models, it was crucial to mimic the ratio of new search terms from train/test split.

We just demonstrated major data splitting strategies employed in competitions. Random split, time-based split, ID-based split, and their combinations. This will help us build reliable validation, make useful decisions about feature generation, and in the end, select models which will perform best on the test data. As the main point of this video, remember the general rule of making a reliable validation. Set up your validation to mimic the train/test split of the competition.

1. In most cases data is split by
 - a. Row number
 - b. Time
 - c. Id
2. Logic of feature generation depends on the data splitting strategy
3. Set up your validation to mimic the train/test split of the competition

Problems Occurring During Validation

- Validation Stage
- Submission Stage

In the previous videos we discussed the concept of validation and overfitting, and discussed how to choose validation strategy based on the properties of data we have. And finally we learned to identify data split made by organizers. After all this work being done, we honestly expect that the score we see on the validation will be the same for the private leaderboard. Or at least, if we improve our model on validation, there will be improvements on the private leaderboard. And this is usually true, but sometimes we encounter some problems here. In most cases these problems can be divided into two groups. In the first group are there problems we encounter during local validation. Usually they are caused by inconsistency of the data, a widespread example is getting different optimal parameters for different folds. In this case we

need to make more thorough validation. The problems from the second group, often reveal themselves only when we send our submissions to the platform. And observe that scores on the validation and on the leaderboard don't match. In this case, the problem usually occurs because we can't mimic the exact train test split on our validation. These are tough problems, and we definitely want to be able to handle them. So before we start, let me provide an overview of this video. For both validation and submission stages we will discuss main problems, their causes, how to handle them.

validation stage problems

Usually, they attract our attention during validation. Generally, the main problem is a significant difference in scores and optimal parameters for different train validation splits.

Let's start with an example. Consider that we need to predict sales in a shop in February. Say we have target values for the last year, and, usually, we will take last month in the validation. This means January, but clearly January has much more holidays than February. And people tend to buy more, which causes target values to be higher overall. And that mean squared error of our predictions for January will be greater than for February. Does this mean that the module will perform worse for February? Probably not, at least not in terms of overfitting. As we can see, sometimes this kind of model behavior can be expected. But what if there is no clear reason why scores differ for different folds? Lets identify several common reasons for this and see what we can do about it.

Too little data

The first hypotheses we should consider is that we have too little data. For example, consider a case when we have a lot of patterns and trends in the data. But we do not have enough samples to generalize these patterns well.

In that case, a model will utilize only some general patterns. And for each train/validation split, these patterns will partially differ. This indeed, will lead to a difference in scores of the model. Furthermore, validation samples will be different each time only increasing the dispersion of scores for different folds.

Causes of different scores and optimal parameters

1. Too little data
2. Too diverse and inconsistent data

We should do extensive validation

1. Average scores from different KFold splits
2. Tune model on one split, evaluate score on the other

Too Diverse and inconsistent data

For example, if you have very similar samples with different target values, a model can be confused by them.

Consider two cases:

- First, if one of such samples is in the train while another is in the validation. We can get a pretty high error for the second sample.
- The second case, if both samples are in validation, we will get smaller errors for them.
- Or let's remember another example of diverse data we have already discussed a bit earlier. The example of predicting sales for January and February. Here we have the nature or the reason for the differences in scores.

As a quick note, notice that in this example, we can reduce this diversity a bit if we will validate on the February from the previous year. So the **main reasons for a difference in scores and optimal model parameters** for different folds are, first, **having too little data**, and second, **having too diverse** and inconsistent data.

What can we do?

If we are facing this kind of problem, it can be useful to make more thorough validation.

1. You can increase K in KFold, but usually 5 folds are enough.
2. Make KFold validation several times with different random splits (seeds). And average scores to get a more stable estimate of model's quality.

The same way we can choose the best parameters for the model if there is a chance to overfit. It is useful to use one set of KFold splits to select parameters and another set of KFold splits to check model's quality.

Causes of different scores and optimal parameters

1. Too little data
2. Too diverse and inconsistent data

We should do extensive validation

1. Average scores from different KFold splits
2. Tune model on one split, evaluate score on the other

Examples of competitions which required extensive validation include the Liberty Mutual Group Property Inspection Prediction competition and the Santander Customer Satisfaction competition. In both of them, scores of the competitors were very close to each other. And thus participants tried to squeeze more from the data. But do not overfit, so the thorough validation was crucial.

Submission stage problems

- LB score is consistently higher/lower than validation score.
- LB score is not correlated to the validation score at all.

Sometimes you can diagnose these problems in the process of doing careful. But still, often you encounter these type of problems only when you submit your solution to the platform. But then again, EDA is your friend when it comes down to finding the root of the problem. Generally speaking, there are two cases of these issues. In the first case, leaderboard score is consistently higher or lower than validation score. In the second, leaderboard score is not correlated with validation score at all.

So in the worst case, we can improve our score on the validation. While, on the contrary, score on the leaderboard will decrease. As you can imagine, these problems can be much more trouble. Now remember that the main rule of making a reliable validation, is to mimic a train/test split made by organizers. I won't lie to you, it can be quite hard to identify and mimic the exact train/test here. Because of that, I highly recommend you to **start submitting your solutions right after you enter the competition**.

It's good to start exploring other possible roots of this problem. Let's first sort out causes we could observe during validation stage.

- We may already have quite different scores in KFold.

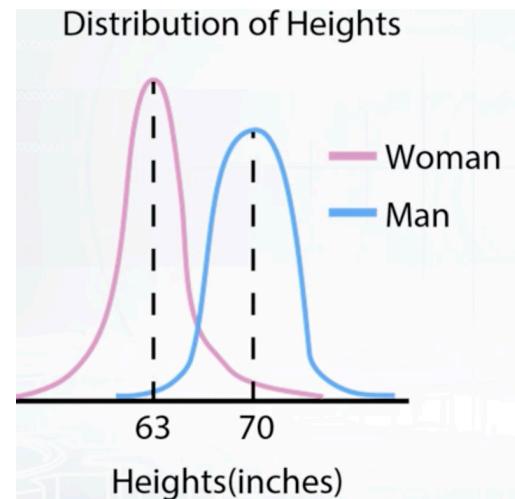
Here it is useful to see a leaderboard as another validation fold. Then, if we already have different scores in KFold, getting a not very similar result on the leaderboard is not surprising. More we can calculate mean and standard deviation of the validation scores and estimate if the leaderboard score is expected. But if this is not the case, then something is definitely wrong.

There could be two more reasons for this problem.

- We already have different scores in KFold
- The first reason: Too little data in public leaderboard, which is pretty self explanatory. Just trust your validation, and everything will be fine.
- The second train and test data are from different distributions.

Now, because our course is a practical one, let's take a moment and think what you can do if you encounter these in a competition.

Let me explain what I mean when I talk about different distributions. Consider a regression task of predicting people's height by their photos on Instagram. The blue line represents the distribution of heights for man, while the red line represents the distribution of heights for women. As you can see, these distributions are different. Now let's consider that the train data consists only of women, while the test data consists only of men. Then all model predictions will be around the average height for women. And the distribution of these predictions will be very similar to that for the train data. No wonder that our model will have a terrible score on the test data. Now, because our course is a practical one, let's take a moment and think what you can do if you encounter this in a competition.



Let's start with a general approach to such problems. At the broadest level, we need to find a way to tackle different distributions in train

and test. Sometimes, these kind of problems could be solved by adjusting your solution during the training procedure. But sometimes, this problem can be solved only by adjusting your solution through the leaderboard. That is through leaderboard probing. The simplest way to solve this particular situation in a competition is to try to figure out the optimal constant prediction for train and test data. And shift your predictions by the difference. Right here we can calculate the average height of women from the train data.

Calculating the average height of men is a bit trickier. If the competition's metric is means squared error, we can send two constant submissions, write down the simple formula. And find out that the average target value for the test is equal to 70 inches. In general, this technique is known as leaderboard probing. And we will discuss it in the topic about leaks. So now we know the difference between the average target values for the train and the test data, which is equal to 7 inches. And as the third step of adjusting our submission to the leaderboard we could just try to add 7 to all predictions. But from this point it is not validational it is a leaderboard probing and list. Yes, we probably could discover this during exploratory data analysis and try to make a correction in our validation scheme. But sometimes it is not possible without leaderboard probing, just like in this example. A competition which has something similar is the Quora question pairs competition. There, distributions of the target from train and test were different. So one could get a good improvement of a score adjusting his predictions to the leaderboard.

But fortunately, this case is rare enough. More often, we encounter situations which are more like the following case. Consider that now train consists not only of women, but mostly of women, and test, consists not only of men, but mostly of men.

The main strategy to deal with these kind of situations is simple. Again, remember to mimic the train test split. If the test consists mostly of Men, force the validation to have the

Submission stage: different distributions



same distribution. In that case, you ensure that your validation will be fair.

This is true for getting both scores and optimal parameters correctly. For example, we could have quite different scores and optimal parameters for women's and men's parts of the data set.

Ensuring the same distribution in test and validation helps us get scores and parameters relevant to test. I want to mention two examples of this here. First the Data Science Game Qualification Phase: Music recommendation challenge. And second, competition with CTR prediction which we discussed earlier in the data topic. Let's start with the second one, do you remember the problem? we have a task of predicting CTR. So, the train data, which basically was the history of displayed ads obviously didn't contain ads which were not shown. On the contrary, the test data consisted of every possible ad. Notice this is the exact case of different distributions in train and test. And again, we need to set up our validation to mimic test here. So we have this huge bias towards showing ads in the train and to set up a correct validation. We had to complete the validation set with rows of not shown ads.

Now, let's go back to the first example. In that competition, participants had to predict whether a user will listen to a song recommended by the system. So, the test contained only recommended songs. But train, on the contrary, contained both recommended songs and songs users selected themselves. So again, one could adjust his validation by 50 renowned songs selected by users. And again, if we will not account for that fact, then improving our model on actually selected songs can result in the validation score going up. But it doesn't have to result and the same improvements for the leaderboard.

Okay let's conclude this overview of handling validation problems for the submission stage. If you have too little data in public leaderboard, just trust your validation. If that's not the case, make sure that you did not overfit. Then check if you made correct train/test split, as we discussed in the previous video. And finally, check if you have different distributions in train and test.

Great, let's move on to the next point of this video. For now, I hope you did everything all right:

- First, you did extensive validation.
- Second, you choose a correct splitting strategy for train/validation split.
- Finally, you ensured the same distributions in validation and test.

But sometimes you have to expect leaderboard shuffle anyway, and not just for you, but for everyone. First, for those who never heard of it, a leaderboard shuffle happens when participants position some public and private leaderboard drastically differ. Take a look at this screenshot from the two sigma financial model in challenge competition. The green and the red arrows mean how far a team moved. For example, the participant who finished the 3rd on the private leaderboard was the 392nd on the public leaderboard. Let's discuss three main reasons for **that shuffle, randomness, too little data, and different public, private distributions**. So first, randomness, this is the case when all participants have very similar scores. This can be either a very good score or a very poor one. But the main point here is that the main reason for differences in scores is randomness. To understand this a bit more, let's go through two quick examples here. The first one is the Liberty Mutual Group, Property Inspection Prediction competition. In that competition, scores of competitors were very close. And though randomness didn't play a major role in that competition, still many people overfit on the public leaderboard. The second example, which is opposite to the first is the TWO SIGMA Financial Model and Challenge competition. Because the financial data in that competition was highly unpredictable, randomness played a major role in it. So one could say that the leaderboard shuffle there was among the biggest shuffles on Kaggle platform.

#	Δpub	Team Name	Kernel	Team Members	Score	Entries	Last
1	▲ 35	Dr.Knope			0.0382244	26	5mo
2	▲ 11	NimaShahbazi & mchahhou			0.0369387	170	5mo
3	▲ 389	rnrq		rnrq	0.0343235	70	5mo
4	▲ 6	Data Finance			0.0323850	100	5mo
5	—	best fitting			0.0320763	172	5mo
6	▲ 33	NIWATORI			0.0301690	31	5mo
7	▲ 8	E2			0.0291539	43	5mo
8	▲ 11	John Ma			0.0289587	97	5mo
9	▲ 25	Pradeep and Arthur			0.0287992	111	5mo
10	▼ 4	William Hau			0.0287899	165	5mo

Okay, that was randomness, the second reason to expect leaderboard shuffle is **too little data** overall, and in private test set especially. An example of this is the Restaurant Revenue Prediction Competition. In that competition, training set consisted of less than 200 rows. And this set consisted of less than 400 rows. So as you can see shuffle here was more than expected.

Last reason of leaderboard shuffle could be **different distributions between public and private test sets**. This is usually the case with time series prediction, like the Rossmann Stores Sales competition. When we have a time-based split, we usually have first few weeks in public leaderboard, and next few weeks in private leaderboard. As people tend to adjust their submission to public leaderboard and overfit, we can expect worse results on private leaderboard. Here again, trust your validation and everything will be fine. Okay, that is all with reasons for leaderboard shuffling.

Conclusion

Now let's conclude both this video and the entire validation topic. Let's start with the video.

- First, if you have big dispersion of scores on validation stage we should do extensive validation. That means
 - every score from different KFold splits, and
 - tune model on one split while evaluating score on the other.
- Second, if submission do not match local validation score, we should.
 - first, check if we have too little data in public leaderboard.
 - Second, check if we did not overfit,
 - check if you chose correct splitting strategy.
 - And finally, check if trained test have different distributions.

You can expect leaderboard shuffle because of three key things, randomness, little amount of data, and different public/private test distributions. So that's it, in this topic we defined validation and its connection to overfitting. Described common validation strategies. Demonstrated major data splitting strategies. And finally analyzed and learned how to tackle main validation problems. Remember this, and it will absolutely help you out in competitions. Make sure you understand the main idea of validation well. That is, you need to mimic the train/test split.

Practice Quiz

1- Suppose we are given a huge dataset. We did a KFold validation once and noticed that scores on each fold are roughly the same. Which validation type is most practical to use?

- A. We can use a simple hold out validation scheme bc the data is homogenous.**
- B. We should keep on using Kfold scheme as data is homogenous and Kfold is the most computationally efficient scheme.
- C. LOO bc the data is not homogenous.

2- Suppose we are given a medium-sized dataset and we did a KFold validation once. We noticed that scores on each fold differ noticeably. Which validation type is the most practical to use?

- A. Hold out
- B. LOO
- C. Kfold**

3- The features we generate depend on the train-test data splitting method. Is this true?

- A. False
- B. True**

4- What of these can indicate an expected leaderboard shuffle in a competition?

- A. Most of the competitions have similar scores.**
- B. Little amount of training and/or testing data.**
- C. Different public/private data or target distributions.**

Quiz

1- Select true statements

- A. Underfitting refers to not capturing enough patterns in the data
- B. We use validation to estimate the quality of our model
- C. The model, that performs best on the validation set is guaranteed to be the best on the test set.
- D. Performance increase on a fixed cross-validation split guarantees performance increase on any cross-validation split.
- E. The logic behind validation split should mimic the logic behind train-test split.

2- Usually on Kaggle it is allowed to select two final submissions, which will be checked against the private LB and contribute to the competitor's final position. A common practice is to select one submission with a best validation score, and another submission which scored best on Public LB. What is the logic behind this choice?

- A. Generally, this approach is based on the assumption that people rarely tend to overfit to the Public LB. Almost always you have a lot of data in the test set and it is quite hard to overfit. Indeed, this render validation useless.
- B. Generally, this approach is based on the assumption that the test data may have a different target distribution compared to the train data. If that would be the true, the submission which was chosen based on Public LB, will perform better. If, otherwise, the above distributions will be similar, the submission which was chosen based on validation scores, will perform better.
- C. Generally, this approach is based on the assumption that validation is rarely valid in competitions. Often it is hard to trust your validation and thus you should account for both cases if the validation will succeed and if the validation will fail.

3- Suppose we have a competition where we are given a dataset of marketing campaigns. Each campaign runs for a few weeks and for each day in campaign we have a target - number of new customers involved. Thus the row in a dataset looks like

Campaign_id, Date, {some features}, Number_of_new_customers

Test set consists of multiple campaigns. For each of them we are given several first days in train data. For example, if a campaign runs for two weeks, we could have three first days in train set, and all next days will be present in the test set. For another campaign, running for weeks, we could have the first 6 days in the train set, and the remaining days in the test set.

Identify train/test split in a competition.

- A. Id-based split
- B. Combined split
- C. Random
- D. Time based

4- Which of the following problems you usually can identify without the Leaderboard?

- A. Train and test target distribution are from different distributions
- B. Public leaderboard score will be unreliable because of too little data
- C. Different scores/optimal parameters between folds
- D. Train and test data are from different distributions

- [Validation in Sklearn](#)
- [Advices on validation in a competition](#)

Data Leakage

Basic Data Leaks

In this section, we will talk about a very sensitive topic data leakage or more simply. We'll define leakage in a very general sense as an unexpected information in the data that allows us to make unrealistically good predictions. For the time being, you may have think of it as of directly or indirectly adding ground truths into the test data. Data leaks are very, very bad. They are completely unusable in real world. They usually provide way too much signal and thus make competitions lose its main point, and quickly turn them into a leak hunt race.

Further in this section, I will show you the main types of data leaks that could appear during solving a machine learning problem.

- Leakage types and examples
- Competition specific. Leaderboard Probing
- Concrete Walkthrough

Also focus on a competition specific leak exploitation technique leaderboard probing. Finally, you will find special videos dedicated to the most interesting and non-trivial data leaks. I will start with the most typical data leaks that may occur in almost every problem.

Time series is our first target. Typically, future picking. It is common sense not to pick into the future like, can we use stock market's price

from day after tomorrow to predict price for tomorrow? Of course not.

However, direct usage of future information in incorrect time splits still exist. When you enter a time serious competition at first, check train, public, and private splits. If even one of them is not on time, then you found a data leak. In such case, unrealistic features like prices next week will be the most important. But even when split by time, data still contains information about future. We still can access the rows from the test set. We can have **future user history in CTR task**, some fundamental indicators in stock market predictions tasks, and so on. There are only two ways to eliminate the possibility of data leakage. It's called competitions, where one can not access rows from future or a test set with no features at all, only IDs.

For example, just day number and instrument ID in stock market prediction, so participants create features based on past and join them themselves.

Now, let's discuss something more unusual. Those types of data leaks are much harder to find. We often have more than just train and test files. For example, a lot of images or text in archive. In such case, we can access some **meta information**, **file creation date**, **image resolution** etcetera. It turns out that this meta information may be connected to target variable. Imagine classic cats versus dogs classification. What if cat pictures were taken before dog? Or taken with a different camera? Because of that, a good practice from organizers is to erase the meta data, resize the pictures, and change creation date. Unfortunately, sometimes we will forget about it. A good example is Truly Native competition, where one could get nearly perfect scores using just the dates from zip archives.

Another type of leakage could be found in IDs. IDs are unique identifiers of every row usually used for convenience. It makes no sense to include them into the model. It is assumed that they are automatically generated. In

Leaks in time series

- Split should be done on time.
 - In real life we don't have information from future
 - In competitions first thing to look: train/public/private split, is it on time?
- Even when split by time, features may contain information about future.
 - User history in CTR tasks
 - Weather

reality, that's not always true. **ID may be a hash of something**, probably not intended for disclosure. It may contain traces of information connected to target variable. It was a case in Caterpillar competition.

A link ID as a feature slightly improve the result. So I advise you to pay close attention to IDs and always check whether they are useful or not. Next is row order. In trivial case, data may be shuffled by target variable. Sometimes simply adding row number or relative number, suddenly improves this course. Like, in Telstra Network Disruptions competition. It's also possible to find something way more interesting like in TalkingData Mobile User Demographics competition. There was some kind of row duplication, rows next to each other usually have the same label. This is it with a regular type of leaks. To sum things up, in this video, we embrace the concept of data leak and cover data leaks from **future picking, meta data, IDs, and row order**.

Leaderboard probing and examples of rare data leaks

Now, I will tell you about a competition-specific technique tightly connected with data leaks. It's leaderboard probing. There are actually two types of leaderboard probing. The first one is simply **extracting all ground truth from public part of the leaderboard**. It's usually pretty harmless, only a little more of straining data. It is also a relatively easy to do and I have a submission change on the small set of rows so that you can unambiguously calculate ground truth for those rows from leaderboard score.

I suggest checking out the link to Alek Trott's post in additional materials. He thoroughly explains how to do it very efficiently with minimum amount of submissions. [Perfect score script by Oleg Trott](#)

Unexpected information

- Meta data
- Information in IDs
- Row order



Leaderboard probing

- Types of LB probing
- Categories tightly connected with 'id' are vulnerable to LB probing
 - Company of user in RedHat competition
 - Year, Month, Week in WestNile competition

Our main focus will be on another type of leaderboard probing. Remember the purpose of public, private split. It's supposed to protect private part of test set from information extraction. It turns out that it's still vulnerable. Sometimes, it's possible to submit predictions in such a way that will give out information about private data. It's all about consistent categories. Imagine, a chunk of data with the same target for every row. Like in the example, rows with the same IDs have the same target. Organizers split it into public and private parts.

id	...	y
1	...	0
1	...	0
1	...	0
2	...	1
2	...	1
2	...	1

But we still know that that particular chunk has the same label for every role. After setting all the predictions close to 0 in our submission for that particular chunk of data, we can expect two outcomes. The first one is when score improved, it means that ground truth in public is 0. And it also means that ground truth in private is 0 as well. Remember, our chunk has the same labels.

The second outcome is when the score became worse. Similarly, it means that ground truth in both public and private is 1. Some competitions indeed have that kind of categories. Categories that with high certainty have the same label.

You could have encountered those type of categories in Red Hat and West Nile competitions. It was a key for winning. With a lot of submissions, one can explore a good part of private test set.

It's probably the most annoying type of data leak. It's mostly technical and even if it's released close to the competition deadline, you simply won't have enough submissions to fully exploit it.

Furthermore, this is on the tip of the iceberg. When I say consistent category, I do not necessarily mean that this category has the same target. It could be consistent in different ways. The definition is quite broad. For example, target label could simply have the same distribution for public and private parts of data. It was the case in Quora Question Pairs competition. In that competition there was a binary classification task being evaluated by log loss metric. What's important target variable had different distributions in train and test, but allegedly the same and private and public parts of these data. And

because of that, we could benefit a lot via leaderboard probing. Treating the whole test set as a consistent category.

Take a look at the formula on the slide. This logarithmic loss for submission with constant predictions C big. Where N big is the real number of rows, N1 big is the number of rows with target one. And L big is the leader board score given by that constant prediction. From this equation, we can calculate N1 divided by N or in other words, the true ratio of ones in the test set. That knowledge was very beneficial. We could use it rebalance training data points to have the same distribution of target variable as in the test set. This little trick gave a huge boost in leaderboard score. As you can see, leaderboard probing is a very serious problem that could occur under a lot of different circumstances. I hope that someday it will become complete the eradicated from competitive machine learning.

Now, finally, I like to briefly walk through the most peculiar and interesting competitions with data leakage.

And first, let's take a look at Truly Native competition from different point of view. In this competition, participants were asked to predict whether the content in an HTML file is sponsored or not. As was already discussed in previous video, there was a data leak in archive dates. We can assume that sponsored and non-sponsored HTML files were gotten during different periods of time. So do we really get rid of data leak after erasing archive dates?

Adapting global mean via LB probing:

$$- L * N = \sum_{i=1}^N (y_i \ln C + (1 - y_i) \ln (1 - C))$$

$$- L * N = N_1 \ln C + (N - N_1) \ln (1 - C)$$

$$\frac{N_1}{N} = \frac{-L - \ln (1 - C)}{\ln C - \ln (1 - C)}$$



Truly Native

- Predict whether the content in an HTML file is sponsored or not
- Data leak in archive dates. But is it all?
 - Data collection
 - Date proxies



The answer is no. Texts in HTML files may be connected to dates in a lot of ways. From explicit timestamps to much more subtle things, like

news contents. As you've probably already realized, the real problem was not metadata leak, but rather data collection. Even without meta information, machine learning algorithms will focus on actually useless features. The features that only act as proxies for the date.

The next example is Expedia Hotel Recommendations, and that competitions, participants worked with logs of customer behavior. These include what customers searched for, how they interacted with search results, and clicks or books, and whether or not the search result was a travel package.

Expedia was interested in predicting which hotel group a user is going to book. Within the logs of customer behavior, there was a very tricky feature. A distance from users seeking (city to?) their hotel. Turned out, that this feature is actually a huge data leak. Using this distance, it was possible to reverse engineer two coordinates, and simply map ground truth from train set to the test set.

I strongly suggest you to check out the special video dedicated to this competition. I hope that you will find it very useful because the approaches and methods of exploiting data leak were extremely nontrivial. And you will find a lot of interesting tricks in it.

The next example is from Flavors of Physics competition. It was a pretty complicated problem dealing with physics at Large Hadron Collider. The special thing about that competition was that signal was artificially simulated. Organizers wanted a machine learning solution for something that has never been observed. That's why the signal was simulated.

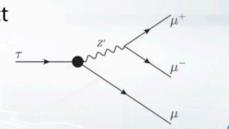
Expedia

- Predict hotel group a user is going to book
- Data leak in distance feature
- Reverse engineering true coordinates



Flavours of physics

- Machine learning problem for something that has never been observed
- Signal events were simulated
- Special statistical tests in order to punish the models that exploit simulation flaws
- However, one could by-pass the tests, fully exploit simulation flaws and get a perfect score on the leaderboard



But simulation cannot be perfect and it's possible to reverse engineer it. Organizers even created special statistical tests in order to punish the models that exploit simulation flaws. However, it was in vain. One could bypass the tests, fully exploit simulation flaws, and get a perfect score on the leaderboard.

The **last example** is going to cover pairwise tasks. Where one needs to predict whether the given pair of items are duplicates or not, like in Quora question pairs competition.

- Data leakage in item frequencies
- Similarities from connectivity matrix

There is one thing common to all the competitions with pairwise tasks. Participants are not asked to evaluate all possible pairs. There is always some nonrandom subsampling, and this subsampling is the cause of data leakage. Usually, organizers sample mostly hard-to-distinguish pairs.

Because of that, of course, imbalance in item frequencies. It results in more frequent items having the higher possibility of being duplicates. But that's not all. We can create a connectivity matrix N times N , where N is the total number of items. If item i and item j appeared in a pair then we place 1 in (i,j) and (j,i) positions. Now, we can treat the rows in connectivity matrix as vector representations for every item. This means that we can compute similarities between those vectors. This tricks works for a very simple reason.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Q

When two items have similar sets of neighbors they have a high possibility of being duplicates.

- [Page about data leakages on Kaggle](#)

Expedia Challenge

In that competition, we worked with lots of customer behavior. These include what customers searched for, how they interacted with search results, clicks or books, and whether or not the search result was a travel package, and Expedia was interested in predicting which hotel group a user is going to book. Important thing here is prediction target the hotel group. In other words, characteristics of actual hotel, remember it. As it turned out, this competition had a very non-trivial and extremely hard to exploit data

leak. From the first glance, data leak was pretty straightforward. We had a destination distance among the feature. It's a distance from user city to an actual hotel he clicked on booked. And, as I said earlier, our prediction target is a characteristic of an actual hotel. Furthermore, destination distance was very precise so unique user city and destination distance pairs corresponded to unique hotels. Putting two and two together, we can treat user city and destination distance pair as a proxy to our target.

Data leakage

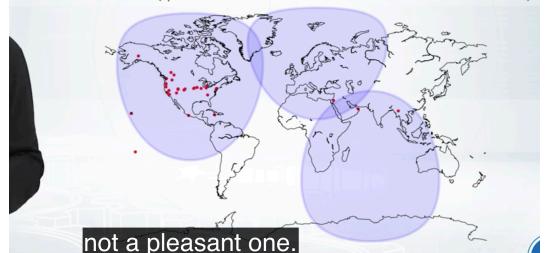
- destination_distance - user_city pair is a leak to true hotel location. A lot of matches between train and test.
- How to improve on that?
- Features based on counts on corteges of such nature
- Try to find the true coordinates

When in this set, we encountered such pair already present in train set, we could simply take a label from there as our prediction. It worked nearly perfect for the pairs present in both train and test. However, nearly half of test set consisted from new pairs without a match from train set. This way we had to go deeper. But, how exactly can we improve our solution? Well, there are two different ways. First, one is to create count features on corteges similar to user city and destination distance pair. For example, like how many hotels of which group there are for user city, hotel country, hotel city triplet. Then, we could train some machine learning model on such features. Another way is to somehow find more matches. For that purpose, we need to find true coordinates of users cities and hotel cities. From that, to guess it was destination distance feature, it was possible to find good approximation for the coordinates of actual hotels. Let's find out how to do it. First of all, we need to understand how to calculate the distance. Here, we work with geographical coordinates so the distances are geodesic. It's done via Haversine formula, not a pleasant one.

Spherical geometry

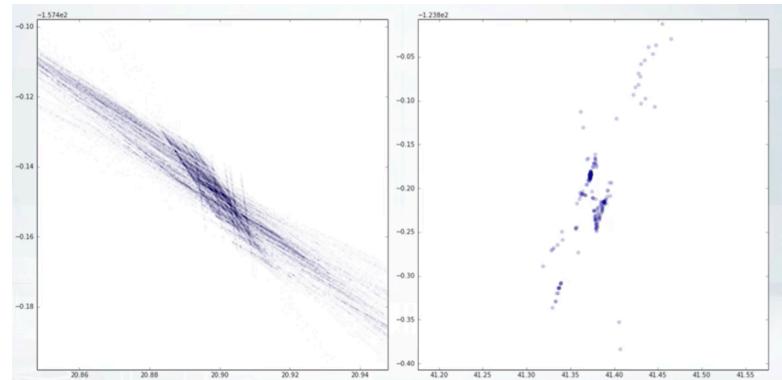
$$d = 2r \arcsin \left(\sqrt{\text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{hav}(\lambda_2 - \lambda_1)} \right)$$

$$= 2r \arcsin \left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1) \cos(\varphi_2) \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)} \right)$$



Now, suppose that we know true coordinates of three points and distances from fourth point with unknown coordinates to each of them, if you write down a system of three equations, one for each distance, we can unambiguously solve it and get true coordinates for the fourth point. Now, we have four points with known coordinates. I think you get the idea. So, at first, by hook or by crook, we reverse engineer true coordinate of three big cities. After that, we can iteratively find coordinates of more and more cities. But as you can see from the picture, some cities ended up in oceans. It means that our algorithm is not very precise. A rounding error accumulates after every iteration and everything starts to fall apart. We get some different method and indeed we can do better. Just compare this picture with the previous one. It's obviously much more accurate. Remember how in iterative method we solved a system of three equations to unambiguously find coordinates of fourth unknown point. But why limit ourselves with three equations? Let's create a giant system of equations from all known distances with true coordinates being the known variables. We end

up with literally hundreds or thousands of equations and tens of thousands of unknown variables. Good thing it's very sparse. We can apply special methods from SciPy to efficiently solve such a system. In the end, after solving that system of



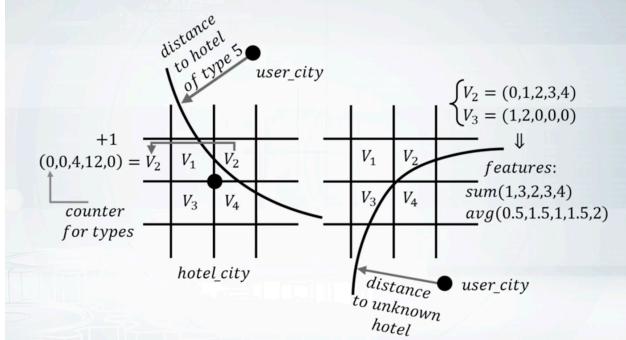
equations, we end up with a very precise coordinates for both hotel cities and user cities. But as you remember, we're predicting a type of a hotel. Using city coordinates and destination distance, it's possible to find an approximation of true coordinates of an actual hotel. When we fix user city and draw a circumference around it with the radius of destination distance, it's obvious that true hotel location must be somewhere on that circumference. Now, let's fix some hotel city and draw such circumferences from all users cities to that fixed hotel cities and draw them for every given destination distance. After doing so, we end up with pictures like the ones on the slide. A city contains a limited number of hotels so the intuition here is that hotels actually are

on the intersection points and the more circumferences intersect in such point, the higher the probability of a hotel being in that point. As you can see, the pictures are beautiful but pretty messy.

It's impossible to operate in terms of singular points. However, there are explicit clusters of points and this information can be of use. We can do some kind of integration. For every city, let's create a grid around its center.

Something like 10 kilometers times 10 kilometers with step size of 100 meters. Now, using training data, for every cell in the grid, we can count how many hotels of which type are present there. If a circumference goes through a cell, we give plus one to the hotel type corresponding to that circumference. During inference, we also draw a circumference based on destination distance feature. We see from what degree its cells it went through and use information from those cells to create features like a sum of all counters, average of all counters, maximum of all counters and so on. Great. We have covered the part of feature engineering. Note that all the features directly used target label. We cannot use them as is in training. We should generate them in out-of-fold fashion for train data. So we had training data for years 2013 and 2014. To generate features for year 2014, we used labelled data from year 2013 and vice versa, used the year 2014 to generate features for the year 2013. For the test features, which was from year 2015, we naturally used all training data. In the end, we calculated a lot of features and shoved them into XGBoost model. After 16 hours of training for the course, we got our results. We ended up on third position on public leader-boards and forth on private. We did good, but we still did not fully exploit data leakage. If you check the leaderboard, you'll notice the difference in scores between first place and the rest. Under speculation, the winner did extraordinary. Although, in general, his methods were very similar to ours. He was able to extract way more signal. Finally, I hope you enjoyed my story. As you can see, sometimes working with data leakage could be very interesting and challenging. You may develop some unusual skills and broaden your horizons.

Counters in grid cells



Quiz

1. Suppose that you have a credit scoring task, where you have to create a ML model that approximates expert evaluation of an individual's creditworthiness. Which of the following can potentially be a data leakage? Select all that apply.
 - A. First half of the data points in the train set has a score of 0, while the second half has scores > 0
 - B. An ID of a data point (row) in the train set correlates with target variable.
 - C. Among the features you have a company_id, an identifier of a company where this person works. It turns out that this feature is very important and adding it to the model significantly improves your score.
2. What is the most foolproof way to set up a time series competition?
 - A. Split train, public and private parts of data by time. Remove all features except IDs (e.g. timestamp) from test set so that participants will generate all the features based on past and join them themselves.
 - B. Split train, public and private parts of data by time. Remove time variable from test set, keep the features.
 - C. Make a time based split for train/test and a random split for public/private.
3. Suppose that you have a binary classification task being evaluated by logloss metric. You know that there are 10000 rows in public chunk of test set and that constant 0.3 prediction gives the public score of 1.01. Mean of target variable in train is 0.44. What is the mean of target variable in public part of test data (up to 4 decimal places)?

4. Suppose that you are solving image classification task. What is the label of this picture?

