[MUSIC] Hello, and welcome. My name is Dimitri, and I'm happy to see you are interested in competitive data science. Data science is all about machine learning applications. And in data science, like everywhere else, people are looking for the very best solutions to their problems. They're looking for the models that have the best predictive capabilities, the models that make as few mistakes as possible. And the competition for one becomes an essential way to find such solutions. Competing for the prize, participants push through the limits, come up with novel ideas. Companies organize data science competitions to get top quality models for not so high price. And for data scientists, competitions become a truly unique opportunity to learn, well, and of course win a prize. This course is a chance for you to catch up on the trends in competitive data science and learn what we, competition addicts and at the same time, lecturers of this course, have already learned while competing. In this course, we will go through competition solving process step by step and tell you about exploratory data analysis, basic and advanced feature generation and preprocessing, various model validation techniques. Data leakages, competition's metric optimization, model ensembling, and hyperparameter tuning. We've put together all our experience and created this course for you. We've also designed quizzes and programming assignments to let you apply your newly acquired skills. Moreover, as a final project, you will have an opportunity to compete with other students and participate in a special competition, hosted on the world's largest platform for data science challenges called Kaggle. Now, let's meet other lecturers and get started.And now, I want to introduce other lecturers of this course. Alexander, Dmitry, Mikhail, and Marios. Mikhail is aka Cassanova, the person who reached the very top of competitive data science. I will tell you a couple of thoughts about the origins of the course. In year 2014, we started our win in data science by joining competitions. We've been meeting every week and discussing the past competitions, solutions, ideas and tweaks what worked and what did not, this exchange of knowledge and experience helped us to learn quickly from each other and improve our skills. Initially our community was small, but over time more and more people were joining. From the format of groups of discussion. We moved on to the format of well organized meetings. Where a speaker makes an overview of his approach and ideas in front of 50 people. These meetings are called machine learning trainings. Now with the help and support of Yandex and get a hundred of participants. Thus we started from zero and learned everything b

y hard work and collaboration. We had an excellent teacher, Alex ander D'yakonov who was top one on Kaggle, he took the course on critical data analysis. In Moscow state university and there we 're grateful to him. At some point we started to share our knowl edge with other people and some of us even started to read lectu res at the university. So now we have decided to summarize every thing and make it available for everyone. Together. We've finish ed and procesed in about 20 different competitions only on Kaggl e and just as many on other not so famous platforms. All of us h ave a tremendous amount of skill and experience in competitive d ata science and now we want to share this experience with you. F or all of us, competitive data science opened a number of opport unities as the competitions we took part were dedicated to a lar ge variety of tasks. Mikhail works in e-commerce. Alexander buil ds predictive model for taxi services, Dmitri works with financi al data, Mario develops machinery learning frameworks and I am a deep learning researcher. Competitions, without a doubt, became a stepping stone for our careers and believe me, good comparati ve record will bring success to you as well. We hope you will fi nd something interesting in this course and wish you good luck.H ello and welcome to our course. In this video, I want to give yo u a sense for what this course is about and I think the best way to do that is to talk about our course goals, our course assign ments and our course schedule. So, at the broadest level, this c ourse is about getting the required knowledge and expertise to s uccessfully participate in data science competitions. That's the goal. Now, we're going to prepare this in a systematic way. We start in week one with a discussion of competitions, what are th ey, how they work, how they are different from real-life industr ial data analysis. Then, we're moving to recap of main machine l earning models. Besides this, we're going to review software and hardware requirements and common Python libraries for data anal ysis. After this is done, we'll go through various feature types , how we preprocess these features and generate new ones. Now, b ecause we sometimes need to extract features from text and image s, we will elaborate on most popular methods to do it. Finally, we will start working on the final project, the competition. But then we move on to week two. So, having figured out methods to work with data frames and models, we're starting to cover things you first do in a competition. And this is, by the way, a great opportunity to start working on the final project as we proceed through material. So, first in this week, we'll analyze data se t in the exploratory data analysis topic or EDA for short. We'll discuss ways to build intuition about the data, explore anonymi zed features and clean the data set. Our main instrument here wi ll be logic and visualizations. Okay, now, after making EDA, we switch to validation. And here, we'll spend some time talking ab out different validation strategies, identifying how data is spl it into train and test and about what problems we may encounter during validation and ways to address those problems. We finish this week with discussion of data leakage and leaderboard proble m. We will define data leakage and understand what are leaks, ho w to discover various leaks and how to utilize them. So basicall y, this week, we set up the main pipeline for our final project. And at this point, you should have intuition about the data, re liable validation and data leaks explored. After this pipeline i

s ready, we'll focus on the improvement of our solution and that 's already the week three. In that week, we'll analyze various m etrics for regression and classification and figure out ways to optimize them both while training the model and afterwards. Afte r we will check that we are correct in measure and improvements of our models, we'll define mean-encodings and work on the encod ed features. So here, we start with categorical features, how me an-encoded features lead to overfitting and how we balance overf itting with regularization. Then, we'll discuss several extensio ns to this approach including applying mean-encodings to numeric features and time series, and this is the point where we move o n to other advanced features in the week four. Basically, this i nclude statistics and distance-based features, metrics factoriza tions, feature interactions and t-SNE. These features often are the key to superior performance in competition, so you should im plement and optimize them here for the final project. After this , we'll get to hyperparameters optimization. Here, we will revis e your knowledge about model tuning in a systematic way and let you apply to the competition. Then, we move onto the practical g uide where all of us have summarized most important moments abou t competitions which became absolutely clear after few years of participation. These include both some general advice on how to choose and participate in the competition and some technical adv ice, how to set up your pipeline, what to do first and so on. Fi nally, we'll conclude this week by working on ensembles with Kaz Anova, the Kaggle top one. We'll start with simple linear ensemb le, then we continue with bagging and boosting, and finally we'l l cover stacking and stacked net approach. And here by the end o f this week, you should already have all required knowledge to s ucceed in a competition. And then finally, we've got the last we ek. Here we will work to analyze some of our winning solutions i n competitions. But all we are really doing in the last week is wrapping up the course, working on and submitting the final proj ect. So, this basic structure of this course. Now, we move throu gh those sections so that you can practice your skills in the co urse assignments and there are three basic types of assignments in this class: quizzes, programming assignments and the final pr oject. You don't have to do all of these in order to pass the cl ass, you only need to complete the required assignments and you can see which ones those are by looking on the course website. B ut let's go ahead and talk about the assignments. We begin with the competition. This is going to be the main assignment for you . In fact, we start working on it on the week two. There we do E DA, exploratory data analysis, set up main pipeline that you'll use for the rest of the course and check the competition for lea kages. Then in week three we update our solution by optimizing g iven metric and adding mean-encoded features. After that, in the week four, we further improve our solution by working on advanc ed features, tune your hyperparameters and uniting models in ens emble. And in last week, we all are wrapping it up and producing solution by Kaggle winning model standards. We ask you to work on the project at your local machine or your server because Cour sera computational resources are limited, and using them for the final project can slow down completing programming assignments for the fellow students. And, in fact, this class is mostly abou t this program and this competition assignment, but we also have

quizzes and programming assignments for you. We include these to give you an opportunity to refine your knowledge about specific parts of this course: how to check data for leakages, how to implement mean encodings, how to produce an ensemble and so on. You can do them at Coursera site directly but you also can download these notebooks and complete them at your local computer or your server. And this basically is an overview of the course goals, course schedule and course assignments. So, let's go ahead and get started.Hi everyone. We are starting course about machine learning competitions. In this course, you will learn a lot of tricks and best practices about data science competitions. Before we start to learn advanced techniques, we need to understand the basics. In this video, I will explain the main concept of competitions and you will become familiar with competition mechanics. A variety of machinery competition is very high. In some, participants are asked to process texts. In others, to classify picture or select the best advertising. Despite the variety, all of these competitions are very similar in structure. Usually, they consist of the same elements or concepts which we will discuss in this video. Let's start with a data. Data is what the organizers give us as training material. We will use it in order to produce our solution. Data can be represented in a variety of formats. SSV file with several columns , a text file, an archive with pictures, a database dump, a disabled code or even all together. With the data, usually there is a description. It's useful to read it in order to understand what we'll work with and which feature can be extracted. Here is an example from Kaggle. From the top, we see several files with data, and below, is their description. Sometimes in addition to data issued by organizers, we can use other data. For example, in order to improve image classification model, one may use a publicly available data set of images. But this depends on a particular competition and you need to check the rules. The next concept is a model. This is exactly what we will build during the competition. It's better to think about model not as one specific algorithm, but something that transforms data into answers. The model should have two main properties. It should produce best possible prediction and be reproducible. In fact, it can be very complicated and contain a lot of algorithms, handcrafted features, use a variety of libraries as this model of the winners of the Homesite competition shown on this slide. It's large and includes many components. But in the course, we will learn how to build such models. To compare our model with the model of other participants, we will send our predictions to the server or in other words, make the submission. Usually, you're asked about predictions only. Sources or models are not required. And also there are some exceptions, cool competitions, where participants submit their code. In this course, we'll focus on traditional challenges where a competitor submit only prediction outputs. Often, I can not just provide a so-called sample submission. An example of how the submission file should look like, look at the sample submission from the Zillow competition. In it is the first column. We must specify the ID of the object and then specify our prediction for it. This is typical format that is used in many competitions. Now, we move to the next concept, evaluation function. When you submit predictions, you need to know how good is your model. The quality of the model is de

fined by evaluation function. In essence and simply the function , the text prediction and correct answers and returns a score ch aracterizes the performance of the solution. The simplest exampl e of such a function is the accurate score. This is just a rate of correct answers. In general, there are a lot of such function s. In our course, we will carefully consider some of them. The d escription of the competition always indicates which evaluation function is used. I strongly suggest you to pay attention to thi s function because it is what we will try to optimize. But often , we are not interested in the score itself. We should only care about our relative performance in comparison to other competito rs. So we move to the last point we are considering, the leaderb oard. The leaderboard is the rate which provides you with inform ation about performance of all participating teams. Most machine learning competition platforms keep your submission history, bu t the leaderboard usually shows only your best score and positio n. They cannot as that submission score, reveal some information about data set. And, in extreme cases, one can obtain ground tr uth targets after sending a lot of submissions. In order to hand le this, the set is divided into two parts, public and private. This split is hidden from users and during the competition, we s ee the score calculated only on public subset of the data. The s econd part of data set is used for private leaderboard which is revealed after the end of the competition. Only this second part is used for final rating. Therefore, a standard competition rou tine looks like that. You as the competition, you analyze the da ta, improve model, prepare submission, send it, see leaderboard score. You repeat this action several times. All this time, only public leaderboard is available. By the end of the competition, you should select submissions which will be used for final scor ing. Usually, you are allowed to select two final submissions. C hoose wisely. Sometimes public leaderboard scores might be misle ading. After the competition deadline, public leaderboard is rev ealed, and its used for the final rating and defining the winner s. That was a brief overview of competition mechanics. Keep in m ind that many concepts can be slightly different in a particular competition. All details, for example, where they can join into teams or use external data, you will find in the rules. Strongl y suggest you to read the rules carefully before joining the com petition. Now, I want to say a few words about competition platf orms. Although Kaggle is the biggest and most famous one, there is a number of smaller platforms or even single-competition site s like KDD and VizDooM. Although this list will change over time , I believe you will find the competition which is most relevant and interesting for you. Finally, I want to tell you about the reasons to participate in data science competition. The main rea son is that competition is a great opportunity for learning. You communicate with other participants, try new approaches and get a lot of experience. Second reason is that competition often of fer you non-trivial problems and state-of-the-art approaches. It allows you to broaden the horizons and look at some everyday ta sk from a different point of view. It's also a great way to beco me recognizable, get some kind of frame inside data science comm unity and receive a nice job offer. The last reason to participa te is that you have a chance for winning some money. It shouldn' t be the main goal, just a pleasant bonus. In this video, we ana

lyzed the basic concept of the competition, talked about platfor
ms and reasons for participation. In the next video, we will tal
k about the difference between real life and competitions.[MUSIC
] Hi, everyone. In this video we'll learn
how to use Kaggle for participation in data
science competitions. Let's open kaggle.com. On the Competitions
 page, we can see
a list of currently running competitions. Every competition has
a page which
consists of title, short description, price budget, number of pa
rticipating
teams, and time before the end. Information involves all
previously running competitions, we can find if we click to All.
 Let's select some challenge and
see how it organized. Here, we see several tabs which we'll
explore, and let's start with Overview. In the Description secti
on we see
an introduction provided by organizers. In the Description, ther
e is a short
story about company and tasks, sometimes with illustration. At t
he Evaluation page, we see
the description of the target metric. In this challenge, target
metric
is the Mean Absolute Error between the logarithmic transform pre
dictions and
ground truth values. This page also contains example of sample
submission file, which is typical for such kind of competitions.
 Now let's move to the Prize page. In the Prize,
page we can find information about prizes. Take notice that in t
he title we have
information about the whole money budget, and this page,
we see how it will be split among winners. I want to highlight t
hat
in order to get money, you need not only be in top three teams,
but also beat a Zillow benchmark model. Now let's see, Timeline
page, which
contains all the information about dates. For example,
when competition starts, ends, when will the Team Merger deadlin
e and
then what month. All the details about competition,
we can find in the Rules. So we need to check really the rules.
Here we can find that team
limit is three individual, that we have maximum of five
submissions per day, that you, for example, should be at least
18 years old to participate. And that, find it, that external da
ta are not allowed. I strongly suggest you to read the rules
carefully before joining the competition. And after reading, you
 should accept it,
but I already accepted it. Now, let's check this, Data. Here we
have data provided by
the organizers, several files which we can download, and sample
submission among
them, and the description of the data. Here we have description
of files,
description of data fields, and more importantly a description
of train and test split. This is quite useful information in

order to set up right validation scheme. If you have any question about data or
other questions to ask, or insights to share, you can go to the forum,
which we can find under Discussion tab. Usually it contain a lot of topics or
threads, like Welcome, questions about validations, questions about train and
test data, and so on and so on. Every topic have title,
number of comments, and number of reports. Let's see some of them. Here we have main message,
a lot of comments, in this particular we
have only one comments. Each we can up vote or down vote and
reply to by click the reply button. That was a brief overview on forum and
now we switch to the Kernels. Usually, I run my code locally,
but sometimes it would be handy to check an idea quickly or share code
with other participants or teammates. This is what Kernels are for. You can think of Kernel as a small virtual
machine in which you write your code, execute it, and share it.
Let's take a look at some Kernel,
for example for this one. This show explanatory data analysis
on the Zillow competition. It took quite long, contain a lot of
pictures, and I believe it very useful. Here we can see comments for
this, different versions. And in order,
if you want to make a copy and edit it, we need to Fork this Notebook. It doesn't matter how your
predictions were produced, locally or by Kernel, you should submit
them through a specialized form. So go back to the competition.
Go to submissions. I already submit sample submission,
you can do the same. Click submit predictions,
and drag and drop file here. Let's look at my submission. After submission,
you will see it on the leaderboard. This is my sample submission. Leaderboard contains information
about all the teams. So here we have team name or just name
in case of single competition team. Score which we produced,
number of submissions, time since the last submissions, and
position data over seven last days. For example, this means that this guy
drops 19 positions during the last week. That was a brief overview
of Kaggle interface. Further, I will tell some extra
information about the platform. So let's move to Overview
page at the bottom. And here,
we see information about points and tiers. As mentioned here, the competition will be
counting towards ranking points an tiers. If you participate,
it will be beneficial for your rating. Sometimes, especially in educational
competitions, it's not like that. Information about Kaggle Progression
System we can find if we click this link, where we can read info

rmation about
tiers like novice, contributor, master, grandmaster. About medal
s and ranking points. This ranking points, I use for
global User Ranking. Let's check it. So, we have user ranking pa
ge, and we see all the users ranked, and
with links to their profile. Let's check some profile,
for example mine. And here we have photo,
name, some information, geo information, information about
past competitions, medals, and so on. In addition, I want to say
 a few words
about ability to host competition. Kaggle has this ability. Clic
k Host competition, and
there is special Kaggle in class. At in class, everyone can host
their own competition for free and invite people to participate.
 This option is quite often used in
various educational competitions. So this was a brief overview
of Kaggle platform. Thank for your attention. [MUSIC]In this vid
eo, I want to talk about complexity of real world machine learni
ng pipelines and how they differ from data science competitions.
 Also, we will discuss the philosophy of the competitions. Real
world machine learning problems are very complicated. They inclu
de several stages, each of them is very important and require at
tention. Let's imagine that we need to build an an anti-spam sys
tem and consider the basic steps that arise when building such a
 system. First of all, before doing any machine learning stuff,
you need to understand the problem from a business point of view
. What do you want to do? For what? How can it help your users?
Next, you need to formalize the task. What is the definition of
spam? What exactly is to be predicted? The next step is to colle
ct data. You should ask yourself, what data can we use? How to m
ine examples of spam and non-spam? Next, you need to take care o
f how to clean your data and pre-process it. After that, you nee
d to move on to building models. To do this, you need to answer
the questions, which class of model is appropriate for this part
icular task? How to measure performance? How to select the best
model? The next steps are to check the effectiveness on the mode
l in real scenario, to make sure that it works as expected and t
here was no bias introduced by learning process. Does the model
actually block spam? How often does it block non-spam emails? If
 everything is fine, then the next step is to deploy the model.
Or in other words, make it available to users. However, the proc
ess doesn't end here. Your need to monitor the model performance
 and re-train it on new data. In addition, you need to periodica
lly revise your understanding of the problem and go for the cycl
e again and again. In contrast, in competitions we have a much s
impler situation. All things about formalization and evaluation
are already done. All data collected and target metrics fixed. T
herefore your mainly focus on pre-processing the data, picking m
odels and selecting the best ones. But, sometimes you need to un
derstand the business problem in order to get insights or genera
te a new feature. Also sometimes organizers allow the usage of e
xternal data. In such cases, data collection become a crucial pa
rt of the solution. I want to show you the difference between re
al life applications and competitions more thoroughly. This tabl
e shows that competitions are much simpler than real world machi
ne learning problems. The hardest part, problem formalization an

d choice of target metric, is already done. Also questions relat ed to deploying out of scope, so participants can focus just on modeling part. One may notice that in this table data collection and model complexity roles have no and yes in competition colum n. The reason for that, that in some competitions you need to ta ke care of these things. But usually it's not the case. I want t o emphasize that as competitors, the only thing we should take c are about is target metrics value. Speed, complexity and memory consumption, all this doesn't matter as long as you're able to c alculate it and re-produce your own results. Let's highlight key points. Real world machine learning pipelines are very complica ted and consist of many stages. Competitions, add weight to a lo t of things about modeling and data analysis, but in general the y don't address the questions of formalization, deployment and t esting. Now, I want to say a few words about philosophy on compe titions, in order to form a right impression. We'll cover these ideas in more details later in the course along with examples. T he first thing I want to show you is that, machine learning comp etitions are not only about algorithms. An algorithm is just a t ool. Anybody can easily use it. You need something more to win. Insights about data are usually much more useful than a returned ensemble. Some competitions could be solved analytically, witho ut any sophisticated machine learning techniques. In this course , we will show you the importance of understanding your data, to ols to use and features you tried to exploit in order to produce the best solution. The next thing I want to say, don't limit yo urself. Keep in mind that the only thing you should care about i s target metric. It's totally fine to use heuristics or manual d ata analysis in order to construct golden feature and improve yo ur model. Besides, don't be afraid of using complex solutions, a dvance feature engineering or doing the huge gritty calculation overnights. Use all the ways you can find in order to improve yo ur model. After passing this course, you will able to get the ma ximum gain from your data. And now the important aspect is creat ivity. You need to know traditional approaches of solid machine learning problems but, you shouldn't be bounded by them. It's ok ay to modify or hack existing algorithm for your particular task . Don't be afraid to read source codes and change them, especial ly for deploying stuff. In our course, we'll show you examples o f how a little bit of creativity can lead to constructing golden features or entire approaches for solving problems. In the end, I want to say enjoy competitions. Don't be obsessed with gettin g money. Experience and fun you get are much more valuables than the price. Also, networking is another great advantage of parti cipating in data science competition. I hope you find this cours e interesting.Hi, everyone. In this video, I want to do a brief overview of basic machine learning approaches and ideas behind t hem. There are several famous of machine learning algorithms whi ch I want to review. It's a Linear Model, Tree-Based Methods, k-Nearest Neighbors, and Neural Nets. For each of this family, I w ill give a short intuitive explanation with examples. If you don 't remember any of these topics, I strongly encourage you to lea rn it using links from additional materials. Let's start with Li near Models. Imagine that we have two sets of points, gray point s belong to one class and green ones to another. It is very intu itive to separate them with a line. In this case, it's quite sim

ple to do since we have only two dimensional points. But this ap
proach can be generalized for a high dimensional space. This is
the main idea behind Linear Models. They try to separate objects
with a plane which divides space into two parts. You can rememb
er several examples from this model class like logistic regressi
on or SVM. They all are Linear Models with different loss functi
ons. I want to emphasize that Linear Models are especially good
for sparse high dimensional data. But you should keep in mind th
e limitations of Linear Models. Often, point cannot be separated
by such a simple approach. As an example, you can imagine two s
ets of points that form rings, one inside the other. Although it
's pretty obvious how to separate them, Linear Models are not an
appropriate choice either and will fail in this case. You can f
ind implementations of Linear Models in almost every machine lea
rning library. Most known implementation in Scikit-Learn library
. Another implementation which deserves our attention is Vowpal
Wabbit, because it is designed to handle really large data sets.
We're finished with Linear Model here and move on to the next f
amily, Tree-Based Methods. Tree-Based Methods use decision tree
as a basic block for building more complicated models. Let's con
sider an example of how decision tree works. Imagine that we hav
e two sets of points similar to a linear case. Let's separate on
e class from the other by a line parallel to the one of the axes
. We use such restrictions as it significantly reduces the numbe
r of possible lines and allows us to describe the line in a simp
le way. After setting the split as shown at that picture, we wil
l get two sub spaces, upper will have probability of gray=1, and
lower will have probability of gray=0.2. Upper sub-space doesn'
t require any further splitting. Let's continue splitting for th
e lower sub-space. Now, we have zero probability on gray for the
left sub-space and one for the right. This was a brief overview
of how decision tree works. It uses divide-and-conquer approach
to recur sub-split spaces into sub-spaces. Intuitively, single
decision tree can be imagined as dividing space into boxes and a
pproximating data with a constant inside of these boxes. The way
of true axis splits and corresponding constants produces severa
l approaches for building decision trees. Moreover, such trees c
an be combined together in a lot of ways. All this leads to a wi
de variety of tree-based algorithms, most famous of them being r
andom forest and Gradient Boosted Decision Trees. In case if you
don't know what are that, I strongly encourage you to remember
these topics using links from additional materials. In general,
tree-based models are very powerful and can be a good default me
thod for tabular data. In almost every competitions, winners use
this approach. But keep in mind that for Tree-Based Methods, it
's hard to capture linear dependencies since it requires a lot o
f splits. We can imagine two sets of points which can be separat
ed with a line. In this case, we need to grow a tree with a lot
of splits in order to separate points. Even in such case, our tr
ee could be inaccurate near decision border, as shown on the pic
ture. Similar to Linear Models, you can find implementations of
tree-based models in almost every machine learning library. Scik
it-Learn contains quite good implementation of random forest whi
ch I personally prefer. All the Scikit-Learn contain implementat
ion of gradient boost decision trees. I prefer to use libraries
like XGBoost and LightGBM for their higher speed and accuracy. S

o, here we end the overview of Tree-Based Methods and move on to the k-NN. Before I start the explanation, I want to say that k-NN is abbreviation for k-Nearest Neighbors. One shouldn't mix it up with Neural Networks. So, let's take a look at the familiar binary classification problem. Imagine that we need to predict label for the points shown with question mark at this slide. We assume that points close to each other are likely to have similar labels. So, we need to find the closest point which displayed by arrow and pick its label as an answer. This is how nearest neighbor's method generally works. It can be easily generalized for k-NN, if we will find k-nearest objects and select plus labeled by majority vote. The intuition behind k-NN is very simple. Closer objects will likely to have same labels. In this particular example, we use square distance to find the closest object. In general case, it can be meaningless to use such a distance function. For example, square distance over images is unable to capture semantic meaning. Despite simplicity of the approach, features based on nearest neighbors are often very informative. We will discuss them in more details later in our course. Implementations of k-NN can be found in a lot of machine learning libraries. I suggest you to use implementation from Scikit-Learn since it use algorithm matrix to speedup recollections and allows you to use several predefined distance functions. Also, it allows you to implement your own distance function. The next big class of model I want to overview is Neural Networks. Neural Nets is a special class of machine learning models, which deserve a separate topic. In general, such methods can be seen in this Black-Box which produce a smooth separating curve in contrast to decision trees. I encourage you to visit TensorFlow playground which is shown on the slide, and play with different parameters of the simple feed-forward network in order to get some intuition about how feed-forward Neural Nets works. Some types of Neural Nets are especially good for images, sounds, text, and sequences. We won't cover details of Neural Nets in this course. Since Neural Nets attracted a lot of attention over the last few years, there are a lot of frameworks to work with them. Packages like TensorFlow, Keras, MXNet, PyTorch, and Lasagne can be used to feed Neural Nets. I personally prefer PyTorch since it's provides flexible and user-friendly way to define complex networks. After this brief recap, I want to say a few words about No Free Lunch Theorem. Basically, No Free Lunch Theorem states that there is no methods which outperform all others on all tasks, or in other words, for every method, we can construct a task for which this particular method will not be the best. The reason for that is that every method relies on some assumptions about data or task. If these assumptions fail, Limited will perform poorly. For us, this means that we cannot every competition with just a single algorithm. So we need to have a variety of tools based off different assumptions. Before the end of this video, I want to show you an example from Scikit-Learn library, which plots decision surfaces for different classifiers. We can see the type of algorithm have a significant influence of decision boundaries and consequently on [inaudible]. I strongly suggest you to dive deeper into this example and make sure that you have intuition why these classifiers produce such surfaces. In the end, I want to remind you the main points of this video. First of all, there is no silver bullet algo

rithm which outperforms all the other in all and every task. Nex
t, is that Linear Model can be imagined as splitting space into
two sub-spaces separated by a hyper plane. Tree-Based Methods sp
lit space into boxes and use constant the predictions in every b
ox. k-NN methods are based on the assumptions that close objects
 are likely to have same labels. So we need to find closest obje
cts and pick their labels. Also, k-NN approach heavily relies on
 how to measure point closeness. Feed-forward Neural Nets are ha
rder to interpret but they produce smooth non-linear decision bo
undary. The most powerful methods are Gradient Boosted Decision
Trees and Neural Networks. But we shouldn't underestimate Linear
 Models and k-NN because sometimes, they may be better. We will
show you relevant examples later in our course. Thank you for yo
ur attention.Hi, everyone. In this video, I want to do an overvi
ew
of hardware and software requirements. You will know what is typ
ical stuff for
data science competitions. I want to start from
hardware related things. Participating in competitions, you gene
rally don't need a lot
of computation resources. A lot of competitions, except imaged b
ased,
have under several gigabytes of data. It's not very huge and can
 be processed on
a high level laptop with 16 gigabyte ram and four physical cores
. Quite a good setup is a tower
PC with 32 gigabyte of ram and six physical cores,
this is what I personally use. You have a choice of hardware to
use. I suggest you to pay attention
to the following things. First is RAM, for this more is better.
If you can keep your data in memory,
your life will be much, much easier. Personally, I found 64
gigabytes is quite enough, but some programmers prefer to have
128 gigabytes or even more. Next are cores, the more core you ha
ve
the more or faster experiments you can do. I find it comfortable
 to
work with fixed cores, but sometimes even 32 are not enough. Nex
t thing to pay attention for
is storage. If you work with large datasets
that don't fit into the memory, it's crucial to have fast disk t
o read and
write chunks of data. SSD is especially important if you train
narrowness or large number of images. In case you really need
computational resources. For example, if you are part of team or
 have a computational heavy approach,
you can rent it on cloud platforms. They offer machines with a l
ot of RAMs,
cores, and GPUs. There are several cloud providers, most famous
are Amazon AWS,
Microsoft's Azure, and Google Cloud. Each one has its own pricin
g, so we can choose which one best
fits your needs and budget. I especially want to draw your
attention to AWS spot option. Spot instances enable you
to be able to use instance, which can lower your cost significan
tly. The higher your price for

spot instance is set by Amazon and fluctuates depending on suppl
y and
demand for spot instances. Your spot instance run whenever you
bid exceeds the current market price. Generally, it's much
cheaper than other options. But you always have risk that your b
id
will get under current market price, and your source will be ter
minated. Tutorials about how to setup and configure cloud resour
ces you may
find in additional materials. Another important thing I
want to discuss is software. Usually, rules in competitions
prohibit to use commercial software, since it requires to buy
a license to reproduce results. Some competitors prefer
R as basic language. But we will describe Python's tech
as more common and more general. Python is quite a good language
 for
fast prototyping. It has a huge amount of high quality and
open source libraries. And I want to reuse several of them. Let'
s start with NumPy. It's a linear algebra library
to work with dimensional arrays, which contains useful linear al
gebra
routines and random number capabilities. Pandas is a library pro
viding fast,
flexible, and expressive way to work with a relational or table
of data,
both easily and intuitive. It allows you to process your
data in a way similar to SQL. Scikit-learn is a library of class
ic
machine learning algorithms. It features various classification,
regression, and clustering algorithms, including support virtual
 machines,
random force, and a lot more. Matplotlib is a plotting library.
It allows you to do
a variety of visualization, like line plots, histograms,
scatter plots and a lot more. As IDE, I suggest you to use
IPython with Jupyter node box, since they allow you to work
interactively and remotely. The last property is especially
useful if you use cloud resources. Additional packages contain
implementation of more specific tools. Usually, single packages
implement single algorithm. XGBoost and LightGBM packages implem
ent
gradient-boosted decision trees in a very efficient and optimize
d way. You definitely should
know about such tools. Keras is a user-friendly framework for
neural nets. This new package is an efficient
implementation of this new]projection method which we will
discuss in our course. Also, I want to say a few words about
external tools which usually don't have any connection despite,
but
still very used for computations. One such tool is Vowpal Wabbit
. It is a tool designed to
provide blazing speed and handle really large data sets,
which don't fit into memory. Libfm and libffm implement differen
t
types of optimization machines, and often used for sparse data l
ike

click-through rate prediction. Rgf is an alternative base method
,
which I suggest you to use in ensembles. You can install these p
ackages one by one. But as alternative, you can use byte and dis
tribution like Anaconda, which already
contains a lot of mentioned packages. And then, through this vid
eo, I want to emphasize the proposed setup
is the most common but not the only one. Don't overestimate the
role of hardware
and software, since they are just tools. Thank you for your atte
ntion. [MUSIC][NOISE]
Hi. In every competition,
we need to pre-process given data set and generate new features
from existing ones. This is often required to stay on
the same track with other competitors and sometimes careful feat
ure
preprocessing and efficient engineering can give you
the edge you strive into achieve. Thus, in the next videos, we w
ill cover
a very useful topic of basic feature preprocessing and basic fea
ture generation
for different types of features. Namely, we will go through nume
ric
features, categorical features, datetime features and coordinate
 features. And in the last video,
we will discus mission values. Beside that, we also will discus
dependence of preprocessing and generation on a model we're goin
g to use. So the broad goal of the next
videos is to help you acquire these highly required skills. To g
et an idea of following topics, let's
start with an example of data similar to what we may encounter i
n competition. And take a look at well
known Titanic dataset. It stores the data about people who were
on the Titanic liner during its last trip. Here we have a typica
l dataframe
to work with in competitions. Each row represents a person and
each column is a feature. We have different kinds of features he
re. For example, the values in
Survived column are either 0 or 1. The feature is binary. And by
 the way, it is what we
need to predict in this task. It is our target. So, age and fare
 are numeric features. Sibims p and parch accounts statement and
embarked a categorical features. Ticket is just an ID and name i
s text. So indeed,
we have different feature types here, but do we understand why w
e should care about
different features having different types? Well, there are two m
ain reasons for it, namely, strong connection between
preprocessing at our model and common feature generation methods
 for
each feature type. First, let's discuss
feature preprocessing. Most of times, we can just take our
features, fit our favorite model and expect it to get great resu
lts. Each type of feature has its own ways
to be preprocessed in order to improve quality of the model. In
other words,

joys of preprocessing matter, depends on the model we're going t
o use. For example, let's suppose that target has nonlinear
dependency on the pclass feature. Pclass linear of 1 usually lea
ds
to target of 1, 2 leads to 0, and 3 leads to 1 again. Clearly, b
ecause this is not
a linear dependency linear model, one get a good result here. So
 in order to improve
a linear model's quality, we would want to preprocess
pclass feature in some way. For example, with the so-called whic
h
will replace our feature with three, one for each of pclass valu
es. The linear model will fit much better
now than in the previous case. However, random forest does not r
equire
this feature to be transformed at all. Random forest can easily
put
each pclass in separately and predict fine probabilities. So, th
at was an example of preprocessing. The second reason why we sho
uld be
aware of different feature text is to ease generation of new fea
tures. Feature types different in this and comprehends in common
feature generation methods. While gaining an ability to
improve your model through them. Also understanding of basics of
 feature
generation will aid you greatly in upcoming advanced feature
topics from our course. As in the first point, understanding of
a model here can
help us to create useful features. Let me show you an example. S
ay, we have to predict the number of
apples a shop will sell each day next week and we already have a
 couple of months
sales history as train in data. Let's consider that we have an o
bvious
linear trend through out the data and we want to inform the mode
l about it. To provide you a visual example, we prepare the seco
nd table with last
days from train and first days from test. One way to help module
neutralize linear train is to add feature indicating
the week number past. With this feature, linear model can succes
sfully find
an existing lineer and dependency. On the other hand,
a gradient boosted decision tree will use this feature to calcul
ate something
like mean target value for each week. Here, I calculated mean va
lues manually
and printed them in the dataframe. We're going to predict number
of apples for the sixth week. node that we indeed have here. So
let's plot how a gradient
within the decision tree will complete the weak feature. As we d
o not train Gradient goosting
decision tree on the sixth week, it will not put splits
between the fifth and the sixth weeks, then,
when we will bring the numbers for the 6th week, the model will
end up
using the wave from the 5th week. As we can see unfortunately,

no users shall land their train here. And vise versa,
we can come up with an example of generated feature that will be beneficial for decisions three. And useful spoliniar model. So this example shows us,
that our approach to feature generation should rely on
understanding of employed model. To summarize this feature,
first feature preprocessing is necessary instrument you have to use to
adapt data to your model.` Second, feature generation is a very powerful technique which can aid you significantly in competitions and
sometimes provide you the required edge. And at last,
both feature preprocessing and feature generation depend on
the model you are going to use. So these three topics,
in connection to feature types, will be general theme of the next videos. We will thoroughly examine
most frequent methods which you can be able to
incorporate in your solutions. Good luck. [SOUND] [MUSIC]Hi. In this video, we will cover basic approach as to feature preprocessing and feature generation for numeric features. We will understand how model choice impacts feature preprocessing. We will identify the preprocessing methods that are used most often, and we will discuss feature generation and go through several examples. Let's start with preprocessing. First thing you need to know about handling numeric features is that there are models which do and don't depend on feature scale. For now, we will broadly divide all models into tree-based models and non-tree-based models. For example, decision trees classifier tries to find the most useful split for each feature, and it won't change its behavior and its predictions. It can multiply the feature by a constant and to retrain the model. On the other side, there are models which depend on these kind of transformations. The model based on your nearest neighbors, linear models, and neural network. Let's consider the following example. We have a binary classification test with two features. The object in the picture belong to different classes. The red circle to class zero, and the blue cross to class one, and finally, the class of the green object is unknown. Here, we will use a one nearest neighbor's model to predict the class of the green object. We will measure distance using square distance, which is also called altometric. Now, if we calculate distances to the red circle and to the blue cross, we will see that our model will predict class one for the green object because the blue cross of class one is much closer than the red circle. But if we multiply the first feature by 10, the red circle will became the closest object, and we will get an opposite prediction. Let's now consider two extreme cases. What will happen if we multiply the first feature by zero and by one million? If the feature is multiplied by zero, then every object will have feature relay of zero, which results in KNN ignoring that feature. On the opposite, if the feature is multiplied by one million, slightest differences in that features values will impact prediction, and this will result in KNN favoring that feature over all others. Great, but what about other models? Linear models are also experiencing difficulties with differently scaled features. First, we want regularization to be applied to linear models coefficients for features in equal amount. But in fact, regulariza

tion impact turns out to be proportional to feature scale. And s
econd, gradient descent methods can go crazy without a proper sc
aling. Due to the same reasons, neural networks are similar to l
inear models in the requirements for feature preprocessing. It i
s important to understand that different features scalings resul
t in different models quality. In this sense, it is just another
hyper parameter you need to optimize. The easiest way to do thi
s is to rescale all features to the same scale. For example, to
make the minimum of a feature equal to zero and the maximum equa
l to one, you can achieve this in two steps. First, we sector at
minimum value. And second, we divide the difference base maximu
m. It can be done with MinMaxScaler from sklearn. Let's illustra
te this with an example. We apply the so-called MinMaxScaler to
two features from the detaining dataset, Age and SibSp. Looking
at histograms, we see that the features have different scale, ag
es between zero and 80, while SibSp is between zero and 8. Let's
apply MinMaxScaling and see what it will do. Indeed, we see tha
t after this transformation, both age and SibSp features were su
ccessfully converted to the same value range of 0,1. Note that d
istributions of values which we observe from the histograms didn
't change. To give you another example, we can apply a scalar na
med StandardScaler in sklearn, which basically first subtract me
an value from the feature, and then divides the result by featur
e standard deviation. In this way, we'll get standardized distri
bution, with a mean of zero and standard deviation of one. After
either of MinMaxScaling or StandardScaling transformations, fea
tures impacts on non-tree-based models will be roughly similar.
Even more, if you want to use KNN, we can go one step ahead and
recall that the bigger feature is, the more important it will be
for KNN. So, we can optimize scaling parameter to boost feature
s which seems to be more important for us and see if this helps.
When we work with linear models, there is another important mom
ent that influences model training results. I'm talking about ou
tiers. For example, in this plot, we have one feature, X, and a
target variable, Y. If you fit a simple linear model, its predic
tions can look just like the red line. But if you do have one ou
tlier with X feature equal to some huge value, predictions of th
e linear model will look more like the purple line. The same hol
ds, not only for features values, but also for target values. Fo
r example, let's imagine we have a model trained on the data wit
h target values between zero and one. Let's think what happens i
f we add a new sample in the training data with a target value o
f 1,000. When we retrain the model, the model will predict abnor
mally high values. Obviously, we have to fix this somehow. To pr
otect linear models from outliers, we can clip features values b
etween two chosen values of lower bound and upper bound. We can
choose them as some percentiles of that feature. For example, fi
rst and 99s percentiles. This procedure of clipping is well-know
n in financial data and it is called winsorization. Let's take a
look at this histogram for an example. We see that the majority
of feature values are between zero and 400. But there is a numb
er of outliers with values around -1,000. They can make life a l
ot harder for our nice and simple linear model. Let's clip this
feature's value range and to do so, first, we will calculate low
er bound and upper bound values as features values at first and
99s percentiles. After we clip the features values, we can see t

hat features distribution looks fine, and we hope now this featu re will be more useful for our model. Another effective preproce ssing for numeric features is the rank transformation. Basically , it sets spaces between proper assorted values to be equal. Thi s transformation, for example, can be a better option than MinMa xScaler if we have outliers, because rank transformation will mo ve the outliers closer to other objects. Let's understand rank u sing this example. If we apply a rank to the source of array, it will just change values to their indices. Now, if we apply a ra nk to the not-sorted array, it will sort this array, define mapp ing between values and indices in this source of array, and appl y this mapping to the initial array. Linear models, KNN, and neu ral networks can benefit from this kind of transformation if we have no time to handle outliers manually. Rank can be imported a s a random data function from scipy. One more important note abo ut the rank transformation is that to apply to the test data, yo u need to store the creative mapping from features values to the ir rank values. Or alternatively, you can concatenate, train, an d test data before applying the rank transformation. There is on e more example of numeric features preprocessing which often hel ps non-tree-based models and especially neural networks. You can apply log transformation through your data, or there's another possibility. You can extract a square root of the data. Both the se transformations can be useful because they drive too big valu es closer to the features' average value. Along with this, the v alues near zero are becoming a bit more distinguishable. Despite the simplicity, one of these transformations can improve your n eural network's results significantly. Another important moment which holds true for all preprocessings is that sometimes, it is beneficial to train a model on concatenated data frames produce d by different preprocessings, or to mix models training differe ntly-preprocessed data. Again, linear models, KNN, and neural ne tworks can benefit hugely from this. To this end, we have discus sed numeric feature preprocessing, how model choice impacts feat ure preprocessing, and what are the most commonly used preproces sing methods. Let's now move on to feature generation. Feature g eneration is a process of creating new features using knowledge about the features and the task. It helps us by making model tra ining more simple and effective. Sometimes, we can engineer thes e features using prior knowledge and logic. Sometimes we have to dig into the data, create and check hypothesis, and use this de rived knowledge and our intuition to derive new features. Here, we will discuss feature generation with prior knowledge, but as it turns out, an ability to dig into the data and derive insight s is what makes a good competitor a great one. We will thoroughl y analyze and illustrate this skill in the next lessons on explo ratory data analysis. For now, let's discuss examples of feature generation for numeric features. First, let's start with a simp le one. If you have columns, Real Estate price and Real Estate s quared area in the dataset, we can quickly add one more feature, price per meter square. Easy, and this seems quite reasonable. Or, let me give you another quick example from the Forest Cover Type Prediction dataset. If we have a horizontal distance to a w ater source and the vertical difference in heights within the po int and the water source, we as well may add combined feature in dicating the direct distance to the water from this point. Among

other things, it is useful to know that adding, multiplications, divisions, and other features interactions can be of help not only for linear models. For example, although gradient within decision tree is a very powerful model, it still experiences difficulties with approximation of multiplications and divisions. And adding size features explicitly can lead to a more robust model with less amount of trees. The third example of feature generation for numeric features is also very interesting. Sometimes, if we have prices of products as a feature, we can add new feature indicating fractional part of these prices. For example, if some product costs 2.49, the fractional part of its price is 0.49. This feature can help the model utilize the differences in people's perception of these prices. Also, we can find similar patterns in tasks which require distinguishing between a human and a robot. For example, if we will have some kind of financial data like auctions, we could observe that people tend to set round numbers as prices, and there are something like 0.935, blah, blah,, blah, very long number here. Or, if we are trying to find spambots on social networks, we can be sure that no human ever read messages with an exact interval of one second. Great, these three examples should have provided you an idea that creativity and data understanding are the keys to productive feature generation. All right, let's summarize this up. In this video, we have discussed numeric features. First, the impact of feature preprocessing is different for different models. Tree-based models don't depend on scaling, while non-tree-based models usually depend on them. Second, we can treat scaling as an important hyper parameter in cases when the choice of scaling impacts predictions quality. And at last, we should remember that feature generation is powered by an understanding of the data. Remember this lesson and this knowledge will surely help you in your next competition.Hi. In this video, we will cover categorical and ordinal features. We will overview methods to work with them. In particular, what kind of pre-processing will be used for each model type of them? What is the difference between categorical and and ordinal features and how we can generate new features from them? First, let's look at several rows from the Titanic dataset and find categorical features here. Their names are: Sex, Cabin and Embarked. These are usual categorical features but there is one more special, the Pclass feature. Pclass stands for ticket class, and has three unique values: one, two, and three. It is ordinal or, in other words, order categorical feature. This basically means that it is ordered in some meaningful way. For example, if the first class was more expensive than the second, or the more the first should be more expensive than the third. We should make an important note here about differences between ordinal and numeric features. If Pclass would have been a numeric feature, we could say that the difference between first, and the second class is equal to the difference between second and the third class, but because Pclass is ordinal, we don't know which difference is bigger. As these numeric features, we can't sort and integrate an ordinal feature the other way, and expect to get similar performance. Another example for ordinal feature is a driver's license type. It's either A, B, C, or D. Or another example, level of education, kindergarten, school, undergraduate, bachelor, master, and doctoral. These categories are sorted in increasingly complex orde

r, which can prove to be useful. The simplest way to encode a ca tegorical feature is to map it's unique values to different numb ers. Usually, people referred to this procedure as label encodin g. This method works fine with two ways because tree-methods can split feature, and extract most of the useful values in categor ies on its own. Non-tree-based-models, on the other side, usuall y can't use this feature effectively. And if you want to train l inear model kNN on neural network, you need to treat a categoric al feature differently. To illustrate this, let's remember examp le we had in the beginning of this topic. What if Pclass of one usually leads to the target of one, Pclass of two leads to zero, and Pclass of three leads to one. This dependence is not linear , and linear model will be confused. And indeed, here, we can pu t linear models predictions, and see they all are around 0.5. Th is looks kind of set but three on the other side, we'll just mak e two splits select in each unique value and reaching it indepen dently. Thus, this entries could achieve much better score here using these feature. Let's take now the categorical feature and again, apply label encoding. Let this be the feature Embarked. A lthough, we didn't have to encode the previous feature Pclass be fore using it in the model. Here, we definitely need to do this with embarked. It can be achieved in several ways. First, we can apply encoding in the alphabetical or sorted order. Unique way to solve of this feature namely S, C, Q. Thus, can be encoded as two,one, three. This is called label encoder from sklearn works by default. The second way is also labeling coding but slightly different. Here, we encode a categorical feature by order of ap pearance. For example, s will change to one because it was meant first in the data. Second then c, and we will change c to two. And the last is q, which will be changed to three. This can make sense if all were sorted in some meaningful way. This is the de fault behavior of pandas.factorize function. The third method th at I will tell you about is called frequency encoding. We can en code this feature via mapping values to their frequencies. Even 30 percent for us embarked is equal to c and 50 to s and the res t 20 is equal to q. We can change this values accordingly: c to 0.3, s to 0. 5, and q to 0.2. This will preserve some informatio n about values distribution, and can help both linear and three models. first ones, can find this feature useful if value freque ncy is correlated to it's target value. While the second ones ca n help with less number of split because of the same reason. The re is another important moment about frequency encoding. If you have multiple categories with the same frequency, they won't be distinguishable in this new feature. We might a apply or run cat egorization here in order to deal with such ties. It is possible to do like this. There are other ways to do label encoding, and I definitely encourage you to be creative in constructing them. Okay. We just discussed label encoding, frequency encoding, and why this works fine for tree-based-methods. But we also have se en that linear models can struggle with label encoded feature. T he way to identify categorical features to non-tree-based-models is also quite straightforward. We need to make new code for eac h unique value in the future, and put one in the appropriate pla ce. Everything else will be zeroes. This method is called, one-h ot encoding. Let's see how it works on this quick example. So he re, for each unique value of Pclass feature, we just created a n

ew column. As I said, this works well for linear methods, kNN, o r neural networks. Furthermore, one -hot encoding feature is alr eady scaled because minimum this feature is zero, and maximum is one. Note that if you care for a fewer important numeric featur es, and hundreds of binary features are used by one-hot encoding , it could become difficult for tree-methods they use first ones efficiently. More precisely, tree-methods will slow down, not a lways improving their results. Also, it's easy to imply that if categorical feature has too many unique values, we will add too many new columns with a few non-zero values. To store these new array efficiently, we must know about sparse matrices. In a nuts hell, instead of allocating space in RAM for every element of an array, we can store only non-zero elements and thus, save a lot of memory. Going with sparse matrices makes sense if number of non-zero values is far less than half of all the values. Sparse matrices are often useful when they work with categorical featur es or text data. Most of the popular libraries can work with the se sparse matrices directly namely, XGBoost, LightGBM, sklearn, and others. After figuring out how to pre-processed categorical features for tree based and non-tree based models, we can take a quick look at feature generation. One of most useful examples o f feature generation is feature interaction between several cate gorical features. This is usually useful for non tree based mode ls namely, linear model, kNN. For example, let's hypothesize tha t target depends on both Pclass feature, and sex feature. If thi s is true, linear model could adjust its predictions for every p ossible combination of these two features, and get a better resu lt. How can we make this happen? Let's add this interaction by s imply concatenating strings from both columns and one-hot encodi ng get. Now linear model can find optimal coefficient for every interaction and improve. Simple and effective. More on features interactions will come in the following weeks especially, in adv anced features topic. Now, let's summarize this features. First, ordinal is a special case of categorical feature but with value s sorted in some meaningful order. Second, label encoding, basic ally replace this unique values of categorical features with num bers. Third, frequency encoding in this term, maps unique values to their frequencies. Fourth, label encoding and frequency enco ding are often used for tree-based methods. Fifth, One-hot encod ing is often used for non-tree-based-methods. And finally, apply ing One-hot encoding combination one heart and chords into combi nations of categorical features allows non-tree- based-models to take into consideration interactions between features, and impr ove. Fine. We just sorted out it feature pre-process for categor ical features, and took a quick look on feature generation. Now, you will be able to apply these concepts in your next competiti on and get better results.Hi. In this video, we will discuss bas ic
visual generation approaches for datetime and coordinate feature s. They both differ significantly from
numeric and categorical features. Because we can interpret the m eaning of
datetime and coordinates, we can came up with specific ideas abo ut future
generation which we'll discuss here. Now, let's start with datet ime. Datetime is quite a distinct feature

because it isn't relying on your nature, it also has several dif
ferent
tiers like year, day or week. Most new features generated from d
atetime
can be divided into two categories. The first one,
time moments in a period, and the second one,
time passed since particular event. First one is very simple. We
 can add features like second,
minute, hour, day in a week, in a month, on the year and
so on and so forth. This is useful to capture
repetitive patterns in the data. If we know about some non-commo
n
materials which influence the data, we can add them as well. For
 example, if we are to predict
efficiency of medication, but patients receive pills one
time every three days, we can consider this as
a special time period. Okay now, time seems particular event. Th
is event can be either
row-independent or row-dependent. In the first case, we just cal
culate
time passed from one general moment for all data. For example, f
rom here to thousand. Here, all samples will become pairable
between each other on one time scale. As the second variant of
time since particular event, that date will depend on the sample
we are calculating this for. For example,
if we are to predict sales in a shop, like in the ROSSMANN's
store sales competition. We can add the number of days passed
since the last holiday, weekend or since the last sales campaign
, or maybe
the number of days left to these events. So, after adding these
features,
our dataframe can look like this. Date is obviously a date, and
sales are the target of this task. While other columns
are generated features. Week day feature indicates which day
in the week is this, daynumber since year 2014 indicates how man
y days
have passed since January 1st, 2014. is_holiday is a binary feat
ure indicating
whether this day is a holiday and days_ till_ holidays indicate
how many
days are left before the closest holiday. Sometimes we have seve
ral
datetime columns in our data. The most for data here is to
subtract one feature from another. Or perhaps subtract generated
 features,
like once we have, we just have discussed. Time moment inside th
e period or
time passed in zero dependent events. One simple example of thir
d generation
can be found in churn prediction task. Basically churn predictio
n
is about estimating the likelihood that customers will churn. We
 may receive a valuable feature here
by subtracting user registration date from the date of some acti
on of his,
like purchasing a product, or calling to the customer service. W

e can see how this works
on this data dataframe. For every user, we know
last_purchase_date and last_call_date. Here we add the differenc
e between
them as new feature named date_diff. For clarity,
let's take a look at this figure. For every user, we have his
last_purchase_date and his last_call_date. Thus, we can add date
_diff
feature which indicates number of days between these events. Not
e that after generation feature is
from date time, you usually will get either numeric features lik
e
time passed since the year 2000, or categorical features like da
y of week. And these features now are need
to be treated accordingly with necessary pre-processings
we have discussed earlier. Now having discussed feature
generation for datetime, let's move onto feature generation for
coordinates. Let's imagine that we're trying to
estimate the real estate price. Like in the Deloitte competition
named
Western Australia Rental Prices, or in the Sberbank Russian Hous
ing Market
competition. Generally, you can calculate distances
to important points on the map. Keep this wonderful map. If you
have additional data with
infrastructural buildings, you can add as a feature distance to
the nearest
shop to the second by distance hospital, to the best school in t
he neighborhood and
so on. If you do not have such data, you can extract interesting
points on
the map from your trained test data. For example, you can do a n
ew
map to squares, with a grid, and within each square,
find the most expensive flat, and for every other object in this
square,
add the distance to that flat. Or you can organize your data
points into clusters, and then use centers of clusters
as such important points. Or again, another way. You can find so
me special areas,
like the area with very old buildings and add distance to this o
ne. Another major approach to use coordinates
is to calculate aggregated statistics for objects surrounding ar
ea. This can include number of lets
around this particular point, which can then be interpreted as a
reas or
polarity. Or we can add mean realty price, which will indicate h
ow expensive
area around selected point is. Both distances and aggregate stat
istics are often
useful in tasks with coordinates. One more trick you need to kno
w about
coordinates, that if you train decision trees from them, you can
add slightly
rotated coordinates is new features. And this will help a model
make

more precise selections on the map. It can be hard to know what exact
rotation we should make, so we may want to add all rotations to 45 or
22.5 degrees. Let's look at the next example
of a relative price prediction. Here the street is dividing
an area in two parts. The high priced district above the street,
and the low priced district below it. If the street is slightly rotated, trees
will try to make a lot of space here. But if we will add new coordinates in
which these two districts can be divided by a single split, this will hugely
facilitate the rebuilding process. Great, we just summarize the most
frequent methods used for future generation from datetime and coordinates. For datetime, these are applying
periodicity, calculates in time passed since particular event, and engine
differences between two datetime features. For coordinates, we should recall
extracting interesting samples from trained test data, using places from
additional data, calculating distances to centers of clusters, and adding aggregated
statistics for surrounding area. Knowing how to effectively handle datetime
and coordinates, as well as numeric and categorical features, will provide you
reliable way to improve your score. And to help you devise that
specific part of solution which is often required to beat very top scores. [SOUND]Often we have to deal with
missing values in our data. They could look like not numbers,
empty strings, or outliers like minus 999. Sometimes they can contain useful
information by themselves, like what was the reason of
missing value occurring here? How to use them effectively? How to engineer new features from them? We'll do the topic for this video. So what kind of information
missing values might contain? How can they look like? Let's take a look at missing values
in the Springfield competition. This is metrics of samples and features. People mainly reviewed each feature, and
found missing values for each column. This latest could be not a number,
empty string, minus 1, 99, and so on. For example, how can we found out
that -1 can be the missing value? We could draw a histogram and see this variable has uniform
distribution between 0 and 1. And that it has small peak of -1 values. So if there are no not numbers there, we
can assume that they were replaced by -1. Or the feature distribution plot
can look like the second figure. Note that x axis has lock scale. In this case, not a numbers probably
were few by features mean value. You can easily generalize this

logic to apply to other cases. Okay on this example we just learned this,
missing values can be hidden from us. And by hidden I mean replaced by some
other value beside not a number. Great, let's talk about
missing value importation. The most often examples are first, replacing not a number with some
value outside fixed value range. Second, replacing not
a number with mean or median. And third,
trying to reconstruct value somehow. First method is useful
in a way that it gives three possibility to take missing
value into separate category. The downside of this is that performance
of linear networks can suffer. Second method usually beneficial for
simple linear models and neural networks. But again for trees it can be harder to
select object which had missing values in the first place. Let's keep the feature value
reconstruction for now, and turn to feature generation for a moment. The concern we just have discussed can
be addressed by adding new feature isnull indicating which rows have
missing values for this feature. This can solve problems with trees and neural networks while computing mean or
median. But the downside of this is that we will
double number of columns in the data set. Now back to missing values
importation methods. The third one, and the last one we will discuss here,
is to reconstruct each value if possible. One example of such possibility is
having missing values in time series. For example,
we could have everyday temperature for a month but several values in
the middle of months are missing. Well of course, we can approximate
them using nearby observations. But obviously, this kind of
opportunity is rarely the case. In most typical scenario rows
of our data set are independent. And we usually will not find any
proper logic to reconstruct them. Great, to this moment we already learned
that we can construct new feature, isnull indicating which
rows contains not numbers. What are other important moments about
feature generation we should know? Well there's one general concern about generating new features from
one with missing values. That is, if we do this,
we should be very careful with replacing missing values
before our feature generation. To illustrate this, let's imagine we have
a year long data set with two features. Daytime feature and
temperature which had missing values. We can see all of this on the figure. Now we fill missing values with some
value, for example with median. If you have data over the whole

year
median probably will be near zero so it should look like that. Now we want to add feature like
difference between temperature today and yesterday, let's do this. As we can see, near the missing values this difference
usually will be abnormally huge. And this can be misleading our model. But hey, we already know that we can
approximate missing values sometimes here by interpolation the error by points,
great. But unfortunately, we usually don't
have enough time to be so careful here. And more importantly, these problems can occur in cases when we
can't come up with such specific solution. Let's review another example
of missing value importation. Which will be substantially
discussed later in advanced feature [INAUDIBLE] topic. Here we have a data set
with independent rows. And we want to encode the categorical
feature with the numeric feature. To achieve that we calculate mean
value of numeric feature for every category, and
replace categories with these mean values. What happens if we fill not
the numbers in the numeric feature, with some value outside of feature range like -999. As we can see, all values we will
be doing them closer to -999. And the more the row's corresponding to
particular category will have missing values. The closer mean value will be to -999. The same is true if we fill missing values
with mean or median of the feature. This kind of missing value importation
definitely can screw up the feature we are constructing. The way to handle this
particular case is to simply ignore missing values while
calculating means for each category. Again let me repeat the idea
of these two examples. You should be very careful with early none
importation if you want to generate new features. There's one more interesting
thing about missing values. [INAUDIBLE] boost can
handle a lot of numbers and sometimes using this approach
can change score drastically. Besides common approaches
we have discussed, sometimes we can treat
outliers as missing values. For example, if we have some easy classification task with songs which are thought to be composed even before
ancient Rome, or maybe the year 2025. We can try to treat these outliers as missing values. If you have categorical features, sometimes it can be beneficial
to change the missing values or categories which present in the test data
but do not present in the train data. The intention for doing so appeals to
the fact that the model which didn't have that category in the train data

will eventually treat it randomly. Here and
of categorical features can be of help. As we already discussed in our course, we
can change categories to its frequencies and thus to it categories was in
before based on their frequency. Let's walk through
the example on the slide. There you see from the categorical
feature, they not appear in the train. Let's generate new feature indicating number of where the occurrence
is in the data. We will name this feature
categorical_encoded. Value A has six occurrences
in both train and test, and that's value of new feature
related to A will be equal to 6. The same works for values B, D, or C. But now new features various related
to D and C are equal to each other. And if there is some dependence in between
target and number of occurrences for each category, our model will be
able to successfully visualize that. To conclude this video, let´s overview
main points we have discussed. The choice of method to fill not a numbers depends on the situation. Sometimes, you can
reconstruct missing values. But usually, it is easier to
replace them with value outside of feature range, like -999 or to replace them with mean or median. Also missing values already can be
replaced with something by organizers. In this case if you want know exact
rows which have missing values you can investigate this
by browsing histograms. More, the model can improve its results using binary feature isnull which indicates what roles have missing values. In general, avoid replacing missing
values before feature generation, because it can decrease
usefulness of the features. And in the end,
Xgboost can handle not a numbers directly, which sometimes can change the score for
the better. Using knowledge you have
derived from our discussion, now you should be able to
identify missing values. Describe main methods to handle them, and apply this knowledge to gain
an edge in your next computation. Try these methods in different scenarios and for sure, you will succeed.[MUSIC] Hi. Often in computations,
we have data like text and images. If you have only them, we can apply
approach specific for this type of data. For example, we can use search
engines in order to find similar text. That was the case in
the Allen AI Challenge for example. For images, on the other hand,
we can use conditional neural networks, like in the Data Science Bowl, and
a whole bunch of other competitions. But if we have text or images as additional data, we usually
must grasp different features, which can be edited as complementary to our

main data frame of samples and features. Very simple example of such case we can
see in the Titanic dataset we have called name, which is more or less like text, and to use it, we first need to derive
the useful features from it. Another most surest example,
we can predict whether a pair of online advertisements are duplicates, like
slighty different copies of each other, and we could have images from these
advertisements as complimentary data, like the Avito Duplicates Ads Detection
competition. Or you may be given the task
of classifying documents, like in the Tradeshift Text
Classification Challenge. When feature extraction is done, we can
treat extracted features differently. Sometimes we just want to add new
features to existing dataframe. Sometimes we even might want to use the
right features independently, and in end, make stake in with the base solution. We will go through stake in and we will
learn how to apply it later in the topic about ensembles, but for now, you should
know that both ways first to acquire, to of course extract features
from text and images somehow. And this is exactly what we
will discuss in this video. Let's start with featured
extraction from text. There are two main ways to do this. First is to apply bag of words,
and second, use embeddings like word to vector. Now, we'll talk about a bit
about each of these methods, and in addition, we will go through text
pre-processings related to them. Let's start with the first approach,
the simplest one, bag of words. Here we create new column for each unique word from the data, then we
simply count number of occurences for each word, and place this value
in the appropriate column. After applying the separation to each row, we will have usual dataframe
of samples and features. In a scalar,
this can be done with CountVectorizer. We also can post process calculated
metrics using some pre-defined methods. To make out why we need post-processing
let's remember that some models like kNN, like neural regression, and neural
networks, depend on scaling of features. So the main goal of post-processing here is to make samples more comparable
on one side, and on the other, boost more important features while
decreasing the scale of useless ones. One way to achieve the first goal
of making a sample small comparable is to normalize sum of values in a row. In this way, we will count not

occurrences but frequencies of words. Thus, texts of different s
izes
will be more comparable. This is the exact purpose of
term frequency transformation. To achieve the second goal,
that is to boost more important features, we'll make post proces
s our matrix
by normalizing data column wise. A good idea is to normalize eac
h feature
by the inverse fraction of documents, which contain the exact wo
rd
corresponding to this feature. In this case,
features corresponding to frequent words will be scaled down com
pared to
features corresponding to rarer words. We can further improve th
is idea
by taking a logarithm of these numberization coefficients. As a
result, this will decrease
the significance of widespread words in the dataset and
do require feature scaling. This is the purpose of inverse
document frequency transformation. General frequency, and invers
e
document frequency transformations, are often used together,
like an sklearn, in TFiDF Vectorizer. Let's apply TFiDF transfor
mation
to the previous example. First, TF. Nice. Occurences which
are switched to frequencies, that means some of variance for
each row is now equal to one. Now, IDF, great. Now data is norma
lized column wise,
and you can see, for those of you who are too excited, IDF trans
formation scaled
down the appropriate feature. It's worth mentioning that there
are plenty of other variants of TFiDF which may work better depe
nding
on the specific data. Another very useful technique is Ngrams. T
he concept of Ngram is simple, you add
not only column corresponding to the word, but also columns corr
esponding
to inconsequent words. This concept can also be applied
to sequence of chars, and in cases with low N, we'll have a colu
mn
for each possible combination of N chars. As we can see, for N =
 1, number of
these columns will be equal to 28. Let's calculate number of
these columns for N = 2. Well, it will be 28 squared. Note that
sometimes it can be cheaper
to have every possible char Ngram as a feature, instead of havin
g a feature for
each unique word from the dataset. Using char Ngrams also helps
our
model to handle unseen words. For example,
rare forms of already used words. In a scalared count vectorizor
has appropriate parameter for using Ngrams, it is called Ngram_r
ange. To change from word Ngrams to char Ngrams,
you may use parameter named analyzer. Usually, you may want to p
reprocess text,
even before applying bag of words, and sometimes, careful text p

reprocessing
can help bag of words drastically. Here, we will discuss such me thods
as converting text to lowercase, lemmatization, stemming,
and the usage of stopwords. Let's consider simple example
which shows utility of lowercase. What if we applied bag of word s
to the sentence very, very sunny? We will get three columns for
each word. So because Very, with capital letter,
is not the same string as very without it, we will get multiple columns for
the same word, and again, Sunny with capital letter
doesn't match sunny without it. So, first preprocessing what we want to
do is to apply lowercase to our text. Fortunately, configurizer from
sklearn does this by default. Now, let's move on to lemmatizatio n and
stemming. These methods refer to more
advanced preprocessing. Let's look at this example. We have two sentences: I had a car,
and We have cars. We may want to unify the words car and
cars, which are basically the same word. The same goes for had a nd have, and so on. Both stemming and lemmatization may
be used to fulfill this purpose, but they achieve this in differ ent ways. Stemming usually refers to a heuristic
process that chops off ending of words and thus unite duration o f related words
like democracy, democratic, and democratization, producing somet hing like,
democr, for each of these words. Lemmatization, on the hand, usu ally means
that you have want to do this carefully using knowledge or vocab ulary, and
morphological analogies of force, returning democracy for
each of the words below. Let's look at another example that show s
the difference between stemming and lemmatization by applying
them to word saw. While stemming will return on
the letter s, lemmatization will try to return either see or saw ,
dependent on the word's meaning. The last technique for text pre processing, which we will discuss here,
is usage of stopwords. Basically, stopwords are words which do not contain important information for our model. They are either insignificant
like articles or prepositions, or so common they do not
help to solve our task. Most languages have predefined list
of stopwords which can be found on the Internet or logged from N LTK, which stands for Natural Language
Toolkit Library for Python. CountVectorizer from sklearn also
has parameter related to stopwords, which is called max_df. max_ df is the threshold
of words we can see, after we see in which,
the word will be removed from text corpus. Good, we just have di scussed classical

feature extraction pipeline for text. At the beginning,
we may want to pre-process our text. To do so, we can apply lowe
rcase,
stemming, lemmatization, or remove stopwords. After preprocessin
g, we can use bag
of words approach to get the matrix where each row represents a
text, and
each column represents a unique word. Also, we can use bag of wo
rds approach for
Ngrams, and in new columns for groups of
several consecutive words or chars. And in the end, when we post
process
these metrics using TFiDF, which often prove to be useful. Well,
 then now we can add extracted
features to our basic data frame, or putting the dependent model
 on
them to create some tricky features. That's all for now. In the
next video, we will continue
to discuss feature extraction. We'll go through two big points.
First, we'll talk about approach for texts, and second, we will
discuss
feature extraction for images. [MUSIC]Hi and welcome back. In th
is video, we'll talk about Word2vec approach for texts and then
we'll discuss feature extraction or images. After we've summariz
ed pipeline for feature extraction with Bag of Words approach in
 the previous video, let's overview another approach, which is w
idely known as Word2vec. Just as the Bag of Words approach, we w
ant to get vector representations of words and texts, but now mo
re concise than before. Word2vec is doing exactly that. It conve
rts each word to some vector in some sophisticated space, which
usually have several hundred dimensions. To learn the word embed
ding, Word2vec uses nearby words. Basically, different words, wh
ich often are used in the same context, will be very close in th
ese vectoring representation, which, of course, will benefit our
 models. Furthermore, there are some prominent examples showing
that we can apply basic operations like addition and subtraction
 on these vectors and expect results of such operations to be in
terpretable. You should already have seen this example by now so
mewhere. Basically, if we calculate differences between the vect
ors of words queen and king, and differences between the vectors
 of words woman and man, we will find that these differences are
 very similar to each other. And, if we try to see this from ano
ther perspective, and subtract the vector of woman from the vect
or of king and then and the vector of man, will pretty much agai
n the vector of the word queen. Think about it for a moment. Thi
s is fascinating fact and indeed creation of Word2vec approach l
ed to many extensive and far reaching results in the field. Ther
e are several implementations of this embedding approach besides
 Word2vec namely Glove, which stands for Global Vector for word
representation. FastText and few others. Complications may occur
, if we need to derive vectors not for words but for sentences.
Here, we may take different approaches. For example, we can calc
ulate mean or sum of words vectors or we can choose another way
and go with special models like Doc2vec. Choice all the way to p
roceed here depends on and particular situation. Usually, it is
better to check both approaches and select the best. Training of

Word2vec can take quite a long time, and if you work with text or some common origin, you may find useful pre-trained models on the internet. For example, ones which are trained on the Wikipedia. Otherwise, remember, the training of Word2vec doesn't require target values from your text. It only requires text to extract context for each word. Note, that all pre-processing we had discussed earlier, namely lowercase stemming, lemmatization, and the usage of stopwords can be applied to text before training Word2vec models. Now, we're ready to summarize difference between Bag of Words and the Word2vec approaches in the context of competition. With Bag of Words, vectors are quite large but is a nice benefit. Meaning of each value in the vector is known. With Word2vec, vectors have relatively small length but values in a vector can be interpreted only in some cases, which sometimes can be seen as a downside. The other advantage of Word2vec is crucial in competitions, is that words with similar meaning will have similar vector representations. Usually, both Bag of Words and Word2vec approaches give quite different results and can be used together in your solution. Let's proceed to images now. Similar to Word2vec for words, convolutional neural networks can give us compressed representation for an image. Let me provide you a quick explanation. When we calculate network output for the image, beside getting output on the last layer, we also have outputs from inner layers. Here, we will call these outputs descriptors. Descriptors from later layers are better way to solve texts similar to one network was trained on. In contrary, descriptors from early layers have more text independent information. For example, if your network was trained on images and data set, you may successfully use its last layer representation in some Kar model classification text. But if you want to use your network in some medical specific text, you probably will do better if you will use an earlier for connected layer or even retrain network from scratch. Here, you may look for a pre-trained model which was trained on data similar to what you have in the exact competition. Sometimes, we can slightly tune network to receive more suitable representations using targets values associated with our images. In general, process of pre-trained model tuning is called fine-tuning. As in the previous example, when we are solving some medical specific task, we can find tune VGG RestNet or any other pre-trained network and specify it to solve these particular texts. Fine-tuning, especially for small data sets, is usually better than training standalone model on descriptors or a training network from scratch. The intuition here is pretty straightforward. On the one hand, fine-tuning is better than training standalone model on descriptors because it allows to tune all networks parameters and thus extract more effective image representations. On the other hand, fine-tuning is better than training network from scratch if we have too little data, or if the text we are solving is similar to the text model was trained on. In this case, model can you use the my knowledge already encoded in networks parameters, which can lead to better results and the faster retraining procedure. Lets discuss the most often scenario of using the fine-tuning on the online stage or the Data Science Game 2016. The task was to classify these laid photos of roofs into one of four categories. As usual, logo was first chosen to the other metric. Competitors had 8,000 different images. In this setting,

it was a good choice to modify some pre-trained network to predi
ct probabilities for these four classes and fine tune it. Let's
take a look at VGG-16 architecture because it was trained on the
1000 classes from VGG RestNet, it has output of size 1000. We h
ave only four classes in our text, so we can remove the last lay
er with size of 1000 and put in its place a new one with size of
four. Then, we just retrain our model with very smaller rate is
usually about 1000 times lesser than our initial low rate. That
is fine-tuning is done, but as we already discussed earlier in
this video, we can benefit from using model pre-trained on the s
imilar data set. Image in by itself consist of very different cl
asses from animals to cars from furniture to food could define m
ost suitable pre-trained model. We just could take model trained
on places data set with pictures of buildings and houses, fine-
tuning this model and further improve their result. If you are i
nterested in details of fine-tuning, you can find information ab
out it in almost every neural networks library namely Keras, PyT
orch, Caffe or other. Sometimes, you also want to increase numbe
r of training images to train a better network. In that case, im
age augmentation may be of help. Let's illustrate this concept o
f image augmentation. On the previous example, we discussed clas
sification of roof images. For simplicity, let's imagine that we
now have only four images one for each class. To increase the n
umber of training samples. let's start with rotating images by 1
80 degrees. Note, that after such rotation, image of class one a
gain belongs to this class because the roof on the new image als
o has North-South orientation. Easy to see that the same is true
for other classes. Great. After doing just one rotation, we alr
eady increase the amount of our trained data twice. Now, what wi
ll happen if we rotate image from the first class by 90 degrees?
What class will it belong to? Yeah, it will belong to the secon
d class and eventually, if you rotate images from the third and
the fourth classes by 90 degrees, they will stay in the same cla
ss. Look, we just increase the size of our trained set four time
s although adding such augmentations isn't so effective as addin
g brand new images to the trained set. This is still very useful
and can boost your score significantly. In general case, augmen
tation of images can include groups, rotations, and the noise an
d so on. Overall, this reduces over fitting and allows you to tr
ain more robust models with better results. One last note about
the extracting vectors from images and this note is important on
e. If you want to fine-tuning convolutiontional neural network o
r train it from scratch, you usually will need to use labels fro
m images in the trained set. So be careful with validation here
and do not over fit. Well then, let's recall main points we have
discussed here. Sometimes, you have a competition with texts or
images as additional data. In this case, usually you want to ex
tract the useful features from them to improve your model. When
you work with text, pre-processing can prove to be useful. These
pre-processing can include all lowercase, stemming, lemmatizati
on, and removing the stopwords. After that pre-processing is don
e, you can go either Bag of Words or with the Word2vec approach.
Bag of Words guarantees you clear interpretation. Each feature
are tuned by means of having a huge amount of features one for e
ach unique word. On other side, Word2vec produces relatively sma
ll vectors by meaning of each feature value can be hazy. The oth

er advantage of Word2vec that is crucial in competitions is that words with similar meaning will have similar vector representation. Also, Ngrams can be applied to include words interactions for text and TFiDF can be applied to post-process metrics produced by Bag of Words. Now images. For images, we can use pre-trained convolutional neural networks to extract the features. Depending on the similarity between the competition data and the data neural network was trained on, we may want to calculate descriptors from different layers. Often, fine-tuning of neural network can help improve quality of the descriptors. For the purpose of effective fine-tuning, we may want to augment our data. Also, fine-tuning and data augmentation are often used in competitions where we have no other date except images. Besides, there are a number of pre-trained models for convolutional neural networks and Word2vec on the internet. Great. Now, you know how to handle competitions with additional data like text and images. By applying and adapting ideas we have discussed, you will be able to gain an edge in this kind of setting.

.

.

[MUSIC] Hi, in this lesson we
will talk about the very first steps a good data scientist
takes when he is given a new data set. Mainly, exploratory data analysis or
EDA in short. By the end of this lesson, you will know,
what are the most important things from data understanding and exploration
prospective we need to pay attention to. This knowledge is required
to build good models and achieve high places on the leader board. We will first discuss what exploratory
data analysis is and why we need it. We will then go through important
parts of EDA process and see examples of what we
can discover during EDA. Next we will take a look at the tools
we have to perform exploration. What plots to draw and
what functions from pandas and matplotlib libraries can be useful for us. We will also briefly discuss a very
basic data set cleaning process that is convenient to perform
while exploring the data. And finally we'll go through
exploration process for the Springleaf competition
hosted on Kaggle some time ago. In this video we'll start talking
about Exploratory Data Analysis. What is Exploratory Data Analysis? It's basically a process of looking
into the data, understanding it and getting comfortable with it. Getting comfortable with a task,
probably always the first thing you do. To solve a problem,
you need to understand a problem, and to know what you
are given to solve it. In data science, complete data
understanding is required to generate powerful features and
to build accurate models. In fact while you explore the data,
you build an intuition about it. And when the data is intuitive
for you, you can generate hypothesis
about possible new features and eventually find some insights in the data
which in turn can lead to a better score. We will see the example of what EDA
can give us later in this lesson. Well, one may argue that
there is another way to go. Read the data from the hard drive,
never look at it and feed the classifier immediately.They use
some pretty advanced modeling techniques, like mixing, stacking, and eventually get
a pretty good score on the leaderboard. Although this approach sometimes works, it cannot take you to the very
top positions and let you win. Top solutions always use advanced and
aggressive modeling. But usually they have
something more than that. They incorporated insights from the data,
and to find those insights,
they did a careful EDA. While we need to admit the raw
computations where all you can do is modeling and EDA will not help
you to build a better model. It is usually the case when
the data is anonymized, encrypted, pre-processed, and obfuscated

. But look it will any way need to perform
EDA to realize that this is the case and you better spend more t
ime on modeling and
make a server busy for a month. One of the main EDA
tools is Visualization. When we visualize the data,
we immediately see the patterns. And with this, ask ourselves,
what are those patterns? Why do we see them? And finally, how do
 we use those
patters to build a better model? It also can be another way arou
nd. Maybe we come up with a particular
hypothesis about the data. What do we do? We test it by making a
 visualization. In one of the next videos, we'll talk about the
main visualization
tools we can use for exploration. Just as a motivation example,
I want to tell you about the competition, alexander D'yakonov, a
 former top one
at Kagel took part some time ago. The interesting thing about th
is
competition is that you do not need to do any modeling,
if you understood your data well. In that competition,
the objective was to predict whether a person will use the promo
that a company offers him. So each role correspond to a particul
ar
promo received by a person. There are features that
describe the person, for example his age, sex,
is he married or not and so on. And there are features that desc
ribe
the promo, the target is 0 or 1, will he use the promo or not. B
ut, among all the features there
were two especially interesting. The first one is, the number of
promos sent by the person before. And the second is the number o
f
promos the person had to use before. So let's take a particular
user,
say with index 13, and sort the rows by number
of promos sent column. And now let's take a look
at the difference at column the number of used promos
between two consecutive rows. It is written here in diff column.
 And look, the values in diff column in
most cases equal the target values. And in fact, there is no mag
ic. Just think about
the meaning of the columns. For example, for the second row we s
ee
that the person used one promo already but he was sent only one
before that time. And that is why we know that he used the
first promo and thus we have an answer for the first row. In gen
eral, if before the current
promo the person used n promos and before the next promo he used
 that, we know that he used n + 1 promos then we
realize that he used the current promo. And so the answer is 1.
If we know that he used n
promos before the next promo, exactly as before the current prom
o, then obviously he did not use
the current promo and the answer is 0. Well, it's not clear what
to do with the last row for every user, or when we have missing
rows,

but you see the point. We didn't even run the classifier,
and we have 80% accuracy already. This would not be possible if we didn't
do an EDA and didn't look into the data. Also as a remark, I should say
that the presented method works because of mistake made by
the organizers during data preparation. These mistakes are called leaks, and in competitions we are usually
allowed to exploit them. We'll see more of these
examples later in this course. So in this video we discussed the main
reasons for performing an EDA. That is to get comfortable with the data
and to find insights in magic features. We also saw an example where EDA and the data understanding was
important to get a better score. And finally, the point to take away. When you start a competition,
you better start with EDA, and not with hardcore modelling. We've had a lot of things to
talk about in this lesson. So let´s move to the next video. [MUSIC]In this video, we'll go through and break down several important steps namely, the first, getting domain knowledge step, second, checking if data is intuitive, and finally, understanding how the data was generated. So let's start with the first step, getting the domain knowledge. If we take a look at the computations hosted in the Kaggle, well, you'll notice, they are rather diverse. Sometimes, we need to detect threats on three dimensional body scans, or predict real estate price, or classify satellite images. Computation can be on a very specific topic which we know almost nothing about, that is, we don't have a domain knowledge. Usually, we don't need to go too deep inside the field but it's preferable to understand what our aim is, what data we have, and how people usually tackle this kind of problems to build a baseline. So, our first step should probably be searching for the topic, Googling within Wikipedia, and making sure we understand the data. For example, let's say we start a new computation in which we need to predict advertisers cost. Our first step is to realize that the computation is about web advertisement. By looking and searching for the column names, using any search engine, we understand that the data was exported from Google AdWords system. And after reading several articles about Google AdWords, we get the meaning of the columns. We now know that impressions column contained a number of times a particular ad appeared before users, and clicks column is how many times the ad was clicked by the users, and of course, the number of clicks should be less or equal than the number of impression. In this video, we'll not go much further into the details about this data set, but you can open the supplementary reading material for a more detailed exploration. After we've learned some domain knowledge, it is necessary to check if the values in the data set are intuitive, and agree with our domain knowledge. For example, if there is a column with age data, we should expect the values rarely to be larger than 100. And for sure, no one ever lived more than 200 years. So, the values should be smaller than 200. But for some reason, we find this super huge value 336. Most probably, is just a typo but it should be 36 or 33, and the best we can do is manuall

y correct it. But the other possibility is that its not a human age, but some alien's age for which it's totally normal to live more than 300 years. To check that, we should probably read the data description one more time, ask on forums. Maybe the data is totally correct, and then we just misinterpret it. Now, take a look at our Google AdWords data set. We understood that the values in the clicks variable should be less or equal than the values in impressions column. And in our case, in the first row, we see zero impressions and three clicker. That sounds like a bug, right? In fact, it probably is, but differently to the example of person's age, it could be rather a regular error made by either data exporting script or another kind of algorithm. That is, the errors were made not at random, but there is some kind of logic why there is an error in that particular place. It means that these mistakes can be used to get a better score. For example, in our case, we could create a new feature, is_incorrect, and mark all the rows that have errors. Probably, our models will find this feature helpful. It is also very important to understand how the data was generated. What was the algorithm for sampling objects from the database? Maybe, the host sample get objects at random, or they over-sample the particular class, that is, they generated more examples of that class. For example, to make the data set more class balanced. In fact, only if you know how the data was generated, you can set up a proper validation scheme for models. Coming down for the correct validation pipeline is crucial, and we will discuss it later in this course. So, what can we possibly find out about generation processes? For example, we could find out the train and test set were generated with different algorithms. And if the test set is different to the train set, we cannot use part of the train set as a validation set, because this part will not be representative of test set. And so, we cannot evaluate our models using it. So once again, to set up a correct validation, we need to know underlying data generation processes. In the ad computation, we've discussed before, that all the symptoms of different train test sampling. Improving the model on validation set didn't result into improved public leader-board score. And more, the leader-board score was unexpectedly higher than the validation score. I was also visualizing various things while trying to understand what's happening, and every time, the plots for the train set were much different to the test set plots. This also could not happen if the train and test set were similar. And finally, it was suspicious that although the train period was more than ten times larger than the test period, the train set had much fewer rows. it was not straight forward, but this triangle on the left figure was the clue for me, and the puzzle was solved. I've adjusted the train set to match test set. The validation score became reliable, and the modeling could be commenced. You can find the entire task description and investigation in the written materials. So, in this video, we've discussed several important exploratory steps. First, we need to get domain knowledge about the task as it helps to better understand the problem and the data. Next, we need to check if the data is intuitive, and agrees with our domain knowledge. And finally, it is necessary to understand how the data was generated by organizers because otherwise, we cannot establish a proper validation for our models.[SOUND] In the previous video,

we were working with the data for which we had a nice descriptio
n. That is,
we knew what the features were, and the data was given us as the
se
without severe modifications. But, it's not always the case. The
 data can be anonymized,
and obfuscated. In this video, we'll first discuss
what is anonymized data, and why organizers decide to
anonymize their data. And next, we will see what we
as competitors can do about it. Sometimes we can decode the data
, or if we can not we can try to guess,
what is the type of feature. So, let's get to the discussion. So
metimes the organizers really want
some information to be reviewed. So, they make an effort to
export competition data, in a way one couldn't get
while you're out of it. Yet all the features are preserved, and
machinery model will be
able to do it's job. For example, if a company wants
someone to classify its document, but doesn't want to reveal
the document's content. It can replace all the word occurrences
with hash values of those words, like in the example you see her
e. In fact, it will not change a thing for
a model based on bags of words. I will refer to Anonymized
data as to any data which organizers intentionally changed. Alth
ough it is not completely correct,
I will use this wording for any type of changes. In computations
 with tabular data, companies can try to hide
information each column stores. Take a look at this data set. Fi
rst, we don't have any
meaningful names for the features. The names are replaced with s
ome dummies,
and we see some hash like values
in columns x1 and x6. Most likely, organizers decided
to hash some sensitive data. There are several things we can do
while exploring the data in this case. First, we can try to deco
de or de-anonymize the data,
in a legal way of course. That is, we can try to guess
true meaning of the features. Sometimes de-anonymization
is not possible, but what we almost surely can do,
is to guess the type of the features, separating them into numer
ic,
categorical, and so on. Then, we can try to find how
features relate to each other. That can be a specific relation
between a pair of features, or we can try to figure out if
the features are grouped in some way. In this video we will conc
entrate
on the first problem. In the next video we will discuss
visualization tools, that we can use both for exploring individu
al
features, and feature relations. Let's now get to an example
how it was possible to decode the meaning of the feature in one
local competition I took part. I want to tell you about
a competition I took part. It was a local competition, and
organizers literally didn't give competitors any information
about a dataset. They just put the link to download data on
the competition page, and nothing else. Let's read the data firs

t, and basically what we see here is
that the data is anonymized. The column names are like x somethi
ng, and the values are hashes, and
then the rest are numeric in here. But, well we don't know
what they mean at all, and basically we don't
what we are to predict. We only know that it is
a multi-class classification task, and we have four labels. So,
as long as we don't
know what the data is, we can probably build a quick baseline. L
et's import Random Forest Classifier. Yeah, of course we need to
 drop
target label from our data frame, as it is included in there. We
'll fill null values with minus 999, and let's encode all the ca
tegorical features, that we can find by looking at the types. Pr
operty of our data frame. We will encode them with Label Encoder
,
and it is easier to do with
function factorize from Pandas. Let's feed to
Random Forest Classifier on our data. And let's plot the feature
 importance's,
and what we see here is that feature
X8 looks like an interesting one. We should probably investigate
it a little bit deeper. If we take the feature X8, and print it'
s mean, and estimate the value. They turn out to be quite close
to 0,
and 1 respectively, and it looks like this feature was
tendered skilled by the organizers. And we don't see here exactl
y 0,
and exactly 1, because probably training test was
concatenated when on the latest scale. If we concatenate trainin
g test,
then the mean will be exactly 0, and the std will be exactly 1.
Okay, so let's also see are there any
other repeated values in these features? We can do it with a val
ue counts function. Let's print first 15
rows of value counts out. And we can see that there
are a lot of repeated values, they repeated a thousand times. Al
l right, so we now know that
this feature was standard scaled. Probably, we can try to scale
it back. The original feature was multiplied by
a number, and was shifted by a number. All we need to do is to f
ind the shooting
parameter, and the scaling parameter. But how do we do that,
and it is really possible? Let's take unique values of the featu
re,
and sort them. And let's print the difference
between two consecutive numbers, in this sorted array. And look,
 it looks like the values
are the same all the time. The distance between two consecutive
unique values in this feature, was the same in the original data
 to. It was probably not 0.043 something, it was who knows,
it could be 9 or 11 or 11.7, but it was the same between all the
 pairs,
so assume that it was 1 because, well,
1 looks like a natural choice. Let's divide our feature by
this number 0.043 something, and if we do it, yes, we see that

the differences become rather close to 1, they are not 1,
only because of some numeric errors. So yes, if we divide our fe
ature by
this value, this is what you get. All right, so what else do we
see here. We see that each number,
it ends with the same values. Each positive number ends
with this kind of value, and each negative with this, look. It l
ooks like this fractional
part was a part of the shifting parameter,
let's just subtract it. And in fact if we subtract it,
the data looks like an integers, actually. Like it was integer d
ata, but
again because of numeric errors, we see some weird numbers in he
re. Let's round the numbers,
and that is what we get. This is actually on the first ten rows,
not the whole feature. Okay, so what's next? What did we do so f
ar? We found the scaling parameter,
probably we were right, because the numbers became integers,
and it's a good sign. We could be not right, because who knows,
the scaling parameter could be 10 or 2 or again 11 and
still the numbers will be integers. But, 1 looks like a good mat
ch. It couldn't be as random, I guess. But, how can we find
the shifting parameter? We found only fractional part,
can we find the other, and can we find the integer part, I mean?
 It's actually a hard question, because
while you have a bunch of numbers in here, and you can probably
build a hypothesis. It could be something, and the regular
values for this something is like that, and we could probably sc
ale it,
shift it by this number. But it could be only an approximation,
and not a hypothesis, and so our journey could
really end up in here. But I was really lucky, and I will show i
t to you,
so if you take your x8. I mean our feature, and
print value counts, what we will see, we will this number 11, 17
, 18, something. And then if we scroll down
we will see this, -1968, and it definitely looks like
year a of birth, right? Immediately I have a hypothesis, that th
is could be a text box where
a person should enter his year of birth. And while most of the p
eople really
enter their year of birth, but one person entered zero. Or syste
m automatically entered 0,
when something wrong happened. And wow, that isn't the key. If w
e assume the value was originally 0, then the shifting parameter
 is
exactly 9068, let's try it. Let's add 9068 to our data,
and see the values. Again we will use value counts function,
and we will sort sorted values. This is the minimum of the value
s,
and in fact you see the minimum is 0, and all the values are not
 negative,
and it looks really plausible. Take a look, 999,
it's probably what people love to enter when they're asked to en
ter something,
or this, 1899. It could be a default value for

this textbook, it occurred so many times. And then we see some w
eird values in here. People just put them at random. And then, w
e see some kind of
distribution over the dates. That are plausible for
people who live now, like 1980. Well maybe 1938, I'm not sure ab
out this, and yes of course we see some
days from the future, but for sure it looks like a year of birth
, right? Well the question, how can we use
this information for the competition? Well again for linear mode
ls, you probably could make a new feature
like age group, or something like that. But In this particular c
ompetition, it was no way to use this for,
to use this knowledge. But, it was really fun to investigate. I
hope you liked the example,
but usually is really hard to recognize anything sensible like
a year of birth anonymous features. The best we can do is to
recognize the type of the feature. Is it categorical, numeric,
text, or something else? Last week we saw that each data
type should be treated differently, and more treatment depends
on the model we want to use. That is why to make a stronger mode
l, we
should know what data we are working with. Even though we cannot
 understand
what the features are about, we should at least detect the types
of variables in the data. Take a look at this example, we don't
have any meaningful companies, but still we can deduce what
the feature types are. So, x1 looks like text or
physical recorded, x2 and x3 are binary, x4 is numeric,
x5 is either categorical or numeric. And more, if it's numeric i
t could
be something like event calendars, because the values are intege
rs. When the number of columns in data
set is small, like in our example, we can just bring the table,
and
manually sort the types out. But, what if there are thousand
of features in the data set? Very useful functions to
facilitate our exploration, function d types from pandas guesses
 the
types for each column in the data frame. Usually it groups all t
he columns
into three categories, flawed, integer, and
so called object type. If dtype function assigned
flawed type to a feature, this feature is most likely to be nume
ric. Integer typed features can be either
binary encoded with a zero or one. Event counters, or even categ
orical,
encoded with the label encoder. Sometimes this function
returns a type named object. And it's the most problematic,
it can be anything, even an irregular numeric feature with
missing values filled with some text. Try it on your data, and a
lso check out a
very similar in full function from Pandas. To deal with object t
ypes, it is useful to
print the data and literally look at it. It is useful to check u
nique
values with value counts function, and nulls location with

isnull function at times. In this lesson, we were discussing two things we can do with anonymized features. We saw that sometimes, it's possible to decode features, find out what this feature really means. It doesn't matter if we understand the meaning of the features or not, we should guess the feature types, in order to pre-process features accordingly to the type we have, and selected model class. In the next video, we'll see a lot of colorful plots, and talk about visualization, and other tools for exploratory data analysis. [SOUND]In the previous video, we've tried to decode anonymized features and guess their types. In fact, we want to do more. We want to generate new features and to find insights in a data. And in this lesson, we will talk about various visualizations that can help us with it. We will first to see what plots we can draw to explore individual features, and then we will get to exploration of feature relations. We'll explore pairs first and then we'll try to find feature groups in a dataset. First, there is no recipe how you find interesting things in the data. You should just spend some time looking closely at the data table, printing it, and examining. If we found something interesting, we then can take a closer look. So, EDA is kind of an art, but we have a bunch of tools for it which we'll discuss right now. The first, we can build histograms. Histograms split feature edge into bins and show how many points fall into each bin. Note that histograms may be misleading in some cases, so try to overwrite its number of bins when using it. Also, know that it aggregates in the data, so we cannot see, for example, if all the values are unique or there are a lot of repeated values. Let's see in other example. The first thing that I want to illustrate here is that histograms can confuse. Looking at this histogram, we could probably think that there are a lot of zero values in this feature. But in fact, if we take logarithm of the values and build histogram again, we'll clearly see that distribution is non-degenerate and there are many more distinct values than one. So my point is never make a conclusion based on a single plot. If you have a hypothesis, try to make several different plots to prove it. The second interesting thing here is that peak. What is it? It turns out that the peak is located exactly at the mean value of this feature. Seems like organizers filled the missing values with the mean values for us. So, now we understand that values were originally missing. How can we use this information? We can replace the missing values we found with not numbers, nulls again. For example, [inaudible] has a special algorithm that can fill missing values on its own and so, maybe [inaudible] will benefit from explicit missing values. Or we can fill the missing values with something other than feature mean, for example, with -999. Or we can generate a new feature which will indicate that the value was missing. This can be particularly useful for linear models. We can also build the plot where on X axis, we have a row index, and on the Y axis, we have feature values. It is convenient not to connect points with line segments but only draw them with circles. Now, if we observe horizontal lines on this kind of plot, we understand there are a

lot of repeated values in this feature. Also, note the randomness over the indices. That is, we see some horizontal patterns but no vertical ones. It means that the data is properly shuffled. We can also color code the points according to their labels. Here, we see that the feature is quite good as it presumably gives a nice class separation. And also, we clearly see that the data is not shuffled here. It is, in fact, sorted by class label. It is useful to examine statistics with Pandas' describe function. You can see examples of its output on the screenshot. It gives you information about mean, standard deviation, and several percentiles of the feature distribution. Of course, you can manually compute those statistics. In Pandas' nan type, you can find functions named by statistics they compute. Mean for mean value, var for variance, and so on, but it's really convenient to have them all in once. And finally, as we already discussed in the previous video, there is value_counts function to examine the number of occurrences of distinct feature values, and a function is null, which helps to find the missing values in the data. For example, you can visualize nulls patterns in the data as on the picture you see. So, here's the full list of functions we've discussed. Make sure you remember each of them. To this end, we've discussed visualizations for individual features. And now, let's get to the next topic of our discussion, exploration of feature relations. It turns out that sometimes, it's hard to make conclusions looking at one feature at a time. So let's look at the pairs. The best two here is a scatter plot. With it, we can draw one sequence of values versus another one. And usually, we plot one feature versus another feature. So each point on the figure correspond to an object with the feature values shown by points position. If it's a classification task, it's convenient to color code the points with their labels like on this picture. The color indicates the class of the object. For regression, the heat map light coloring can be used, too. Or alternatively, the target value can be visualized by point size. We can effectively use scatter plots to check if the data distribution in the train and test sets are the same. In this example, the red points correspond to class zero, and the blue points to class one. And on top of red and blue points, we see gray points. They correspond to test set. We don't have labels for the test set, that is why they are gray. And we clearly see that the red points are mixed with part of the gray ones, and that that is good actually. But other gray points are located in the region where we don't have any training data, and that is bad. If you see some kind of discrepancy between colored and gray points distribution, you should probably stop and think if you're doing it right. It can be just a bug in the code, or it can be completely overfitted feature, or something else that is for sure not healthy. Now, take a look at this scatter plot. Say, we plot feature X1 versus feature X2. What can we say about their relation? The right answer is X2 is less or equal than one_minus_X1. Just realize that the equation for the diagonal line is X1 + X2 = 1, and for all the points below the line, X2 is less or equal than one_minus_X1. So, suppose we found this relation between two features, how do we use this fact? Of course, it depends, but at least there are some obvious features to generate. For tree-based models, we can create a new features like the difference or ratio between X1 and X2. Now, take a

look at this scatter plot. It's hard to say what is the true relation between the features, but after all, our goal is not to decode the data here but to generate new features and get a better score. And this plot gives us an idea how to generate the features out of these two features. We see several triangles on the picture, so we could probably make a feature to each triangle a given point belongs, and hope that this feature will help. When you have a small number of features, you can plot all the pairwise scatter plots at once using scatter metrics function from Pandas. It's pretty handy. It's also nice to have histogram and scatter plot before the eyes at the same time as scatter plot gives you very vague information about densities, while histograms do not show feature interactions. We can also compute some kind of distance between the columns of our feature table and store them into a matrix of size number of features by a number of features. For example, we can compute correlation between the counts. It's the most common type of matrices people build, correlation metric. But we can compute other things than correlation. For example, how many times one feature is larger than the other? I mean, how many rows are there such that the value of the first feature is larger than the value of the second one? Or another example, we can compute how many distinct combinations the features have in the dataset. With such custom functions, we should build the metrics manually, and we can use matshow function from Matplotlib to visualize it like on the slide you see. If the metrics looks like a total mess like in here, we can run some kind of clustering like K-means clustering on the rows and columns of this matrix and reorder the features. This one looks better, isn't it? We actually came to the last topic of our discussion, feature groups. And it's what we see here. There are groups of very similar features, and usually, it's a good idea to generate new features based on the groups. Again, it depends, but maybe some statistics could collated over the group will work fine as features. Another visualization that helps to find feature groups is the following: We calculate the statistics of each feature, for example, mean value, and then plot it against column index. This plot can look quite random if the columns are shuffled. So, what if we sorted the columns based on this statistic? Feature and mean, in this case. It looks like it worked out. We clearly see the groups here. So, now we can take a closer look to each group and use the imagination to generate new features. And here is a list of all the functions we've just discussed. Pause the video and check if you remember the examples we saw. So, finally in this video, we we're talking about the tools and functions that help us with data exploration. For example, to explore features one by one, we can use histograms, plots, and we can also examine statistics. To explore a relation between the features, the best tool is a scatter plot. Scatter metrics combines several scatter plots and histograms on one figure. Correlation plot is useful to understand how similar the features are. And if we reorder the columns and rows of the correlation metrics, we'll probably find feature groups. And feature groups was the last topic we discussed in this lesson. We also saw a plot of sorted feature statistics and how it can reveal as feature groups. Well, of course, we've discussed only a fraction of helpful plots there are. With practice, you will develop and find your own tools further expl

oration.[MUSIC] Hi, in this video we will discuss
a little bit of dataset cleaning and see how to check if dataset is shuffled. It is important to understand that
the competition data can be only apart of the data organizers have. The organizers could give us
a fraction of objects they have or a fraction of features. And that is why we can have
some issues with the data. For example, we can encounter a feature
which takes the same value for every object in both train and test set. This could be due to
the sampling procedure. For example, the future is a year, and the organizers exported
us only one year of data. So in the original data
that the organizers have, this future is not constant, but
in the competition data it is constant. And obviously, it is not useful for
the models and just occupy some memory. So we are about to remove
such constant features. In this example data set
feature of zero is constant. It can be the case that the feature is constant on the train set but how is different values on the test set. Again, it is better to remove such
features completely since it is constant during training. In our dataset feature is f1. What is the problem, actually? For example, my new model can assign
some weight to this future, so this future will be a part of the prediction formula, and this formula will be completely unreliable for the objects
with the new values of that feature. For example, for
the last row in our data set. J row, even if categorical feature is not
constant on the train path but there were values that present only in the test data,
we need to handle this situation properly. We need to decide,
do these new values matter much or not? For example, we can simulate this
situation with a validation set and compare the quality of the predictions
on the objects with the syn feature values and
objects with the new feature values. Maybe we will decide to remove
the feature or maybe we will decide to create a separate model for
the object with a new feature values. Sometimes there are duplicated
numerical features that these two columns are completely identical. In our example data set,
these columns f2 and f3. Obviously, we should leave only one of those two features since the other one will not give any new information to the
model and will only slow down training. From a number of features, it's easy
to check if two columns are the same. We just can compare them element wise. We can also have duplicated
categorical features. The problem is that the features

can be identical but their levels have different names. That is it can be possible to rename
levels of one of the features and two columns will become identical. For example features f4 and f5. If we rename levels of the feature f5, C to A, A to B, and B to C. The result will look exactly as feature f4. But how do we find such
duplicated features? Fortunately, it's quite easy, it will take us only one more
line of code to find them. We need to label and code all
the categorical features first, and then compare them as if they were numbers. The most important part
here is label encoding. We need to do it right. We need to encode the features
from top to bottom so that the first unique value we see gets label 1, the second gets 2 and so on. For example for feature f4, we will encode A with 1,
B with 2 and C with 3. Now feature f5 will encode
it differently C will be 1, A will be 2 and B will be 3. But after such encodings columns f4 and f5 turn out to be identical and we can remove one of them. Another important thing
to check is if there are any duplicated rows in the train and test. Is to write a lot of duplicated rows
that also have different target, it can be a sign the competition
will be more like a roulette, and our validation will be different
to public leader board score, and private standing will be rather random. Another possibility, duplicated rows
can just be the result of a mistake. There was a competition where
one row was repeated 100,000 times in the training data set. I'm not sure if it was intentional or
not, but it was necessary to remove those duplicated rows to have a high score on the test set. Anyway, it's better to explain it to ourselves why do we
observe such duplicated rows? This is a part of data
understanding in fact. We should also check if train and
test have common rows. Sometimes it can tell us something
about data set generation process. And again we should probably think what
could be the reason for those duplicates? Another thing we can do,
we can set labels manually for the test rows that
are present in the train set. Finally, it is very useful to check
that the data set is shuffled, because if it is not then, there is
a high chance to find data leakage. We'll have a special topic about
date leakages later, but for now we'll just discuss
that the data is shuffled. What we can do is we can plug a feature or
target vector versus row index. We can optionally smooth
the values using running average. On this slide rolling target value from pairs competition is plotted while mean target value is shown with dashed blue line. If the data was shuffled properl

y we
would expect some kind of oscillation of the target values aroun
d
the mean target value. But in this case, it looks like
the end of the train set is much different to the start,
and we have some patterns. Maybe the information from this parti
cular
plot will not advance our model. But once again,
we should find an explanation for all extraordinary things we ob
serve. Maybe eventually, we will find something
that will lead us to the first place. Finally, I want to encoura
ge
you one more time to visualize every possible
thing in a dataset. Visualizations will lead
you to magic features. So this is the last slide for this lesson
. Hope you've learned something new and
excited about it. Here's a whole list of
topics we've discussed. You can pause this video and try to reme
mber what we were
talking about and where. See you later. [MUSIC][MUSIC] So in thi
s video,
I will go through Springleaf data, it was a competition on Kaggl
e. In that competition,
the competitors were to predict whether a client will respond to
direct mail offer provided by Springleaf. So presumably,
we'll have some features about client, some features about offer
, and we'll
need to predict 1 if he will respond and 0 if he will not, so le
t's start. We'll first import some libraries in here,
define some functions, it's not very interesting. And finally, l
et's load the data and train our test one, and
do a little bit of data overview. So the first thing we want to
know about
our data is the shapes of data tables, so let's bring the train
shape,
and test that test shape. What we see here, we have one
150,000 objects, both in train and test sets, and about 2000
features in both train and test. And what we see more than,
we have one more feature in train, and as humans, just target ca
n
continue to move the train. So we should just keep it in mind an
d
be careful, and drop this column when we feed our models. So let
's examine training and test, so let's use this function had
to print several rows of both. We see here we have ID column, an
d
what's interesting here is that I see in training we have values
 2, 4, 5,
7, and in test we have 1, 3, 6, 9. And it seems like they
are not overlapping, and I suppose the generation
process was as following. So the organizers created a huge
data set with 300,000 rules, and then they sampled at random,
rows for the train and for the test. And that is basically how w
e
get this train and test, and we have this column IG, it is row
index in this original huge file. Then we have something categor

ical,
then something numeric, numeric again, categorical, then
something that can be numeric or binary. But you see has decimal part,
so I don't know why, then some very strange values in here,
and again, something categorical. And actually,
we have a lot of in between, and yeah, we have target as the last column
of the train set, so let's move on. Probably another thing we want to
check is whether we have not a numbers in our data set, like nonce values,
and we can do it in several ways. And one way we, let's compute how many NaNs are there for
each object, for each row. So this is actually what we do here,
and we print only the values for
the first 15 rows. And so the row 0 has 25 NaNs, row 1 has 19 NaN,, and so on, but what's interesting here,
six rows have 24 NaNs. It doesn't look like we got it in random, it's really unlikely to
have these at random. So my hypothesis could be that
the row order has some structure, so the rows are not shuffled, and
that is why we have this kind of pattern. And that means that we probably could use row index as another feature for
our classifier, so that is it. And the same, we can do with columns, so for each column, let's compute how
many NaNs are there in each column. And we see that ID has 0 NaNs,
then some 0s, and then we see that a lot of
columns have the same 56 NaNs. And that is again something really
strange, so either every column will have 56 NaNs, and so it's not magic,
it's probably just how the things go. But if we know that there are a lot
of columns, and every column have more different number of NaNs, then it's
really unlikely to have a lot of columns nearer to each other in the data
set with the same number of NaNs. So probably, our hypothesis could
be here that the column order is not random, so
we could probably investigate this. So we have about 2,000
columns in this data, and it's a really huge number of columns.
And it's really hard to work
with this data set, and basically we don't have any names,
so the data is only mice. As I told you,
the first thing we can do is to determine the types of the data,
so we will do it here. So we're first going to continue train and
test on a huge data frame like the organizers had,
it will have 300,000 rows. And then we'll first use
a unique function to determine how many unique
values each column has. And basically here we bring
several values of what we found, and it seems like there are fiv

e columns
that have only one unique number. So we can drop the,
basically what we have here, we just find them in this line,
and then we drop them. So next we want to remove
duplicated features, but first, for convenience,
fill not a numbers with something that we can find easily later, and
then we do the following. So we create another data frame of size, of a similar shape as the training set. What we do we take
a column from train set, we apply a label encoder,
as we discussed in a previous video, and we basically store it
in this new train set. So basically we get another
data frame which is train, but label encoded train set. And having this data frame,
we can easily find duplicated features, we just start iterating
the features with two iterators. Basically, one is fixed and the second one
goes from the next feature to the end. Then we try to compare the columns, the
two columns that we're standing at, right. And if they are element wise the same,
then we have duplicated columns, and basically that is how we fill up
this dictionary of duplicated columns. We see it here, so
we found that variable 9 is duplicated for input 8, and
variable 18 again is duplicated for variable 8, and so on, and so we have
really a lot of duplicates in here. So this loop, it took some time,
so I prefer to dump the results to disk, so
we can easily restore them. So I do it here, and
then I basically drop those columns that we found from
the train test data frame. So yeah, in the second video, we will go through some features and do some work to data set. [MUSIC]So, let's continue exploration. We wanted to determine the types of variables, and to do that we will first use this nunique function to determine how many unique values again our feature have. And we use this dropna=False to make sure this function computes and accounts for nons. Otherwise, it will not count nun as unique value. It will just unhit them. So, what we see here that ID has a lot of unique values again and then we have not so huge values in this series, right? So I have 150,000 elements but 6,000 unique elements. 25,000, it's not that a huge number, right? So, let's aggregate this information and do the histogram of the values from above. And it's not histogram of these exact values but but it's normalized values. So, we divide each value by the number of rows in the tree. It's the maximum value of unique values we could possibly have. So what we see here that there are a lot of features that have a few unique values and there are several that have a lot, but not so much, not as much as these. So these features have almost in every row unique value. So, let's actually explore these. So, ID essentially is having a lot of unique values. No problem with that. But what is this? So what we actually see here, they are integers. They are huge numbers but they're integers. Well, I would expect a real, nunique variable with real values to have a lot of unique values, not integer typ

e variable. So, what could be our guess what these variables rep resent? Basically, it can be a counter again. But what else it c ould be? It could be a time in let's say milliseconds or nanosec onds or something like that. And we have a lot of unique values and no overlapping between the values because it's really unlike ly to have two events or two rows in our data set having the sam e time, let's say it's time of creation and so on, because the t ime precision is quite good. So yeah, that could be our guess. S o next, let's explore this group of features. Again with some ma nipulations, I found them and these are presented in this table. So, what's interesting about this? Actually, if you take a look at the names. So the first one is 541. And the second one is 54 3. Okay. And then we have 1,081 and 1,082, so you see they are s tanding really close to each other. It's really unlikely that ha lf of the row, if the column order was random, if the columns we re shuffled. So, probably the columns are grouped together accor ding to something and we could explore this something. And what' s more interesting, if we take a look at the values correspondin g to one row, then we'll find that'll say this value is equal to this value. And this value is equal to this value and this valu e, and this is basically the same value that we had in here. So, we have five features out of four of this having the same value . And if you examine other objects, some of them will have the s ame thing happening and some will not. So, you see it could be s omething that is really essential to the objects and it could be a nice feature that separates the objects from each other. And, it's something that we should really investigate and where we s hould really do some feature engineering. So, for say [inaudible] , it will be really hard to find those patterns. I mean, it ca nnot find. Well, it will struggle to find that two features are equal or five features are equal. So, if we create or say featur e that will calculate how many features out of these, how many f eatures we have have the same value say for the object zero wher e we'll have the value five in this feature and something for ot her rows, then probably this feature could be discriminative. An d then we can create other features, say we set it to one if the values in this column, this and this and this and this are the same and zero to otherwise, and so on. And basically, if you go through these rows, you will find that the patterns are differen t and sometimes the values are the same in different columns. So for example, for this row, we see that this value is equal to t his value. And this value is different to previous ones but its equal to this one. And it's really fascinating, isn't it? And if it actually will work and improve the model, I will be happy. A nd another thing we see here is some strange values and they loo k like nons. I mean, it's something that a human typed in or a m achine just autofilled. So, let's go further. Oh, yeah. And the last thing is just try to pick one variable from this group and see what values does it have. So, let's pick variable 15 and her e's its values. And minus 999 is probably how we've filled in th e nons. And yeah, we have 56 of them and all other values are no n-negative, so probably it's counters. I mean, how many events h appened in, I don't know, in the month or something like that. O kay. And finally, let's filter the columns and then separate col umns into categorical and numeric. And it's really easy to do us ing this function select_dtypes. Basically, all the columns that

will have objects type, if you would use a function dtypes. We think of them as categorical variables. And otherwise, if they are assigned type integer or float or something like that, or numeric type then we will think of these columns as numeric columns. So, we can go through the features one-by-one as actually I did during the competition. Well, we have 2,000 features in this data set and it is unbearable to go through a feature one-by-one. I've stopped at about 250 features. And you can find in my notebook and reading materials if you're interested. It's a little bit messy but you can see it. So, What we will do here, just several examples of what I was trying to investigate in data set, let's do the following. Let's take the number of columns, we computed them previously. So, we'll now work with only the first 42 columns and we'll create such metrics. And it looks like correlation matrices and all of that type of matrices like when we have the features along the y axis, features along the x axis. Basically, well, it's really huge. Yeah. And in this case, what we'll have as the values is the number or the fraction of elements of one feature that are greater than elements of the second feature. So, for example, this cell shows that all variables or all values in variable 50 are less than values and variable ID, which is expected. So, yeah. And it's opposite in here. So, if we see one in here it means that variable 45, for example, is always greater than variable 24. And, while we expect this metrics to be somehow random, if the count order was random. But, in here we see, for example, these kind of square. It means that every second feature is greater, not to the second but let's say i+1 feature is greater than the feature i. And, well it could be that this information is about, for example, counters in different periods of time. So, for example, the first feature is how many events happened in the first month. The second feature is how many events happened in the first two month and so kind of cumulative values. And, that is why one feature is always greater than the other. And basically, what information we can extract from this kind of metrics is that we have this group and we can generate new features and these features could be, for example, the difference between two consecutive features. That is how we will extract, for example, the number of events in each month. So, we'll go from cumulative values back to normal values. And, well linear models, say, neural networks, they could do it themselves but tree-based algorithms they could not. So, it could be really helpful. So, in attached to non-book in the reading materials you will see that a lot of these kind of patterns. So, we have one in here, one in here. The patterns, well, this is also a pattern, isn't it? And now we will just go through several variables that are different. So, for example, variable two and variable three are interesting. If you build a histogram of them, you will see something like that. And, the most interesting part here are these spikes. And you see, again, they're not random. There's something in there. So, if we take this variable two and build there, well, use this value count's function, we'll have value and how many times it occurs in this variable. We will see that the values, the top values, are 12, 24, 36, 60 and so on. So, they can be divided by 12 and well probably, this variable is somehow connected to time, isn't it? To hours. Well, and what can we do? We want to generate features so we will generate feature like the val

ue of these variable modular 12 or, for example, value of this v ariable integer division by 12. So, this could really help. In o ther competition, you could build a variable and see something l ike that again. And what happened in there, the organizers actua lly had quantized data. So, they only had data that in our case could be divided by 12. Say 12, 24 and so on. But, they wanted t o kind of obfuscate the data probably and they added some noise. And, that is why if you plot an histogram, you will still see t he spikes but you will also see something in between the spikes. And so, again, these features in that competition they work qui te well and you could dequantize the values and it could really help. And the same is happening with variable 3 basically, 0, 12 , 24 and so on. And variable 4, I don't have any plot for variab le 4 itself in here but actually we do the same thing. So, we ta ke variable 4, we create a new feature variable 4 modulus 50. An d now, we plot this kind of histogram. What you see here is ligh t green, there are actually two histograms in there. The first o ne for object from the class 0 and the second one for the object s from class 1. And one is depicted with light green and the sec ond one is with dark green. And, you see these other values. And , you see only difference in these bar, but, you see the differe nce. So, it means that these new feature variable 4 modulus 50 c an be really discriminative when it takes the value 0. So, one c ould say that this is kind of, well, I don't know how to say tha t., I mean, certain people would never do that. Like, why do we want to take away modular 50? But, you see sometimes this can re ally help. Probably because organizers prepare the data that way . So, let's get through categorical features. We have actually n ot a lot of them. We have some labels in here, some binary varia bles. I don't know what is this, this is probably is some proble ms with the encoding I have. And then, we have some time variabl es. This is actually not a time. Time. Not a time. Not a time. T his is time. Whoa, this is interesting. This looks like cities, right? Or towns, I mean, city names. And, if you remember what f eatures we can generate from geolocation, it's the place to gene rate it. And, then again, it was some time, some labels and once again, it's the states. Isn't it? So, again, we can generate so me geographic features. But particularly interesting, the featur es are the date. Dates that we had in here. And basically, these are all the columns that I found having the data information. S o, it was one of the best features for this competition actually . You could do the following, you could do a scatter plot betwee n two date features to particular date features and found that t hey have some relation, and, one is always greater than another. It means that probably these are dates of some events and one e vent is happening always after the first one. So, we can extract different features like the difference between these two dates. And in this competition, it really helped a lot. So, be sure to do exploratory data analysis and extract all the powerful featu res like that. Otherwise, if you don't want to look into the dat a, you will not find something like that. And, it's really inter esting. So, thank you for listening.Hi, everyone. In this video, I will tell you about the specifics of Numerai Competition that was held throughout year 2016. Note that Numerai organizers cha nged the format in 2017. So, the findings I'm going to read will not work on new data. Let's state the problem. Participants wer

e solving a binary classification task on a data set with 21 ano
nymized numeric features. Unusual part is that both train and te
st data sets have been updating every week. Data sets were also
shuffled column-wise. So it was like a new task every week. Pret
ty challenging. As it turned out, this competition had a data le
ak. Organizers did not disclose any information about the nature
of data set. But allegedly, it was some time series data with t
arget variable highly dependent on transitions between time poin
ts. Think of something like predicting price change in stock mar
ket here. Means that, if we knew true order or had timestamp var
iable, we could easily get nearly perfect score. And therefore,
we had to somehow reconstruct this order. Of course, approximate
ly. But even a rough approximation was giving a huge advantage o
ver other participants. The first and most important step is to
find a nearest neighbor for every point in a data set, and add a
ll 21 features from that neighbor to original point. Simple logi
stic regression of those 42 features, 21 from original, and 21 f
rom neighboring points, allowed to get into top 10 on the leader
board. Of course, we can get better scores with some Hardcore E
DA. Let's start exploring correlation metrics of new 21 features
. If group features with highest correlation coefficient next to
each other, we'll get a right picture. This picture can help us
in two different ways. First, we can actually fix some column o
rder. So, weekly column shuffling won't affect our models. And s
econd, we can clearly notice seven groups with three highly corr
elated features in each of them. So, the data actually has some
non-trivial structure. Now, let's remember that we get new data
sets every week. What is more? Each week, train data sets have t
he same number of points. We can assume that there is some conne
ction between consecutive data sets. This is a little strange be
cause we already have a time series. So, what's the connection b
etween the data from different weeks? Well, if we find nearest n
eighbors from every point in current data set from previous data
set, and plot distance distributions, we can notice that first
neighbor is much, much closer than the second. So, we indeed hav
e some connection between consecutive data sets. And it looks li
ke we can build a bijective mapping between them. But let's not
quickly jump into conclusions and do more exploration. Okay. We
found a nearest neighbor in previous data set. What if we examin
e the distances between the neighboring objects at the level of
individual features? We clearly have three different groups of s
even features. Now remember, the sorted correlation matrix? It t
urns out that each of three highly correlated features belong to
a different group. A perfect match. And if we multiply seven fe
atures from the first group by three, and seven features from th
e second group by two in the original data set, recalculate near
est neighbor-based features within the data sets, and re-train o
ur models, we'll get a nice improvement. So, after this magic mu
ltiplications, of course, I'd tried other constants, our true or
der approximation became a little better. Great. Now, let's move
to the true relation. New data, weekly updates, all of it was a
lie. Remember, how we were calculating neighbors between consec
utive data sets? Well, we can forget about consecutiveness. Calc
ulate neighbors between current data set, and the data set from
two weeks ago or two months ago. No matter what, we will be gett
ing pretty much the same distances. Why? The simplest answer is

that the data actually didn't change. And every week, we were ge
tting the same data, plus a little bit of noise. And thus, we co
uld find nearest neighbor in each of previous data sets, and ave
rage them all, successfully reducing the variance of added noise
. After averaging, true order approximation became even better.
I have to say that a little bit of test data actually did change
from time to time. But nonetheless, most of the roles migrated
from week to week. Because of that, it was possible to probe the
whole public leader board which helped even further, and so on,
and so on. Of course, there are more details regarding that com
petition, but they aren't very interesting. I wanted to focus on
the process of reverse engineering. Anyway, I hope you like thi
s kind of detective story and realized how important exploratory
data analysis could be. Thank you for your attention and always
pay respect to EDA.This isn't the rare case in competitions whe
n you see people jumping down on leaderboard after revealing pri
vate results. So, we ask ourselves, what is happening out there?
There are two main reasons for these jumps. First, competitors
could ignore the validation and select the submission which scor
ed best against the public leaderboard. Second, is that sometime
s competitions have no consistent public/private data split or t
hey have too little data in either public or private leaderboard
. Well, we as participants, can't influence competitions organiz
ation. We can certainly make sure that we select our most approp
riate submission to be evaluated by private leaderboard. So, the
broad goal of next videos is to provide you a systematic way to
set up validation in a competition, and tackle most common vali
dation problems. Let's quickly overview of the content of the ne
xt videos. First, in this video, we will understand the concept
of validation and overfitting. In the second video, we will iden
tify the number of splits that should be done to establish stabl
e validation. In the third video, we will go through most freque
nt methods which are used to make train/test split in competitio
ns. In the last video, we will discuss most often validation pro
blems. Now, let me start to explain the concept for validation f
or those who may never heard of it. In the nutshell, we want to
check if the model gives expected results on the unseen data. Fo
r example, if you've worked in a healthcare company which goal i
s to improve life of patients, we could be given the task of pre
dicting if a patient will be diagnosed a particular disease in t
he near future. Here, we need to be sure that the model we train
will be applicable in the future. And not just applicable, we n
eed to be sure about what quality this model will have depending
on the number of mistakes the model make. And on the predictive
probability of a patient having this particular disease, we may
want to decide to run special medical tests for the patient to
clarify the diagnosis. So, we need to correctly understand the q
uality of our model. But, this quality can differ on train data
from the past and on the unseen test data from the future. The m
odel could just memorize all patients from the train data and be
completely useless on the test data because we don't want this
to happen. We need to check the quality of the model with the da
ta we have and these checks are the validation. So, usually, we
divide data we have into two parts, train part and validation pa
rt. We fit our model on the train part and check its quality on
the validation part. Beside that, in the last example, our model

will be checked against the unseen data in the future and actually these data can differ from the data we have. So we should be ready for this. In competitions, we usually have the similar situation. The organizers of a competition give us the data in two chunks. First, train data with all target values. And second, test data without target values. As in the previous example, we should split the data with labels into train and validation parts. Furthermore, to ensure the competition spirit, the organizers split the test data into the public test set and the private test set. When we sent our submissions to the platform, we see the scores for the public test set while the scores for the private test set are released only after the end of the competition. This also ensures that we don't need the test set or in terms of a model do not overfit. Let me draw you an analogy with the disease projection, if we already divided our data into train and validation parts. And now, we are repeatedly checking our model against the validation set, some models, just by chance, will have better scores than the others. If we continue to select best models, modify them, and again select the best from them, we will see constant improvements in the score. But that doesn't mean we will see these improvements on the test data from the future. By repeating this over and over, we could just achieve the validation set or in terms of a competition, we could just cheat the public leaderboard. But again, if it overfit, the private leaderboard will let us down. This is what we call overfitting in a competition. Get an unrealistically good scores on the public leaderboard that later result in jumping down the private leaderboard. So, we want our model to be able to capture patterns in the data but only those patterns that generalize well between both train and test data. Let me show you this process in terms of underfitting and overfitting. So, to choose the best model, we basically want to avoid underfitting on the one side and overfitting on the other. Let's understand this concept on a very simple example of a binary classification test. We will be using simple models defined by formulas under the pictures and visualize the results of model's predictions. Here on the left picture, we can see that if the model is too simple, it can't capture underlined relationship and we will get poor results. This is called underfitting. Then, if we want our results to improve, we can increase the complexity of the model and that will probably find that quality on the training data is going down. But on the other hand, if we make too complicated model like on the right picture, it will start describing noise in the train data that doesn't generalize the test data. And this will lead to a decrease of model's quality. This is called overfitting. So, we want something in between underfitting and overfitting here. And for the purpose of choosing the most suitable model, we want to be able to evaluate our results. Here, we need to make a remark, that the meaning of overfitting in machine learning in general and the meaning of overfitting competitions in particular are slightly different. In general, we say that the model is overfitted if its quality on the train set is better than on the test set. But in competitions, we often say, that the models are overfitted only in case when quality on the test set will be worse than we have expected. For example, if you train gradient boosting decision tree in the competition is our area under a curve metric. We sometimes can ob

serve that the quality on the training data is close to one whil
e on the test data, it could be less for example, near 0.9. In g
eneral sense, the models overfitted here but while we get area u
nder curve was 0.9 on both validation and public/private test se
ts, we will not say that it is overfitted in the context of a co
mpetition. Let me illustrate this concept again in a bit differe
nt way. So, lets say for the purpose of model evaluation, we div
ided our data into two parts. Train and validation parts. Like w
e already did, we will derive model's complexity from low to hig
h and look at the models here. Note, that usually, we understand
 error or loss is something which is opposite to model's quality
 or score. In the figure, the dependency looks pretty reasonable
. For two simple models, we have underfitting which means higher
 on both train and validation. For two complex models, we have o
verfitting which means low error on train but again high error o
n validation. In the middle, between them, if the perfect model'
s complexity, it has the lowest train on the validation data and
 thus we expect it to have the lowest error on the unseen test d
ata. Note, that here the training error is always better than th
e test error which implies overfitting in general sense, but doe
sn't apply in the context of competitions. Well done. In this vi
deo, we define validation, demonstrated its purpose, and interpr
eted validation in terms of underfitting and overfitting. So, on
ce again, in general, the validation helps us answer the questio
n, what will be the quality of our model on the unseeing data an
d help us select the model which will be expected to get the bes
t quality on that test data. Usually, we are trying to avoid und
erfitting on the one side that is we want our model to be expres
sive enough to capture the patterns in the data. And we are tryi
ng to avoid overfitting on the other side, and don't make too co
mplex model, because in that case, we will start to capture nois
e or patterns that doesn't generalize to the test data.[SOUND] I
n the previous video,
we understood that validation helps us select a model which will
perform best on the unseen test data. But, to use validation, we
 first need
to split the data with given labels, entrain, and validation par
ts. In this video, we will discuss
different validation strategies. And answer the questions. How m
any splits should we make and what are the most often methods
to perform such splits. Loosely speaking, the main difference
between these validation strategies is the number of splits bein
g done. Here I will discuss three of them. First is holdout, sec
ond is K-fold,
and third is leave-one-out. Let's start with holdout. It's a sim
ple data split which
divides data into two parts, training data frame, and
validation data frame. And the important note here
is that in any method, holdout included, one sample can go
either to train or to validation. So the samples between train a
nd
the validation do not overlap, if they do,
we just can't trust our validation. This is sometimes the case,
when we have repeated samples in the data. And if we are,
we will get better predictions for these samples and
more optimistic all estimation overall. It is easy to see that t

hese can prevent
us from selecting best parameters for our model. For example,
over fitting is generally bad. But if we have duplicated samples
that present, and train, and test simultaneously and over feed,
validation scores can deceive us into a belief that maybe we are
moving
in the right direction. Okay, that was the quick note about
why samples between train and validation must not overlap. Back
to holdout. Here we fit our model on
the training data frame, and evaluate its quality on
the validation data frame. Using scores from this evaluation,
we select the best model. When we are ready to make a submission
, we can retrain our model on
our data with given labels. Thinking about using
holdout in the competition. It is usually a good choice,
when we have enough data. Or we are likely to get similar scores
for the same model,
if we try different splits. Great, since we understood
what holdout is, let's move onto the second validation
strategy, which is called K-fold. K-fold can be viewed as a repe
ated
holdout, because we split our data into key parts and iterate th
rough them, using
every part as a validation set only once. After this procedure,
we average scores over these K-folds. Here it is important to un
derstand
the difference between K-fold and usual holdout or bits of K-tim
es. While it is possible to average scores
they receive after K different holdouts. In this case,
some samples may never get invalidation, while others can be the
re multiple times. On the other side, the core idea of K-fold
is that we want to use every sample for validation only once. Th
is method is a good choice when we
have a minimum amount of data, and we can get either a sufficien
tly
big difference in quality, or different optimal
parameters between folds. Great, having dealt with K-fold, we ca
n move on to the third
validation strategy in our release. It is called leave-one-out.
And basically it is a special
case of Kfold when K is equal to the number
of samples in our data. This means that it will iterate
through every sample in our data. Each time usion came in a slot
minus
one object is a train subset and one object left is a test subse
t. This method can be helpful if
we have too little data and just enough model to entrain. So tha
t there, validation strategies. Holdout, K-fold and leave-one-ou
t. We usually use holdout or
K-fold on shuffle data. By shuffling data we are trying to
reproduce random trained validation split. But sometimes, especi
ally if you
do not have enough samples for some class, a random split can fa
il. Let's consider, for an example. We have binary classificatio
n tests and
a small data set with eight samples. Four of class zero, and fou

r of class one. Let's split 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 two third instead of one half. This can drastically change
predictions of our model. What we need here to handle
this problem is stratification. It is just the way to insure we'll get similar target
distribution over different faults. If we split data into four
faults with stratification, the average of each false target
values will be equal to one half. It is easier to guess that 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 multiclass classification
tasks with huge amount of classes. For good classification data sets, stratification split will be quite
similar to a simple shuffle split. That is, to a random split. Well done, in this video we have discussed
different validation strategies and reasons to use each one of them. Let's summarize this all. If we have enough data, and
we're likely to get similar scores and optimal model's parameters for
different splits, we can go with Holdout. If on the contrary, scores and
optimal parameters differ for different splits,
we can choose KFold approach. And event, if we too little data,
we can apply leave-one-out. The second big takeaway from this video
for you should be stratification. It helps make validation more stable,
and especially useful for small and unbalanced datasets. Great. In the next videos we will continue to
comprehend validation at it's core. [SOUND] [MUSIC]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. Let's plan these two options next. Now, when you think about features you need to generate and the model you need to train, how complicated these two cases are? In the first block, we can just interpret 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 really that if we make train validation split different fr

om train/test split, then we are going to create a useless model . And here, we get to the main rule of making a reliable validat ion. We should, if possible, set up validation to mimic train/te st split, but that's a little later. Let's go back to our exampl e. On the second picture, for most of test point, we have neithe r the next value nor the previous one. Now, let's imagine we hav e 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 signi ficantly? Well, it is certain that if you want to predict what w ill happen a few points later, then the model which favor featur es like previous and next target values will perform poorly. It happens because in this case, we just don't have such observatio ns for the test data. But we have to give the model something in the feature value, and it probably will be not numbers or missi ng values. How much experience that model have with these type o f situations? Not much. The model just won't expect that and qua lity will suffer. Now, let's remember the second case. Actually, here we need to rely more on the time trend. And so, the featur es, which is the model really we need here, are more like what w as the trend in the last couple of months or weeks? So, that sho ws that the model selected as the best model for the first type of validation will perform poorly for the second type of validat ion. On the opposite, the best model for the second type of vali dation was trained to predict many points ahead, and it will not use adjacent target values. So, to conclude this comparison, th ese 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 t o time-based patterns, we'll get a reliable validation with a ra ndom-based split. Let me say this again in another words. If we' ll create features which are useful for a time-based split and a re useless for a random split, will be correct to use a random s plit to select the model? It's a tough question. Let's take a mo ment and think about it. Okay, now let's answer this. Consider t he case when target falls a linear train. In the first block, we see the exact case of randomly chosen validation. In the second , we see the same time-based split as we consider before. first, let's notice that in general, model predictions will be close t o targets mean value calculated using train data. So in the firs t block, if the validation points will be closer to this mean va lue compared to test points, we'll get a better score in validat ion than on test. But in the second case, the validation points are roughly as far as the test points from target mean value. An d 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 i ncorrect validation, not only features, but the value target can lead to unrealistic estimation of the score. Now, that example was quite similar to what you may encounter while solving real c ompetitions. Numerous competitions use time-based split namely: the Rossmann Store Sales competition, the Grupo Bimbo Inventory Demand competition and others. So, to quickly summarize this val uable example we just have discussed, different splitting strate gies can differ significantly, namely: in generated features, in

the way the model will rely on that features, and 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. Most splits can be united into three categories: a random split, a time-based split and the id-based split. Let's start with the most basic one, the random split. Let's start, 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 alone. 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, within 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 will present in the train data and his wife will present in the test data. We probably can explore this and devise a special feature for that case. For in such possibilities, and realizing that kind of features is really interesting. More in this case and others I will mention here, comes in the next lesson of our course. So again, that was a random split. The second method is a time-based split. We already discussed the unit 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 the rating 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 came 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. As I mentioned before, this split is widespread enough. It was used in a Rossmann store sales competition and in the Grupo Bimbo inventory demand competition, and in other's competitions. 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. Now, having dealt with the random and the time-based splits, let's discuss the ID-based split. ID can be a unique identifier of user, shop, or any other entity. 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 competiti

on, we had to classify patients into three 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 risk and make a random-based 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. Fine. 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 underfitting. So, in order to select optimal models, it was crucial to mimic the ratio of new search terms from train/test split. Great. This is it. 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 a 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.[SOUND] Hi and welcome back. In the previous videos we discussed
the concept of validation and overfitting. And discussed how to chose
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 relation will, in a way, substitute
a leaderboard for us. That is the score we see on
the validation will be the same for the private leaderboard. Or
at least, if we improve our model and 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 p
roblems can
be divided into two big groups. In the first group are the probl
ems
we encounter during local validation. Usually they are caused by
inconsistency of the data, a widespread example is getting diffe
rent
optimal parameters for different faults. In this case we need to
make
more thorough validation. The problems from the second group, of
ten 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 us
ually occurs because we can't mimic the exact
train test split on our validation. These are tough problems, an
d we
definitely want to be able to handle them. So before we start,
let me provide an overview of this video. For both validation an
d submission
stages we will discuss main problems, their causes, how to handl
e them. And then, we'll talk a bit about when
we can expect a leaderboard shuffle. Let's start with discussion
of 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.
So we can easily explain this problem. Consider that we need to
predict
sales in a shop in February. Say we have target values for the l
ast year, and, usually,
we will take last month in the validation. This means January, b
ut clearly January
has much more holidays than February. And people tend to buy mor
e, 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 t
his kind
of model behavior can be expected. But what if there is no clear
reason
why scores differ for different folds? Let identify several comm
on reasons for
this and see what we can do about it. The first hypotheses we sh
ould consider
and that we have too little data. For example, consider a case w

hen we have
a lot of patterns and trends in the data. But we do not have eno ugh samples
to generalize these patterns well. In that case, a model will ut ilize
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 sam ples
will be different each time only increasing the dispersion of sc ores for
different folds. The second type of this,
is data is too diverse and inconsistent. For example, if you hav e very similar
samples with different target variance, a model can confuse them . Consider two cases, first, if one of such examples is in the t rain
while another is in the validation. We can get a pretty high err or for
the second sample. And the second case,
if both samples are in validation, we will get smaller errors fo r them. Or let's remember another
example of diverse data we have already discussed a bit earlier. I'm talking about the example of
predicting sales for January and February. Here we have the natu re 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 d ifference in
scores and optimal model parameters for different folds are, fir st,
having too little data, and second, having too diverse and inconsistent data. Now let's outline our actions here. If we are facing this kind of problem, it can be useful to make
more thorough validation. You can increase K in KFold,
but usually 5 folds are enough. Make KFold validation several ti mes
with different random splits. 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 overfi t. It is useful to use one set of KFold
splits to select parameters and another set of KFold splits
to check model's quality. Examples of competitions which
required extensive validation include the Liberty Mutual Group P roperty
Inspection Prediction competition and the Santander Customer Sat isfaction
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. Now, having discussed validation stage problems, let's move on to
submission stage problems. Sometimes you can diagnose these prob lems

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, 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 tests
pre made by organizers. I won't lie to you,
it can be quite hard to identify and mimic the exact train tests here. Because of that, I highly 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. Recall, we already have different
model scores on different folds during validation. 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 suprising. 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. The first reason, we have too
little data in public leaderboard, which is pretty self explanatory. Just trust your validation,
and everything will be fine. And the second train and test data are from different distributions. Let me explain what I mean when I
talk about different distributions. Consider a regression test 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 these in a competition. Okay, 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 big 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. 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, vice versa. 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 same distribution. In that case, you ensure that
your validation will be fair. This is true for getting raw 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 test 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 that 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 assistant. 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 splitter
strategy for train validation split. And finally, you ensured the same
distributions in validation and testing. But sometimes you have to expect
leaderboard shuffle anyway, and not just for you, but for everyone. First, for those who've never heard of it,
a leaderboard shuffle happens when participants position some public and

private leaderboard drastically different. Take a look at this s
creenshot from
the two sigma financial model in challenge competition. The gree
n and
the red arrows mean how far a team moved. For example, the parti
cipant 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 wh
en all participants
have very similar scores. This can be either a very good score o
r
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 L
iberty Mutual Group, Property Inspection Prediction
competition. In that competition,
scores of competitors were very close. And though randomness did
n'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 KFold platform.
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 Restau
rant
Revenue Prediction Competition. In that competition, trained set
consisted of less than 200 gross. And this set consisted
of less than 400 gross. 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 leaderboards. As people tend to adjust
 their
submission to public leaderboard and overfit, we can expect wors
e
results on private leaderboard. Here again, trust your validatio
n and
everything will be fine. Okay, let's go over reasons for
leaderboard shuffling. 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 team 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 trained test split. [MUSIC]Hi everyone. In this section, we will talk about a very sensitive topic data leakage or more simply, leaks. 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 increase. People often are very sensitive about this matter. They tend to overreact. That's completely understandable. After spending a lot of time on solving the problem, a sudden data leak may render all of that useless. It is not a pleasant position to be in. I cannot force you to turn the blind eye but keep in mind, there is no ill intent whatsoever. Data leaks are the result of unintentional errors, accidents. Even if you find yourself in a competition with an unexpected data leak close to the deadline, please be more tolerant. The question of whether to exploit the data leak or not is exclusive to machine learning competitions. In real world, the answer is obviously a no, nothing to discuss. But in a competition, the ultimate goal is to get a higher leaderboard position. And if you truly pursue that goal, then exploit the leak in every way possible. Further in this section, I will show you the main types of data leaks that could appear during solving a machine learning problem. 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 the 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't 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 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.[SOUND] 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. 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. 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 were able 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 once 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? 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 metainformation, 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. At distance from users
seeking 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
Flavours 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. Org
anizers wanted a machine
learning solution for something that has never been observed. Th
at's why the signal was simulated. But simulation cannot be perf
ect 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 coul
d bypass the tests,
fully exploit simulation flaws, and get a perfect score on the l
eaderboard. 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. There is one thing com
mon 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 tha
t'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 ca
n 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. When two items have
similar sets of neighbors they have a high possibility
of being duplicates. This is it with data leaks. I hope you got
the concept and
found a lot of interesting examples. Thank you for your attentio
n. [SOUND]Hi, everyone. In this video, I will tell you how I and
 my teammates, Stanislav Smirnov solved Kaggle Expedia hotel rec
ommendations competition. Personally, one of my favorites, proba
bly among top five most interesting competitions I've ever parti
cipated in. I'll state the problem now. So, if you came here rig
ht after Data Leaks lesson, it should already be familiar to you
. Anyway, in that competition, we worked with lots of customer b
ehavior. These include what customers searched for, how they int
eracted with search results, clicks or books, and whether or not
 the search result was a travel package, and Expedia was interes
ted in predicting which hotel group a user is going to book. Imp
ortant thing here is prediction target the hotel group. In other
 words, characteristics of actual hotel, remember it. As it turn
ed out, this competition had a very non-trivial and extremely ha
rd to exploit data leak. From the first glance, data leak was pr
etty straightforward. We had a destination distance among the fe
ature. It's a distance from user city to an actual hotel he clic
ked on booked. And, as I said earlier, our prediction target is
a characteristic of an actual hotel. Furthermore, destination di
stance was very precise so unique user city and destination dist

ance 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. 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 current 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. 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 or 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 done on 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 poin t. 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, l et's create a grid around its center. Something like 10 kilomete rs 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 correspondin g to that circumference. During inference, we also draw a circum ference 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 featu res directly used target label. We cannot use them as is in trai ning. We should generate them in out-of-fold fashion for train d ata. So we had training data for years 2013 and 2014. To generat e features for year 2014, we used labelled data from year 2013 a nd vice versa, used the year 2014 to generate features for the y ear 2013. For the test features, which was from year 2015, we na turally used all training data. In the end, we calculated a lot of features and serve them into Xgboost model. After 16 hours of training for the course, we got our results. We ended up on thi rd position on public leader-boards and forth on private. We did good, but we still did not fully exploit data leakage. If you c heck the leaderboard, you'll notice the difference in scores bet ween first place and the rest. Under speculation, the winner did extraordinary. Although, in general, his methods were very simi lar to ours. He was able to extract way more signal. Finally, I hope you enjoyed my story. As you can see, sometimes working wit h data leakage could be very interesting and challenging. You ma y develop some unusual skills and broaden your horizons. Thank y ou for your attention.

.

.

[MUSIC] Hi, in this lesson, we will talk about
a major part of any competition. The metrics that are used
to evaluate a solution. In this video, we'll discuss why
there are so many metrics and why it is necessary to know what
metric is used in a competition. In the following videos, we wil
l study what is the difference
between a loss and a metric? And we'll overview and show optimiz
ation
techniques for the most important and common metrics. In the cou
rse, we focus on regression and
classification. So we only discuss metric for these tasks. For b
etter understanding, we will also
build a simple baseline for each metric. That is what the best c
onstant to
predict for that particular method. So metrics are an essential
part of any competition. They are used to evaluate our submissio
ns. Okay, but why do we have a different
evolution metric on each competition? That is because there are
plenty of ways
to measure equality of an algorithm and each company decides for
 themselves what is the most appropriate
way for their particular problem. For example, let's say an onli
ne shop is trying to
maximize effectiveness of their website. The thing is you need t
o
formalize what is effectiveness. You need to define a metric
how effectiveness is measured. It can be a number of times
a website was visited, or the number of times something
was ordered using this website. So the company usually decides f
or
itself what quantity is most important for it and
then tries to optimize it. In the competitions, the metrics
is fixed for us and the models and competitors are ranked using
it. In order to get higher leader board score
you need to get a better metric score. That's basically the only
 thing in the
competition that we need to care about, how to get a better scor
e. And so it is very important to
understand how metric works and how to optimize it efficiently.
I want to stress out that
it is really important to optimize exactly the metric we're give
n in
the competition and not any other metric. Consider an example, b
lue and red lines represent objects of
a class zero and one respectively. And say we decided to use
a linear classifier,and came up with two matrix to optimize, M1
and M2. The question is, how much different
the resulting classifiers would be? Actually by a lot. The two l
ines here, the solid and the dashed one show the best line
your boundaries for the two cases. For the dashed, M1 score is t
he highest
among all possible hyperplanes. But M2 score for the hyperplane
is low. And we have an opposite situation for
the solid boundary. M2 score is the highest,
whereas M1 score is low. Now, if we know that in this particular
competition, the ranking is based on M1 score, then we need to o

ptimize M1 score
and so we should submit the prediction. Predictions of the model with dash boundary. Once again,
if your model is scored with some metric, you get best results by
optimizing exactly that metric. Now, the biggest problem is that some
metrics cannot be optimized efficiently. That is there is no simple enough way
to find, say, the optimal hyperplane. That is why sometimes we need to
train our model to optimize something different than competition metric. But in this case we will need to
apply various heuristics to improve competition metric score. And there's another case where we
need to be smart about the metrics. It is one that train and
the test sets are different. In the lesson about leaks,
we'll discuss leader board probing. That is, we can check, for example, if the mean target value on public part
of test set is the same as on train. If it's not, we would need to adapt our
predictions to suit rest set better. This is basically a specific metric
optimization technique we apply, because train and test are different. Or there can be more severe cases
where improved metric validation set could possibly not result into
improved metric on the test set. In these situations,
it's a good idea to stop and think maybe there is a different
way to approach the problem. In particular, time series can
be very challenging to forecast. Even if you did a validation just right. [INAUDIBLE] by time, rolling windows,
fill the distribution in the future can be much different
from what we had in the train set. Or sometimes,
there's just not enough training data, so a model cannot capture the patterns. In one of the compositions I took part, I had to use some tricks to boost
my score after the modeling. And the trick was as a consequence
of a particular metric used in that competition. The metric was quite unusual actually,
but it is intuitive. If a trend is guessed correctly, then the
absolute difference between the prediction and the target is considered as an error. If for instance, model predict end
value in the prediction horizon to be higher than the last value from the train
side but in reality it is lower, then the trend is predicted incorrectly,
and the error was set to
absolute difference squared. So if we predict a value to be
above the dashline, but it turns out to be below or vice versa, the trend
[INAUDIBLE] to be predicted incorrectly. So this metric carries
a lot more about correct trend to be predicted than
about actual value you predict. And that is something it
was possible to exploit. There were several times
series was to forecast, the horizon to predict was wrong, and

the model's predictions were unreliable. Moreover, it was not possible
to optimize this metric exactly. So I realized that it would be much better
to set all the predictions to either last value plus a very tiny constant,
or last value minus very tiny constant. The same value for all the points in
the time interval, we are to predict for each time series. And design depends on the estimation. What is more likely the values in the horizon to be lower than the last known value,
or to be higher? This trick actually took me to
the first place in that competition. So finding a nice way to optimize
a metric can give you an advantage over other participants,
especially if the metric is peculiar. So maybe I should formulate it like that. We should not forget to do kind of
exploratory metric analysis along with exploratory data analysis. At least when the metric
is an unusual one. So in this video we've understood
that each business has its own way to measure ineffectiveness
of an algorithm based on its needs, and therefore, there are so many different metrics. And we saw two motivational examples. Why should we care about the metrics? Well, basically because it is how
competitors are compared to each other. In the following videos we'll
talk about concrete metrics. We'll first discuss high level intuition for each metric and then talk about optimization techniques. [MUSIC]In this video, we will review the most common ranking metrics and establish an intuition about them. Although in a competition, the metric is fixed for us, it is still useful to understand in what cases one metric could be preferred to another. In this course, we concentrate on regression and classification, so we will only discuss related metrics. For a better understanding, for each metric, we will also build the most simple baseline we could imagine, the constant model. That is, if we are only allowed to predict the same value for every object, what value is optimal to predict according to the chosen metric? Let's start with regression task and related metrics. In the following videos, we'll talk about metrics for classification. First, let us clarify the notation we're going to use throughout the lesson. N will be the number of samples in our training data set, y is that the target, and y-hat is our model's predictions. And y-hat and y with index i are the predictions, and target value respectively for i-th object. The first metric we will discuss is Mean Square Error. It is for sure the most common metric for regression type of problems. In data science, people use it when they don't have any specific preferences for the solution to their problem, or when they don't know other metric. MSE basically measures average squared error of our predictions. For each point, we calculate square difference between the predictions of the target and then average those values over the objects. Let's introduce a simple data set now. Say, we have five objects, and each object has some features, X, and the target is shown in the column Y. Let's ask ourselves a question. How will the error change if

we fix all the predictions but want to be perfect, and we'll de rive the value of the remaining one? To answer this question, ta ke a look at this plot. On the horizontal line, we will first pu t points to the positions of the target values. The points are c olored according to the corresponding rows in our data table. An d on the Y-axis, we will show the mean square error. So, let's n ow assume that our predictions for the first four objects are pe rfect, and let's draw a curve. How the metric value will change if we change the prediction for the last object? For MSE metric, it looks like that. In fact, if we predict 25, the error is zer o, and if we predict something else, then it is greater than zer o. And the error curve looks like parabola. Let's now draw analo gous curves for other objects. Well, right now it's hard to make any conclusions but we will build the same kind of plot for eve ry metric and we will note the difference between them. Now, let 's build the simplest baseline model. We'll not use the features X at all and we will always predict a constant value Alpha. But , what is the optimal constant? What constant minimizes the mean square error for our data set? In fact, it is easier to set the derivative of our total error with respect to that constant to zero, and find it from this equation. What we'll find is that th e best constant is the mean value of the target column. If you t hink you don't know how to derive it, take a look at the reading materials. There is a fine explanation and links to related boo ks. But let us constructively check it. Once again, on the horiz ontal axis, let's denote our target values with dot and draw a f unction. How the error changes is if we change the value of that constant Alpha? We can do it with a simple grid search over a g iven range by changing Alpha intuitively and recomputing an erro r. Now, the green square shows a minimum value for our metric. T he constant we found is 10.99, and it's quite close to the true mean of the target which is 11. In fact, the value we got deviat es from the true mean value only because with the grid search, w e get only approximate answer. Also note that the red curve on t he second plot is uniformly same and average of the curves from the first plot. We are finished discussing MSE metric itself, bu t there are two more related metrics used frequently, RMSE and R _squared. And we will briefly study them now. RMSE, Root Mean Sq uare Error, is a very similar metric to MSE. In fact, it is calc ulated in two steps. First, we calculate regular mean square err or and then, we take a square root of it. The square root is int roduced to make scale of the errors to be the same as the scale of the targets. For MSE, the error is squared, so taking a root out of it makes total error a little bit easier to comprehend be cause it is linear now. Now, it is very important to understand in what sense RMSE is similar to MSE, and what is the difference . First, they are similar in terms of their minimizers. Every mi nimizer of MSE is a minimizer of RMSE and vice versa. But genera lly, if we have two sets of predictions, A and B, and say MSE of A is greater than MSE of B, then we can be sure that RMSE of A is greater RMSE of B. And it also works in the opposite directio n. This is actually true only because square root function is no n-decreasing. What does it mean for us? It means that, if our ta rget the metric is RMSE, we still can compare our models using M SE, since MSE will order the models in the same way as RMSE. And we can optimize MSE instead of RMSE. In fact, MSE is a little b

it easier to work with, so everybody uses MSE instead of RMSE. But there is a little bit of difference between the two for gradient-based models. Take a look at the gradient of RMSE with respect to i-th prediction. It is basically equal to gradient of MSE multiplied by some value. The value doesn't depend on the index I. It means that travelling along MSE gradient is equivalent to traveling along RMSE gradient but with a different flowing rate and the flowing rate depends on MSE score itself. So, it is kind of dynamic. So even though RMSE and MSE are really similar in terms of models scoring, they can be not immediately interchangeable for gradient based methods. We will probably need to adjust some parameters like the learning rate. Now, what if I told you that MSE for my models predictions is 32? Should I improve my model or is it good enough? Or what if my MSE was 0.4? Actually, it's hard to realize if our model is good or not by looking at the absolute values of MSE or RMSE. It really depends on the properties of the dataset and their target vector. How much variation is there in the target vector. We would probably want to measure how much our model is better than the constant baseline. And say, the desired metrics should give us zero if we are no better than the baseline and one if the predictions are perfect. For that purpose, R_squared metric is usually used. Take a look. When MSE of our predictions is zero, the R_squared is 1, and when our MSE is equal to MSE over constant model, then R_squared is zero. Well, because the values in numerator and denominator are the same. And all reasonable models will score between 0 and 1. The most important thing for us is that to optimize R_squared, we can optimize MSE. It will be absolutely equivalent since R_squared is basically MSE score divided by a constant and subtracted from another constant. These constants doesn't matter for optimization. Lets move on and discuss another metric called Mean Absolute Error, or MAE in short. The error is calculated as an average of absolute differences between the target values and the predictions. What is important about this metric is that it penalizes huge errors that not as that badly as MSE does. Thus, it's not that sensitive to outliers as mean square error. It also has a little bit different applications than MSE. MAE is widely used in finance, where $10 error is usually exactly two times worse than $5 error. On the other hand, MSE metric thinks that $10 error is four times worse than $5 error. MAE is easier to justify. And if you used RMSE, it would become really hard to explain to your boss how you evaluated your model. What constant is optimal for MAE? It's quite easy to find that its a median of the target values. In this case, it is eight. See reading materials for a proof. Just to verify that everything is correct, we again can try to Greek search for an optimal value with a simple loop. And in fact, the value we found is 7.98, which indicates we were right. Here, we see that MAE is more robust than MSE, that is, it is not that influenced by the outliers. In fact, recall that the optimal constant for MSE was about 11 while for MAE it is eight. And eight looks like a much better prediction for the points on the left side. If we assume that point with a target 27 is an outlier and we should not care about the prediction for it. Another important thing about MAE is its gradients with respect to the predictions. The grid end is a step function and it takes -1 when Y_hat is smaller than the target and +1 when it is larger. Now,

the gradient is not defined when the prediction is perfect, because when Y_hat is equal to Y, we can not evaluate gradient. It is not defined. So formally, MAE is not differentiable, but in fact, how often your predictions perfectly measure the target. Even if they do, we can write a simple IF condition and return zero when it is the case and through gradient otherwise. Also know that second derivative is zero everywhere and not defined in the point zero. I want to end the discussion with the last note. Well, it has nothing to do with competitions but every data scientists should understand this. We said that MAE is more robust than MSE. That is, it is less sensitive to outliers, but it doesnt mean it is always better to use MAE. No, it does not. It is basically a question. Are there any real outliers in the dataset or there are just, let's say, unexpectedly high values that we should treat just as others? Outliers have usually mistakes, measurement errors, and so on, but at the same time, similarly looking objects can be of natural kind. So, if you think these unusual objects are normal in the sense that they're just rare, you should not use a metric which will ignore them. And it is better to use MSE. Otherwise, if you think that they are really outliers, like mistakes, you should use MAE. So in this video, we have discussed several important metrics. We first discussed, mean square error and realized that the best constant for it is the mean targeted value. Root Mean Square Error, RMSE, and R_squared are very similar to MSE from optimization perspective. We then discussed Mean Absolute Error and when people prefer to use MAE over MSE. In the next video, we will continue to study regression metrics and then we'll get to classification ones.[SOUND] In the previous video,
we started to discuss regression metrics. In this video,
we'll talk about three more metrics, (R)MSPE, MAPE, and (R)MSLE. Think about the following problem. We need to predict,
how many laptops two shops will sell? And in the train set for a particular date, we see that the first shop sold 10 items, and the second sold 1,000 items. Now suppose our model predicts
9 items instead of 10 for the first shop, and
999 instead of 1,000 for the second. It could happen that off by one error in the first case, is much more critical
than in the second case. But MSE and MAE are equal to one for both shops predictions, and thus according to those metrics, these
off by one errors are indistinguishable. This is basically because MSE and
MAE work with absolute errors while relative error can
be more important for us. Off by one error for
the shops that sell ten items is equal to mistaking by 100 items for
shops that sell 1,000 items. On the plot for MSE and MAE, we can see that all the error curves have
the same shape for every target value. The curves are kind of shifted
version of each other. That is an indicator that metric
works with absolute errors. The relative error preference
can be expressed with Mean Square Percentage Error, MSPE in short, or
Mean Absolute Percentage Error, MAPE. If you compare them to MSE

and MAE,
you will notice the difference. For each object, the absolute er
ror
is divided by the target value, giving relative error. MSPE and
MAPE can also be thought
as weighted versions of MSE and MAE, respectively. For the MAPE,
the weight of its sample is
inversely proportional to it's target. While for MSPE, it is inv
ersely
proportional to a target square. Know that the weight do
not sum up to one here. You can take a look at this
individual error plus for our individual sample dataset. Now, we
see the course became more
flat as the target value increases. It means that, the cost we p
ay for
a fixed absolute error, depends on the target value. And as the
target increases, we pay less. So having talk about definition a
nd
motivation behind MSPE and MAPE. Let's now think, what are the o
ptimal
constant predictions for these matrix? Recall that for MSE, the
optimal
constant is the mean over target values. Now, for MSPE, the weig
hted
version of MSE, in turns out that the optimal constant is weight
ed
mean of the target values. For our dataset,
the optimal value is about 6.6, and we see that it's biased
towards small targets. Since the absolute error for
them is weighted with the highest weight, and thus inputs metric
the most. Now the MAPE, this is a question for you. What do you
think is
an optimal constant for it? Just use your intuition here and
knowledge from the previous slides. Especially recall that MAPE
is weighted version of MAE. The right answer is,
the best constant is weighted median. It is not a very commonly
used
quantity actually, so take a look for a bit of explanation in
the reading materials. The optimal value here is 6, and it is
even smaller than the constant for MSPE. But do not try to expla
in
it using outliers. If an outlier had a very,
very small value, MAPE would be very biased towards it, since th
is
outlier will have the highest weight. All right, now let's move
on to
the last metric in this video, Root Mean Square Logarithmic Erro
r,
or RMSLE in short. What is RMSLE? It is just an RMSE calculated
in logarithmic scale. In fact, to calculate it,
we take a logarithm of our predictions and the target values, an
d
compute RMSE between them. The targets are usually non-negative
but
can equal to 0, and the logarithm of 0 is not defined. That is w
hy a constant is usually

added to the predictions and the targets before applying
the logarithmic operation. This constant can also be
chosen to be different to one. It can be for example 300
depending on organizer's needs. But for us, it will not change much. So, this metric is usually used
in the same situation as MSPE and MAPE, as it also carries about relative
errors more than about absolute ones. But note the asymmetry
of the error curves. From the perspective of RMSLE, it is always better to predict more
than the same amount less than target. Same as root mean square error doesn't
differ much from mean square error, RMSLE can be calculated
without root operation. But the rooted version
is more widely used. It is important to know that the plot
we see here on the slide is built for a version without the root. And for a root version,
an analogous plot would be misleading. Now let's move on to the question
about the best constant. I will let you guess the answer again. Just recall that, Just recall what
is the best constant prediction for RMSE and
use the connection between RMSLE and RMSE. To find the constant, we should realize
that we can first find the best constant for RMSE in the log space, will
be the weighted mean in the log space. And after it, we need to get back from log space to
the usual one with an inverse transform. The optimal constant turns out to be 9.1. It is higher than constants for
both MAPE and MSPE. Here we see the optimal constants for
the metrics we've broken down. MSE is quite biased towards
the huge value from our dataset, while MAE is much less biased. MSPE and MAPE are biased
towards smaller targets because they assign higher weight to
the object with small targets. And RMSLE is frequently considered
as better metrics than MAPE, since it is less biased towards small
targets, yet works with relative errors. I strongly encourage you to
think about the baseline for metrics that you can face for first time. It truly helps to build an intuition and
to find a way to optimize the metrics. So, in this video, we will discuss different metrics
that works with relative errors. MSPE, means square percentage error,
MAPE, mean absolute percentage error, and RMSLE,
root mean squared logarithmic error. We'll discussed the definitions and
the baseline solutions for them. In the next video, we will study
several classification matrix. [MUSIC][MUSIC] In the previous videos, we discussed
metrics for regression problems. And here,
we'll review classification metrics. We will first talk about ac

curacy,
logarithmic loss, and then get to area under a receiver
operating curve, and Cohen's Kappa. And specifically Quadratic w
eighted Kappa. Let's start by fixing the notation. N will be the
 number of objects in our
dataset, L, the number of classes. As before, y will stand for t
he target,
and y hat, for predictions. If you see an expression in square
brackets, that is an indicator function. It fields one if the ex
pression
is true and zero if it's false. Throughout the video,
we'll use two more terms hard labels or hard predictions, and
soft labels or soft predictions. Usually models output some kind
 of scores. For example, probabilities for
an objects to belong to each class. The scores can be written
as a vector of size L, and I will refer to this vector
as to soft predictions. Now in classification we are usually
asked to predict a label for the object, do a hard prediction. T
o do it, we usually find a maximum
value in the soft predictions, and set class that corresponds to
 this
maximum score as our predicted label. So hard label is
a function of soft labels, it's usually arg max for
multi class tasks, but for binary classification it can be
thought of as a thresholding function. So we output label 1
when the soft score for the class 1 is higher than the threshold
,
and we output class 0 otherwise. Let's start our journey
with the accuracy score. Accuracy is the most straightforward
measure of classifiers quality. It's a value between 0 and 1. Th
e higher, the better. And it is equal to the fraction
of correctly classified objects. To compute accuracy,
we need hard predictions. We need to assign each
object a specific table. Now, what is the best constant
to predict in case of accuracy? Actually, there are a small
number of constants to try. We can only assign a class label
to all the objects at once. So what class should we assign? Obvi
ously, the most frequent one. Then the number of correctly guess
ed
objects will be the highest. But exactly because of that reason,
 there is a caveat in interpreting
the values of the accuracy score. Take a look at this example. S
ay we have 10 cats and
90 dogs in our train set. If we always predicted dog for
every object, then the accuracy would be already 0.9. And imagin
e you tell someone that your
classifier is correct 9 times out of 10. The person would probab
ly
think you have a nice model. But in fact, your model just predic
ts
dog class no matter what input is. So the problem is, that the b
ase
line accuracy can be very high for a data set, even 99%, and tha
t makes
it hard to interpret the results. Although accuracy score is ver
y clean and

intuitive, it turns out to be quite hard to optimize. Accuracy a lso doesn't care how confident
the classifier is in the predictions, and what soft predictions are. It cares only about arg
max of soft predictions. And thus, people sometimes prefer to use different metrics that are first, easier to optimize. And se cond, these metrics work with
soft predictions, not hard ones. One of such metrics is logarith mic loss. It tries to make the classifier to
output two posterior probabilities for their objects to be of a certain kind,
of a certain class. A log loss is usually the reason
a little bit differently for binary and multi class tasks. For b inary, it is assumed that y
hat is a number from 01 range, and it is a probability of
an object to belong to class one. So 1 minus y hat is the probab ility for
this object to be of class 0. For multiclass tasks,
LogLoss is written in this form. Here y hat ith is a vector of s ize L,
and its sum is exactly 1. The elements are the probabilities
to belong to each of the classes. Try to write this formula down for
L equals 2, and you will see it is exactly
binary loss from above. And finally, it should be mentioned
that to avoid in practice, predictions are clipped to
be not from 0 to 1, but from some small positive number to
1 minus some small positive number. Okay, now let us analyze it a little bit. Assume a target for an object is 0,
and here on the plot, we see how the error will change if we change our predictions from 0 to 1. For comparison, we'll plot absolute error with another color. Logloss usually penalizes completely wrong answers and prefers to make a lot of small mistakes to one but severer mistake. Now, what is the best const ant for
logarithmic loss? It turns out that you need to set
predictions to the frequencies of each class in the data set. In our case, the frequencies for the cat class is 0.1, and
it is 0.9 for class dog. Then the best constant is
vector of those two values. How do I, well how do I know that is so? To prove it we should take a derivative
with the respect to constant alpha, set it to 0, and
find alpha from this equation. Okay, we've discussed accuracy an d
log loss, now let's move on. Take a look at the example. We show ground truth target
value with color, and the position of the point
shows the classifier score. Recall that to compute accuracy scor e for
a binary task, we usually take soft predictions
from our model and apply threshold. We can see the prediction to be green
if the score is higher than 0.5 and red if it's lower. For this example the accuracy is 6 or
7, as we misclassified one red object. But look, if the threshol d was 0.7, then all the objects would

be classified correctly. So this is kind of motivation for
our next metric, Area Under Curve. We shouldn't fix the threshold for it, but this metric kind of tries all possible
ones and aggregates those scores. So this metric doesn't really cares about
absolute values of the predictions. But it depends only on
the order of the objects. Actually, there are several ways AUC, or
this area under curve, can be explained. The first one explains under what
curve we should compute area. And the second explains
AUC as the probability of object pairs to be correctly
ordered by our model. We will see both
explanations in the moment. So let's start with the first one. So we need to calculate
an area under a curve. What curve? Let's construct it right now. Once again, say we have six objects, and
their true label is shown with a color. And the position of the dot shows
the classifier's predictions. And for now we will use word positive
as synonym to belongs to the red class. So positive side is on the left. What we will do now, we'll go from left to
right, jump from one object to another. And for
each we will calculate how many red and green dots are there to the left,
to this object that we stand on. The red dots we'll have a name for
them, true positives. And for the green ones we'll
have name false positives. So we will kind of compute
how many true positives and false positives we see to the left
of the object we stand on. Actually it's very simple,
we start from bottom left corner and go up every time we see red point. And right when we see a green one. Let's see. So we stand on the leftmost point first. And it is red, or positive. So we increase the number of
true positives and move up. Next, we jump on the green point. It is false positive, and so we go right. Then two times up for two red points. And finally two times right for
the last green point. We finished in the top right corner. And it always works like that. We start from bottom left and end up in top right corner when
we jump on the right most point. By the way, the curve we've just built
is called Receiver Operating Curve or ROC Curve. And now we are ready to calculate
an area under this curve. The area is seven and we need to normalize
it by the total plural area of the square. So AUC is 7/9, cool. Now what AUC will be for
the data set that can be separated with a threshold,
like in our initial example? Actually AUC will be 1,
maximum value of AUC. So it works. It doesn't need a threshold
to be specified and it doesn't depend on absolute values. Recall that we've never used absolute
values while constructing the curve. Now in practice,

if you build such curve for a huge data set in real classifier, you would observe a picture like that. Here curves for different classifiers are shown with different colors. The curves usually lie above the dashed line which shows how would the curve look like if we made predictions at random. So it kind of shows us a baseline. And note that the area under the dashed line is 0.5. All right, we've seen that we can build a curve and compute area under it. There is another total different explanation for the AUC. Consider all pairs of objects, such that one object is from red class and another one is from green. AUC is a probability that score for the green one will be higher than the score for the red one. In other words, AUC is a fraction of correctly ordered pairs. You see in our example we have two incorrectly ordered pairs and nine pairs in total. And then there are 7 correctly ordered pairs and thus AUC is 7/9. Exactly as we got before, while computing area under the curve. All right, we've discussed how to compute AUC. Now let's think what is the best constant prediction for it. In fact, AUC doesn't depend on the exact values of the predictions. So all constants will lead to the same score and this score will be around 0.5, the baseline. This is actually something that people love about AUC. It is clear what the baseline is. Of course there are flaws in AUC, every metric has some. But still AUC is metric I usually use when no one sets up another one for me. All right, finally let's get to the last metric to discuss, Cohen's Kappa and it's derivatives. Recall that if we always predict the label of the most frequent class, we can already get pretty high accuracy score, and that can be misleading. Actually in our example all the models will fit, will have a score somewhere between 0.9 and 1. So we can introduce a new metric such that for an accuracy of 1 it would give us 1, and for the baseline accuracy it would output 0. And of course, baselines are going to be different for every data, not necessarily 0.9 or whatever. It is also very similar to what r squared does with MSE. It informally saying is kind of normalizes it. So we do the same here. And this is actually already almost Cohen's Kappa. In Cohen's Kappa we take another value as the baseline. We take the higher predictions for the data set and shuffle them, like randomly permute. And then we calculate an accuracy for these shuffled predictions. And that will be our baseline. Well to be precise, we permute and calculate accuracies many times and take, as the baseline, an average for those computed accuracies. In practice, of course,

we do not need to do any permutations. This baseline score can be computed analytically. We need, first, to multiply the empirical frequencies of our predictions and grant those labels for each class, and then sum them up. For example, if we assign 20 cat labels and 80 dog labels at random, then the baseline accuracy will be 0.2*0.1 + 0.8*0.9 = 0.74. You can find more examples in actually. Here I wanted to explain a nice way of thinking about eliminator as a baseline. We can also recall that error is equal to 1 minus accuracy. We could rewrite the formula as 1 minus model's error/baseline error. It will still be Cohen's Kappa, but now, it would be easier to derive weighted Cohen's Kappa. To explain weighted Kappa, we first need to do a step aside, and introduce weighted error. See now we have cats, dogs and tigers to classify. And we are more or less okay if we predict dog instead of cat. But it's undesirable to predict cat or dog if it's really a tiger. So we're going to form a weight matrix where each cell contains The weight for the mistake we might do. In our case, we set error weight to be ten times larger if we predict cat or dog, but the ground truth label is tiger. So with error weight matrix, we can express our preference on the errors that the classifier would make. Now, to calculate weight and error we need another matrix, confusion matrix, for the classifier's prediction. This matrix shows how our classifier distributes the predictions over the objects. For example, the first column indicates that four cats out of ten were recognized correctly, two were classified as dogs and four as tigers. So to get a weighted error score, we need to multiply these two matrices element-wise and sum their results. This formula needs a proper normalization to make sure the quantity is between 0 and 1, but it doesn't matter for our purposes, as the normalization constant will anyway cancel. And finally, weighted kappa is calculated as 1- weighted error / weighted baseline error. In many cases, the weight matrices are defined in a very simple way. For example, for classification problems with ordered labels. Say you need to assign each object a value from 1 to 3. It can be, for instance, a rating of how severe the disease is. And it is not regression, since you do not allow to output values to be somewhere between the ratings and the ground truth values also look more like labels, not as numeric values to predict. So such problems are usually treated as classification problems, but weight matrix is introduced to account for order of the labels. For example, weights can be linear, if we predict two instead of one, we pay one. If we predict three inst

ead of of one,
we pay two. Or the weights can be quadratic,
if we'll predict two instead of one, we still pay one, but if we predict
three instead of one, we now pay for. Depending on what weight matrix is used, we get either linear weighted kappa or
quadratic weighted kappa. The quadratic weighted kappa has been used in several competitions on Kaggle. It is usually explained as
inter-rater agreement coefficient, how much the predictions of the model
agree with ground-truth raters. Which is quite intuitive for medicine applications, how much the model agrees
with professional doctors. Finally, in this video,
we've discussed classification matrix. The accuracy, it is an essential
metric for classification. But a simple model that predicts always
the same value can possibly have a very high accuracy that makes it hard to interpret this metric. The score also depends on the threshold
we choose to convert soft predictions to hard labels. Logloss is another metric, as opposed to accuracy it depends on soft
predictions rather than on hard labels. And it forces the model to predict
probabilities of an object to belong to each class. AUC, area under receiver operating curve,
doesn't depend on the absolute values predicted by the classifier, but
only considers the ordering of the object. It also implicitly tries all the
thresholds to converge soft predictions to hard labels, and thus removes the
dependence of the score on the threshold. Finally, Cohen's Kappa fixes the baseline
for accuracy score to be zero. In spirit it is very
similar to how R-squared beta scales MSE value
to be easier explained. If instead of accuracy we used weighted accuracy, we would get weighted kappa. Weighted kappa with quadratic weights
is called quadratic weighted kappa and commonly used on Kaggle.
[MUSIC]In this video, we will discuss what is the loss and what is a metric, and what is the difference between them. And then we'll overview what are the general approaches to metric optimization. Let's start with a comparison between two notions, loss and metric. The metric or target metric is a function which we want to use to evaluate the quality of our model. For example, for a classification task, we may want to maximize accuracy of our predictions, how frequently the model outputs the correct label. But the problem is that no one really knows how to optimize accuracy efficiently. Instead, people come up with the proxy loss functions. They are such evaluation functions that are easy to optimize for a given model. For example, logarithmic loss is widely used as an optimization loss, while the accuracy score is how the solution is eventually evaluated. So, once again, the loss function is a function that our model optimizes and uses to evalua

te the solution, and the target metric is how we want the soluti on to be evaluated. This is kind of expectation versus reality t hing. Sometimes we are lucky and the model can optimize our targ et metric directly. For example, for mean square error metric, m ost libraries can optimize it from the outset, from the box. So the loss function is the same as the target metric. And sometime s we want to optimize metrics that are really hard or even impos sible to optimize directly. In this case, we usually set the mod el to optimize a loss that is different to a target metric, but after a model is trained, we use hacks and heuristics to negate the discrepancy and adjust the model to better fit the target me tric. We will see the examples for both cases in the following v ideos. And the last thing to mention is that loss metric, cost o bjective and other notions are more or less used as synonyms. It is completely okay to say target loss and optimization metric, but we will fix the wording for the clarity now. Okay, so far, w e've understood why it's important to optimize a metric given in a competition. And we have discussed the difference between opt imization loss and target metric. Now, let's overview the approa ches to target metrics optimization in general. The approaches c an be broadly divided into several categories, depending on the metric we need to optimize. Some metrics can be optimized direct ly. That is, we should just find a model that optimizes this met ric and run it. In fact, all we need to do is to set the model's loss function to these metric. The most common metrics like MSE , Logloss are implemented as loss functions in almost every libr ary. For some of the metrics that cannot be optimized directly, we can somehow pre-process the train set and use a model with a metric or loss function which is easy to optimize. For example, while MSPE metric cannot be optimized directly with XGBoost, we will see later that we can resample the train set and optimize M SE loss instead, which XGBoost can optimize. Sometimes, we'll op timize incorrect metric, but we'll post-process the predictions to fit classification, to fit the communication metric better. F or some models and frameworks, it's possible to define a custom loss function, and sometimes it's possible to implement a loss f unction which will serve as a nice proxy for the desired metric. For example, it can be done for quadratic-weighted Kappa, as we will see later. It's actually quite easy to define a custom los s function for XGBoost. We only need to implement a single funct ion that takes predictions and the target values and computes fi rst and second-order derivatives of the loss function with respe ct to the model's predictions. For example, here you see one for the Logloss. Of course, the loss function should be smooth enou gh and have well-behaved derivatives, otherwise XGBoost will dri ve crazy. In this course, we consider only a small set of metric s, but there are plenty of them in fact. And for some of them, i t is really hard to come up with a neat optimization procedure o r write a custom loss function. Thankfully, there is a method th at always works. It is called early stopping, and it is very sim ple. You set a model to optimize any loss function it can optimi ze and you monitor the desired metric on a validation set. And y ou stop the training when the model starts to fit according to t he desired metric and not according to the metric the model is t ruly optimizing. That is important. Of course, some metrics cann ot be even easily evaluated. For example, if the metric is based

on a human assessor's opinions, you cannot evaluate it on every iteration. For such metrics, we cannot use early stopping, but we will never find such metrics in a competition. So, in this video, we have discussed the discrepancy between our target metric and the loss function that our model optimizes. We've reviewed several approaches to target metric optimization and, in particular, discussed early stopping. In the following videos, we will go through the regression and classification metrics and see the hacks we can use to optimize them.[SOUND] So far we've discussed different metrics, their definitions, and intuition for them. We've studied the difference between optimization loss and target metric. In this video, we'll see how we can efficiently optimize metrics used for regression problems. We've discussed, we always can use earl stopping. So I won't mention it for ever metrics. But keep it in mind. Let's start with mean squared error. It's the most commonly used metric for regression tasks. So we should expect it to be easy to work with. In fact, almost every modelling software will implement MSE as a loss function. So all you need to do to optimize it is to turn this on in your favorite library. And here are some of the library that support mean square error optimization. Both XGBoost and LightGBM will do it easily. A RandomForestRegresor from a scaler and also can split based on MSE, thus optimizing individually. A lot of linear models implemented in siclicar, and most of them are designed to optimize MSE. For example, ordinarily squares, reach regression, regression and so on. There's also SGRegressor class and Sklearn. It also implements a linear model but differently to other linear models in Sklearn. It uses [INAUDIBLE] gradient decent to train it, and thus very versatile. Well and of course MSE was built in. The library for online learning of linear models, also accepts MSC as lost function. But every neural net package like PyTorch, Keras, Flow, has MSE loss implemented. You just need to find an example on GitHub or wherever, and see what name MSE loss has in that particular library. For example, it is sometimes called L two loss, as L to distance in Matt Luke's using. But basically for all the metrics we consider in this lesson, you may find plaintal flames since they were used and discovered independently in different communities. Now, what about mean absolute error. MAE is popular too, so it is easy to find a model that will optimize it. Unfortunately, the extra boost cannot optimize MAE because MAE has zero as a second derivative while LightGBM can. So you still can use gradient boosting

decision trees to this metric. MAE criteria was implemented for RandomForestRegressor from Sklearn. But note that running time will be quite high compared with MSE Corte. Unfortunately, linear models from SKLearn including SG Regressor can not optimize MAE negatively. But, there is a loss called Huber Loss, it is implemented in some of the models. Basically, it is very similar to MAE, especially when the errors are large. We will discuss it in the next slide. In [INAUDIBLE], MAE loss is implemented, but under a different name that's called quantile loss. In fact, MAE is just a special case of quantile loss. Although I will not go into the details here, but just recall that MAE is somehow connected to median values and median is a particular quantile. What about neural networks? As we've discussed MAE is not differentiable only when the predictions are equal to target. And it is of a rare case. That is why we may use any model train to put to optimize MAE. It may be that you will not find MAE implemented in a neural library, but it is very easy to implement it. In fact, all the models need is a loss function gradient with respect to predictions. And in this case, this is just a set function. Different names you may encounter for MAE is, L1 that fit and a one loss, and sometimes people refer to that special case of quintile regression as to median regression. A lot, a lot of, a lot of ways to make MAE smooth. You can actually make up your own smooth function that have upload that loops like MAE error. The most famous one is Huber loss. It's basically a mix between MSE and MAE. MSE is computed when the error is small, so we can safely approach zero error. And MAE is computed for large errors given robustness. So, to this end, we discuss the libraries that can optimize mean square error and mean absolute error. Now, let's get to not ask common relative metrics. MSPE and MAPE. It's much harder to find the model which can optimize them out of the box. Of course we can always can use, either, of course we can always either implement a custom loss for an integer boost or a neural net. It is really easy to do there. Or we can optimize different metric and do early stopping. But there are several specific approaches that I want to mention. This approach is based on the fact that MSP is a weighted version of MSE and MAP is a weighted version of MAE. On the right side, we've sen expression for MSP and MAP. The summon denominator just t ensures that the weights are summed up to 1, but it's not required. Intuitively, the sample weights are

indicating how important the object is for us while training the model. The smaller the target,
is the more important the object. So, how do we use this knowledge? In fact,
many libraries accept sample weights. Say we want to optimize MSP. So if we can set sample weights to
the ones from the previous slide, we can use MSE laws with it. And, the model will actually
optimize desired MSPE loss. Although most important libraries like
XGBoost, LightGBM, most neural net packages support sample weighting,
not every library implements it. But there is another method which works
whenever a library can optimize MSE or MAE. Nothing else is needed. All we need to do is to create a new
training set by sampling it from the original set that we have and
fit a model with, for example, I'm a secretarian if you
want to optimize MSPE. It is important to set
the probabilities for each object to be sampled to
the weights we've calculated. The size of the new data set is up to you. You can sample for example, twice as many
objects as it was in original train set. And note that we do not need to
do anything with the test set. It stays as is. I would also advise you to
re-sample train set several times. Each time fitting a model. And then average models predictions,
if we'll get the score much better and more stable. The results will,
another way we can optimize MSPE, this approach was widely used during
Rossmund Competition on Kagle. It can be proved that if
the errors are small, we can optimize the predictions
in logarithmic scale. Where it is similar to what we will
do on the next slide actually. We will not go into details but you can find a link to explanation
in the reading materials. And finally, let's get to the last
regression metric we have to discuss. Root, mean, square, logarithmic error. It turns out quite easy to optimize,
because of the connection with MSE loss. All we need to do is first to apply and
transform to our target variables. In this case,
logarithm of the target plus one. Let's denote the transformed target
with a z variable right now. And then, we need to fit a model
with MSE loss to transform target. To get a prediction for a test subject,
we first obtain the prediction, z hat, in the logarithmic scale just by calling
model.predict or something like that. And next, we do an inverse transform from
logarithmic scale back to the original by expatiating z hat and subtracting one, and this is how we obtain the predictions
y hat for the test set. In this video, we run through regression

matrix and tools to optimize them. MSE and MAE are very common and
implemented in many packages. RMSPE and MAPE can be optimized by either resampling the data set or setting proper sample weights. RMSLE is optimized by
optimizing MSE in log space. In the next video,
we will see optimization techniques for classification matrix. [MUSIC][MUSIC] In this and the next video,
we will discuss, what are the ways to optimize
classification metrics? In this video,
we will discuss logloss and accuracy, and in the next one, AUC and
quadratic-weighted kappa. Let's start with logloss, logloss for classification is like MSE for
aggression, it is implemented everywhere. All we need to do is to find out what
arguments should be passed to a library to make it use logloss for training. There are a huge number of libraries
to try, like XGBoost, LightGBM, Logistic Regression, and [INAUDIBLE]
classifier from sklearn, Vowpal Wabbit. All neural nets, by default,
optimize logloss for classification. Random forest classifier predictions turn
out to be quite bad in terms of logloss. But there is a way to make them better, we can calibrate the predictions
to better fit logloss. We've mentioned several times that
logloss requires model to output exterior probabilities,
but what does it mean? It actually means that if we take all the points that have a score of, for example, 0.8, then there will be exactly four times
more positive objects than negatives. That is, 80% of the points will be
from class 1, and 20% from class 0. If the classifier doesn't directly optimize logloss, its predictions should be calibrated. Take a look at this plot, the blue line
shows sorted by value predictions for the validation set. And the red line shows correspondent
target values smoothed with rolling window. We clearly see that our predictions
are kind of conservative. They´re much greater than two
target mean on the left side, and much lower than they should be on the right side. So this classifier is not calibrated, and the green curve shows
the predictions after calibration. But if we plot sorted predictions for calibrated classifier, the curve will
be very similar to target rolling mean. And in fact, the calibrator
predictions will have lower log loss. Now, there are several ways to
calibrate predictions, for example, we can use so-called Platt scaling. Basically, we just need to fit a logistic
regression to our predictions. I will not go into the details how to do
that, but it's very similar to how we stack models, and we will discuss

stacking in detail in a different video. Second, we can fit isotonic
regression to our predictions, and again, it is done very similar
to stacking, just another model. While finally, we can use stacking, so the idea is, we can fit any classifier. It doesn't need to optimize logloss,
it just needs to be good, for example, in terms of AUC. And then we can fit another model on top that will take the predictions of our
model, and calibrate them properly. And that model on top will use
logloss as its optimization loss. So it will be optimizing indirectly,
and its predictions will be calibrated. Logloss was the only metric that
is easy to optimize directly. With accuracy, there is no easy
recipe how to directly optimize it. In general, the recipe is following,
actually, if it is a binary classification task, fit any metric, and
tune with the binarization threshold. For multi-class tasks, fit any metric and tune parameters comparing
the models by their accuracy score, not by the metric that the models
were really optimizing. So this is kind of early stopping and the cross validation,
where you look at the accuracy score. Just to get an intuition why accuracy is
hard to optimize, let's look at this plot. So on the vertical axis we
will show the loss, and the horizontal axis shows signed distance
to the decision boundary, for example, to a hyper plane or for a linear model. The distance is considered to be positive
if the class is predicted correctly. And negative if the object is located at
the wrong side of the decision boundary. The blue line here shows zero-one loss, this is the loss that
corresponds to accuracy score. We pay 1 if the object is misclassified,
that is, the object has negative distance,
and we pay nothing otherwise. The problem is that, this loss has zero almost everywhere gradient, with respect to the predictions. And most learning algorithms require
a nonzero gradient to fit, otherwise it's not clear how we need to change the
predictions such that loss is decreased. And so people came up with proxy losses that are upper bounds for
these zero-one loss. So if you perfectly fit the proxy loss,
the accuracy will be perfect too, but differently to zero-one loss,
they are differentiable. For example, you see here logistic loss,
the red curve used in logistic regression, and
hinge loss, loss used in SVM. Now recall that to obtain hard lab

els for
a test object, we usually take argmax of our soft predictions,
picking the class with a maximum score. If our task is binary an
d
soft predictions sum up to 1, argmax is equivalent
to threshold function. Output 1 when the predictions for
the class one is higher than 0.5, and output 0 when the predicti
on's lower. So we've already seen this example
where threshold 0.5 is not optimal, so what can we do? We can tu
ne the threshold we apply, we can do it with a simple grid
search implemented with a for loop. Well, it means that we can b
asically
fit any sufficiently powerful model. It will not matter much wha
t loss exactly,
say, hinge or log loss the model will optimize. All we want from
 our
model's predictions is the existence of a good threshold
that will separate the classes. Also, if our classifier
is ideally calibrated, then it is really returning
posterior probabilities. And for such a classifier,
threshold 0.5 would be optimal, but such classifiers are rarely
the case,
and threshold tuning helps often. So in this video, we discussed
 logloss and accuracy, in the next video
we will discuss AUC and quadratic weighted kappa. [MUSIC]So in t
he previous video, we've discussed Logloss and Accuracy. In this
 video we'll discuss Area Under Curve, AUC, and (Quadratic weigh
ted) Kappa. Let's start with AUC. Although the loss function of
AUC has zero gradients almost everywhere, exactly as accuracy lo
ss, there exists an algorithm to optimize AUC with gradient-base
d methods, and some models implement this algorithm. So we can u
se it by setting the right parameters. I will give you an idea a
bout this method without much details as there is more than one
way to implement it. Recall that originally, classification task
 is usually solved at the level of objects. We want to assign 0
to red objects, and 1 to the green ones. But we do it independen
tly for each object, and so our loss is pointwise. We compute it
 for each object individually, and sum or average the losses for
 all the objects to get a total loss. Now, recall that AUC is th
e probability of a pair of the objects to be ordered in the righ
t way. So ideally, we want predictions Y^ for the green objects
to be larger than for the red ones. So, instead of working with
single objects, we should work with pairs of objects. And instea
d of using pointwise loss, we should use pairwise loss. A pairwi
se loss takes predictions and labels for a pair of objects and c
omputes their loss. Ideally, the loss would be zero when the ord
ering is correct, and greater than zero when the ordering is not
 correct, incorrect. But in practice, different loss functions c
an be used. For example, we can use logloss. We may think that t
he target for this pairwise loss is always one, red minus green
should be one. That is why there is only one term in logloss obj
ective instead of two. The prob function in the formula is neede
d to make sure that the difference between the predictions is st
ill in the 0,1 range, and I use it here just for the sake of sim
plicity. Well, basically, XGBoost, LightGBM have pairwise loss w
e've discussed implemented. It is straightforward to implement i

n any neural net library, and for sure, you can find implementat ions on GitHub. I should say that in practice, most people still use logloss as an optimization loss without any more post proce ssing. I personally observed XGBoost learned with loglosst to gi ve comparable AUC score to the one learned with pairwise loss. A ll right. Now, let's move to the last topic to discuss. It is Qu adratic weighted Kappa metric. There are two methods. One is ver y common and very easy, the second is not that common and will r equire you to implement a custom loss function for either XGBoos t or neural net. But we've already implemented it for XGBoost, s o you will be able to find the implementation in the reading mat erials. But let's start with the simple one. Recall that we're s olving an ordered classification problem and our labels can be f ound of us integer ratings, say from one to five. The task is cl assification as we cannot output, for example, 4.5 as an answer. But anyway, we can treat it as a regression problem, and then s omehow, post-process the predictions and convert them to integer ratings. And actually quadratic weights make Kappa as somehow s imilar to regression with MSE loss. If we allow our predictions to take values between the labels, that is relax the predictions . But in fact, it is different to MSE. So if relaxed, Kappa woul d be one minus MSE divided by something that really depends on t he predictions. And it looks like everyone's logic is, well, the re is MSE in the denominator, we can optimize it, and let's don' t care about denominator. Well, of course it's not correct way t o do it, but it turns out to be useful in practice. But anyway, MSE gives us flat values instead of integers. So now, we need so mehow to convert them into integers. And the straightforward way would be to do rounding all the predictions. But we can think a bout rounding as of applying a threshold. Like if the value is g reater than 3.5 and less than 4.5, then output 3. But then we ca n ask ourselves a question, why do we use exactly those threshol ds? Let's tune them. And again, it's just straightforward, it ca n be easily done with grid search. So to summarize, we need to f it MSE loss to our data and then find appropriate thresholds. Fi nally, there is a paper which suggests a way to relax classifica tion problem to regression, but it deals with this- hard to deal with part in denominator that we had. I will not get into the d etails here, but it's clearly written and easy to understand pap er, so I really encourage you to read it. And more, you can find loss implementation in the reading materials, and just use it i f you don't want to read the paper. Finally, we finished this le sson. We've discussed that evaluation or target metric is how al l submissions are scored. We've discussed the difference between target metric and optimization loss. Optimization loss is what our model optimizes, and it is not always the same as target met ric that we want to optimize. Sometimes, we only can set our mod el to optimize completely different to target metric. But later, we usually try to post-process the predictions to make them bet ter fit target metric. We've discussed intuition behind differen t metrics for regression and classification tasks, and saw how t o efficiently optimize different metrics. I hope you've enjoyed this lesson, and see you later.[MUSIC] Hi, everyone. In this sec tion, we'll cover a very
powerful technique, mean encoding. It actually has a number of n ames. Some call it likelihood encoding,

some target encoding, but in this course,
we'll stick with plain mean encoding. The general idea of this
technique is to add new variables based on some
feature to get where we started,. In simplest case, we encode ea
ch
level of categorical variable with corresponding target mean. Le
t's take a look at
the following example. Here, we have some binary
classification task in which we have a categorical variable, som
e city. And of course,
we want to numerically encode it. The most obvious way and
what people usually use is label encoding. It's what we have in
second column. Mean encoding is done differently, via encoding e
very city with
corresponding mean target. For example, for Moscow, we have
five rows with three 0s and two 1s. So we encode it with 2 divid
ed by 5 or
0.4. Similarly, we deal with the rest
of cities, pretty straightforward. What I've described here
is a very high level idea. There are a huge number of pitfalls o
ne
should overcome in actual competition. We went deep into details
 for
now, just keep it in mind. At first, let me explain. Why does it
 even work? Imagine, that our dataset is much bigger
and contains hundreds of different cities. Well, let's try to co
mpare,
of course, very abstractly, mean encoding with label encoding. W
e plot future histograms for
class 0 and class 1. In case of label encoding,
we'll always get total and random picture because
there's no logical order, but when we use mean target to encode
the
feature, classes look way more separable. The plot looks kind of
 sorted. It turns out that this sorting quality
of mean encoding is quite helpful. Remember, what is the most po
pular and effective way to solve
machine learning problem? Is grading using trees, [INAUDIBLE] OI
GBM. One of the few downsides is
an inability to handle high cardinality categorical variables. T
rees have limited depth,
with mean encoding, we can compensate it, we can reach better lo
ss
with shorter trees. Cross validation loss
might even look like this. In general, the more complicated and
non linear feature target dependency, the more effective is mean
 encoding, okay. Further in this section, you will
learn how to construct mean encodings. There are actually a lot
of ways. Also keep in mind that we use
classification tests only as an example. We can use mathematics
on other tests as well. The main idea remains the same. Despite
the simplicity of the idea, you
need to be very careful with validation. It's got to be impeccab
le. It's probably the most important part. Understanding the cor
rect linkless
validation is also a basis for staking. The last, but not least,

are extensions. There are countless possibilities to derive new features from target variable. Sometimes, they produce significant improvement for your models. Let's start with some characteristics of data sets, that indicate the usefulness of main encoding. The presence of categorical variables with a lot of levels is already a good indicator, but we need to go a little deeper. Let's take a look at each of these learning logs from Springleaf competition. I ran three models with different depths, 7, 9, and 11. Train logs are on the top plot. Validation logs are on the bottom one. As you can see, with increasing the depths of trees, our training care becomes better and better, nearly perfect and that's a normal part. But we don't actually over feed and that's weird. Our validation score also increase, it's a sign that trees need a huge number of splits to extract information from some variables. And we can check it for mortal dump. It turns out that some features have a tremendous amount of split points, like 1200 or 1600 and that's a lot. Our model tries to treat all those categories differently and they are also very important for predicting the target. We can help our model via mean encodings. There is a number of ways to calculate encodings. The first one is the one we've been discussing so far. Simply taking mean of target variable. Another popular option is to take initial logarithm of this value, it's called weight of evidence. Or you can calculate all of the numbers of ones. Or the difference between number of ones and the number of zeros. All of these are variable options. Now, let's actually construct the features. We will do it on sprinkled data set, suppose we've already separated the data for train and validation, X_tr and X val data frames. These called snippet shows how to construct mean encoding for an arbitrary column and map it into a new data frame, train new and val new. We simply do group by on that column and use target as a map. Resulting commands were able [INAUDIBLE]. It is then mapped to tree and validation data sets by a map operator. After we've repeated this process for every call, we can fit each of those model on this new data. But something's definitely not right, after several efforts training AOC is nearly 1, while on validation, the score set rates around 0.55, which is practically noise. It's a clear sign of terrible overfitting. I'll explain what happened in a few moments. Right now, I want to point out that at least we validated correctly. We separated train and validation, and used all the train

data to estimate mean encodings. If, for instance, we would have estimated mean encodings before train validation split, then we would
not notice such an overfitting. Now, let's figure out
the reason of overfitting. When they are categorized, it's pretty
common to get results like in an example, target 0 in train and target 1 in validation. Mean encodings turns into a perfect feature for such categories. That's why we immediately get very good scores on train and fail hardly on validation. So far, we've grasped the concept of mean
encodings and walked through some trivial examples, that obviously can not use
mean encodings like this in practice. We need to deal with overfitting first,
we need some kind of regularization. And I will tell you about different methods in the next video. [MUSIC][MUSIC] In previous video, we realized that
mean encodings cannot be used as is and requires some kind of regularization
on training part of data. Now, we'll carry out four different methods of regularization, namely, doing a cross-validation loop to construct mean encodings. Then, smoothing based on
the size of category. Then, adding random noise. And finally, calculating expanding
mean on some parametrization of data. We will go through all of these methods one by one. Let's start with CV loop regularization. It's a very intuitive and robust method. For a given data point, we don't want to
use target variable of that data point. So we separate the data into
K-node intersecting subsets, or in other words, folds. To get mean an encoding value for
some subset, we don't use data points from that subset and estimate
the encoding only on the rest of subset. We iteratively walk through
all the data subsets. Usually, four or five folds
are enough to get decent results. You don't need to tune this number. It may seem that we have completely
avoided leakage from target variable. Unfortunately, it's not true. It will become apparent if we perform
leave one out scheme to separate the data. I'll return to it a little later, but first let's learn how to
apply this method in practice. Suppose that our training
data is in a DFTR data frame. We will add mean encoded features into another train new data frame. In the outer loop,
we iterate through stratified K-fold iterator in order to separate
training data into chunks. X_tr is used to estimate the encoding. X_val is used to apply
estimating encoding. After that,
we iterate through all the columns and map estimated encodings to X_val data frame. At the end of the outer loop we fill train new data frame with the result. Finally, some rare categories may

be present only in a single fold. So we don't have the data to estimate target mean for them. That's why we end up with some nans. We can fill them with global mean. As you can see, the whole process is very simple. Now, let's return to the question of whether we leak information about target variable or not. Consider the following example. Here we want to encode Moscow via leave-one-out scheme. For the first row, we get 0.5, because there are two 1s and two 0s in the rest of rows. Similarly, for the second row we get 0.25 and so on. But look closely, all the resulting and the resulting features. It perfect splits the data, rows with feature mean equal or greater than 0.5 have target 0 and the rest of rows has target 1. We didn't explicitly use target variable, but our encoding is biased. Furthermore, this effect remains valid even for the KFold scheme, just milder. So is this type of regularization useless? Definitely not. In practice, if you have enough data and use four or five folds, the encodings will work fine with this regularization strategy. Just be careful and use correct validation. Another regularization method is smoothing. It's based on the following idea. If category is big, has a lot of data points, then we can trust this to [INAUDIBLE] encoding, but if category is rare it's the opposite. Formula on the slide uses this idea. It has hyper parameter alpha that controls the amount of regularization. When alpha is zero, we have no regularization, and when alpha approaches infinity everything turns into globalmean. In some sense alpha is equal to the category size we can trust. It's also possible to use some other formula, basically anything that punishes encoding software categories can be considered smoothing. Smoothing obviously won't work on its own but we can combine it with for example, CD loop regularization. Another way to regularize encodence is to add some noise without regularization. Meaning codings have better quality for the [INAUDIBLE] data than for the test data. And by adding noise, we simply degrade the quality of encoding on training data. This method is pretty unstable, it's hard to make it work. The main problem is the amount of noise we need to add. Too much noise will turn the feature into garbage, while too little noise means worse regularization. This method is usually used together with leave one out regularization. You need to diligently fine tune it. So, it's probably not the best option if you don't have a lot of time. The last regularization method I'm going

to cover is based on expanding mean. The idea is very simple. We fix some sorting order of our data and use only rows from zero to n minus
one to calculate encoding for row n. You can check simple implementation
in the code snippet. Cumsum stores cumulative sum
of target variable up to the given row and
cumcnt stores cumulative count. This method introduces the least amount
of leakage from target variable and it requires no hyper parameter tuning. The only downside is that
feature quality is not uniform. But it's not a big deal. We can average models on encodings calculated from
different data permutations. It's also worth noting that
it is expanding mean method that is used in CatBoost grading, boosting to it's library, which proves to perform magnificently on data sets with categorical features. Okay, let's summarize what
we've discussed in this video. We covered four different
types of regularization. Each of them has its own advantages and disadvantages. Sometimes unintuitively we
introduce target variable leakage. But in practice, we can bear with it. Personally, I recommend CV loop or
expanding mean methods for practical tasks. They are the most robust and easy to tune. This is was regularization. In the next video, I will tell
you about various extensions and practical applications of mean encodings. Thank you. [MUSIC][SOUND] In the final video,
we will cover various generalizations and extensions of mean encodings. Namely how to do meaning coding in
regression and multiclass tasks. How can we apply encoding to domains
with many-to-many relations. What features can we build based on target we're able in time series. And finally, how to encode numerical
features and interactions of features. Let's start with regression tasks. They are actually more flexible for
feature encoding. Unlike binary classification where
a mean is frankly the only meaningful statistic we can extract from target variable. In regression tasks, we can try
a variety of statistics, like medium, percentile, standard deviation of target variable. We can even calculate
some distribution bins. For example, if target variable
is distributed between 1 and 100, we can create 10 bin features. In the first feature, we'll count how many
data points have targeted between 1 and 10, in the second between 10 and
20 and so on. Of course,
we need to realize all of these features. In a nutshell, regression tasks are like classification. Just more flexible in terms
of feature engineering. Men encoding for multi-class tasks
is also pretty straightforward. For every feature we want to encode, we will have n different encodings
where n is the number of classes. It actually has non obvious advantage. Three models for example, usually solve multi-class

task in one versus old fashion. So every class had a different m
odel, and when we feed that model, it doesn't
have any information about structure of other classes because th
ey
are merge into one entity. Therefore, together with mean encodin
gs, we introduce some additional information
about the structure of other classes. The domains with many-to-m
any
relations are usually very complex and require special approache
s
to create mean encodings. I will give you only a very high
level idea, consider an example. Binary classification task for
users based
on apps installed on their smartphones. Each user may have multi
ple apps and
each app is used by multiple users. Hence, many-to-many relation
. We want to mean encode apps. The hard part we need to deal wit
h is
that the user may have a lot of apps. So let's take a cross prod
uct of user and
app entities. It will result in a so
called long representation of data. We will have a role for
each user app pair. Using this table, we can naturally
calculate mean encoding for apps. So now every app is encoded wi
th target
mean, but how to map it back to users. Every user has a number o
f apps, so instead of app1, app2, app3, we will now have a vecto
r like 0.1,
0.2, 0.1. That was pretty simple. We can collect various statist
ics
from those vectors, like mean, minimal, maximum,
standard deviation, and so on. So far we assume that our data
has no inner structure, but with time series we can obviously
use future information. On one hand, it's a limitation, on the o
ther hand, it actually allows
us to make some complicated features. In data sets without time
component
when encoding the category, we are forced to use all the rules
to calculate the statistic. It makes no sense to choose
some subset of rules. Presence of time changes it. For a given c
ategory, we can't. For example, calculate the mean from
previous day, previous two days, previous week, etc. Consider an
 example. We need to predict which
categories users spends money. In these two example we have
a period of two days, two users, and three spending categories.
Some good features would be
the total amount of money users spent in previous day. An averag
e amount of money spent
by all users in given category. So, in day 1, user 101 spends $6
, user 102, $3. Therefore, we feel those numbers
as future values for day 2. Similarly, with the average
amount by category. The more data we have, the more
complicated features we can create. In practice, it is often bee
n official
to mean encode numeric features and some combination of features
. To encode a numeric feature, we only need

to bin it and then treat as categorical. Now, we need to answer two questions. First, how to bin numeric feature, and second how to select useful combination of features. Well, we can find it out from a model structure by analyzing the trees. So at first, we take for example, [INAUDIBLE] model and raw features without any encodings. Let's start with numeric features. If numeric feature has a lot of [INAUDIBLE] points, it means that it has some complicated dependency with target and its was trying to mean encode it. Furthermore, these exact split points may be used to bin the feature. So by analyzing model structure, we both identify suspicious numeric feature and found a good way to bin it. It's going to be a little harder with selecting interactions, but nothing extraordinary. First, let's define how to extract to way interaction from decision tree. The process will be similar for three way, four way arbitrary way interactions. So two features interact in a tree if they are in two neighbouring notes. With that in mind, we can iterate through all the trees in the model and calculate how many times each feature interaction appeared. The most frequent interactions are probably worthy of mean encoding. For example, if we found that feature one and feature two pair is most frequent, then we can concatenate that those feature values in our data. And mean encode resulting interaction. Now let me illustrate how important interaction encoding may be. Amazon Employee Access Challenge Competition has a very specific data set. There are only nine categorical features. If we blindly fit say like GBM model on the raw features, then no matter how we return the parameters, we'll score in a 0.87 AUC range. Which will place roughly on 700 position on the leaderboard. Furthermore, even if we mean encode all the labels, we won't have any progress. But if we fit cat boost model, which internally mean encodes some feature interactions, we will immediately score in 0.91 range, which will place us onto win this position. The difference in both absolute AUC values and relative leaderboard positions is tremendous. Also note that cat boost is no silver bullet. In order to get even higher on the leader board, would still need to manually add more mean encoded interactions. In general, if you participate in a competition with a lot of categorical variables, it's always worth trying to work with interactions and mean encodings. I also want to remind you about

correct validation process. During all local experiments, you should at first split data in X_tr and
X_val parts. Estimate encodings on X_tr,
map them to X_tr and X_val, and
then regularize them on X_tr and only after that validate your model on X_tr / X_val split. Don't even think about estimating encodings before splitting the data. And at submission stage, you can
estimate encodings on whole train data. Map it to train and test, then apply regularization on training
data and finally fit a model. And note that you should have already
decided on regularization method and its strength in local experiments. At the end of this section,
let's summarize main advantages and disadvantages of mean encodings. First of all, mean encoding allows us to make a compact
transformation of categorical variables. It is also a powerful basis for
feature engineering. Then the main disadvantage
is target rebel leakage. We need to be very careful with
validation and irregularization. It also works only on specific data sets. It definitely won't help
in every competition. But keep in mind, when this method works,
it may produce significant improvements. Thank you for your attention. [MUSIC]

.

.

[MUSIC] Hi, in this lecture, we will study
hyperparameter optimization process and talk about hyperparamete
rs in
specific libraries and models. We will first discuss
hyperparameter tuning in general. General pipeline, ways to tuni
ng
hyperparameters, and what it actually means to understand how a
particular
hyperparameter influences the model. It is actually what we will
discuss in this video, and then we will talk about libraries and
frameworks, and see how to tune hyperparameters
of several types of models. Namely, we will first
study tree-based models, gradient boosting decision trees and
RandomForest. Then I'll review important
hyperparameters in neural nets. And finally, we will talk about
linear models, where to find them and how to tune them. Another
class of interesting
models is factorization machines. We will not discuss factorizat
ion
machines in this lecture, but I suggest you to read
about them on the internet. So, let's start with a general
discussion of a model tuning process. What are the most importan
t things to
understand when tuning hyperparameters? First, there are tons of
potential
parameters to tune in every model. And so we need to realize whi
ch
parameters are affect the model most. Of course,
all the parameters are reliable, but we kind of need to select
the most important ones. Anyway we never have time to tune
all the params, that's right. So we need to come up with a nice
subset of parameters to tune. Suppose we're new to xgboost and w
e're trying to find out what
parameters will better to tune, and say we don't even understand
how
gradient boosting decision tree works. We always can search what
parameters
people usually set when using xgboost. It's quite easy to look u
p, right? For example, at GitHub or Kaggle Kernels. Finally, the
documentation sometimes
explicitly states which parameter to tune first. From the select
ed set of parameters
we should then understand what would happen if we
change one of the parameters? How the training process and the t
raining
invalidation course will change if we, for example,
increased a certain parameter? And finally, actually tune
the selected parameters, right? Most people do it manually. Just
run, examine the logs,
change parameters, run again and
iterate till good parameters found. It is also possible to use
hyperparameter optimization tools like hyperopt, but it's usuall
y
faster to do it manually to be true. So later in this video, act
ually discuss
the most important parameters for some models along with some in

tuition how
to tune those parameters of those models. But before we start, I actually want
to give you a list of libraries that you can use for
automatic hyperparameter tuning. There are lots of them actually, and
I didn't try everything from this list myself, but from what I actually tried,
I did not notice much difference in optimization speed on
real tasks between the libraries. But if you have time,
you can try every library and compare. From a user side these
libraries are very easy to use. We need first to define the function
that will run our module, in this case, it is XGBoost. That will run our module with
the given set of parameters and return a resulting validation score. And second,
we need to specify a source space. The range for the hyperparameters where
we want to look for the solution. For example, here we see that a parameter,
it is fix 0.1. And we think that optimal max depth
is somewhere between 10 and 30. And actually that is it,
we are ready to run hyperopt. It can take much time, so
the best strategy is to run it overnight. And also please note that everything
we need to know about hyperparameter's, in this case,
is an adequate range for the search. That's pretty convenient,
if you don't know the new model and you just try to run. But still,
most people tuned the models manually. So, what exactly does it mean to understand how parameter influences the model? Broadly speaking, different values of parameters result
in three different fitting behavior. First, a model can underfit. That is, it is so constrained that
it cannot even learn the train set. Another possibility is that the model is so powerful that it just overfits to the train set and
is not able to generalize it all. And finally, the third behavior is something
that we are actually looking for. It's somewhere between underfitting and
overfitting. So basically, what we should examine
while turning parameters is that we should try to understand if the model
is currently underfitting or overfitting. And then, we should somehow
adjust the parameters to get closer to desired behavior. We need to kind of split all the
parameters that we would like to tune into two groups. In the first group, we'll have
the parameters that constrain the model. So if we increase
the parameter from that group, the model would change its behaviour
from overfitting to underfitting. The larger the value of the parameter,

the heavier the constraint. In the following videos, we'll color such
parameters in red, and the parameters in the second group are doing an opposite
thing to our training process. The higher the value,
more powerful the main module. And so by increasing such parameters, we can change fitting behavior
from underfitting to overfitting. We will use green color for
such parameters. So, in this video we'll be discussing some general aspects of
hyperparameter organization. Most importantly,
we've defined the color coding. If you did not understand
what color stands for what, please watch a part of
the video about it again. We'll use this color coding
throughout the following videos. [MUSIC][MUSIC] In this video, we will talk about
hyperparameter optimization for some tree based models. Nowadays, XGBoost and
LightGBM became really gold standard. They are just awesome implementation
of a very versatile gradient boosted decision trees model. There is also a CatBoost library it
appeared exactly at the time when we were preparing this course, so CatBoost
didn't have time to win people's hearts. But it looks very interesting and
promising, so check it out. There is a very nice
implementation of RandomForest and ExtraTrees models sklearn. These models are powerful, and
can be used along with gradient boosting. And finally, there is a model
called regularized Greedy Forest. It showed very nice results from several
competitions, but its implementation is very slow and hard to use, but
you can try it on small data sets. Okay, what important parameters do
we have in XGBoost and LightGBM? The two libraries have similar parameters
and we'll use names from XGBoost. And on the right half of the slide
you will see somehow loosely corresponding parameter
names from LightGBM. To understand the parameters,
we better understand how XGBoost and LightGBM work at least a very high level. What these models do, these models
build decision trees one after another gradually optimizing a given objective. And first there are many parameters
that control the tree building process. Max_depth is the maximum depth of a tree. And of course, the deeper a tree can be
grown the better it can fit a dataset. So increasing this parameter will lead
to faster fitting to the train set. Depending on the task,
the optimal depth can vary a lot, sometimes it is 2, sometimes it is 27. If you increase the depth and can not get
the model to overfit, that is, the model is becoming better and better on the

validation set as you increase the depth. It can be a sign that there are a lot
of important interactions to extract from the data. So it's better to stop tuning and
try to generate some features. I would recommend to start with a max_depth of about seven. Also remember that as
you increase the depth, the learning will take a longer time. So do not set depth to
a very higher values unless you are 100% sure you need it. In LightGBM,
it is possible to control the number of leaves in the tree rather
than the maximum depth. It is nice since a resulting
tree can be very deep, but have small number of leaves and
not over fit. Some simple parameter controls a fraction
of objects to use when feeding a tree. It's a value between 0 and 1. One might think that it's better
always use all the objects, right? But in practice,
it turns out that it's not. Actually, if only a fraction of objects is used at every duration, then the model is less
prone to overfitting. So using a fraction of objects, the model will fit slower on the train set, but at the same time it will probably generalize
better than this over-fitted model. So, it works kind of as a regularization. Similarly, if we can consider only
a fraction of features [INAUDIBLE] split, this is controlled by parameters
colsample_bytree and colsample_bylevel. Once again, if the model is over fitting, you can try to lower
down these parameters. There are also various regularization parameters, min_child_weight, lambda, alpha and others. The most important one
is min_child_weight. If we increase it,
the model will become more conservative. If we set it to 0,
which is the minimum value for this parameter,
the model will be less constrained. In my experience, it's one of the most important parameters
to tune in XGBoost and LightGBM. Depending on the task,
I find optimal values to be 0, 5, 15, 300, so do not hesitate to try a wide
range of values, it depends on the data. To this end we were discussing
hyperparameters that are used to build a tree. And next, there are two very important
parameters that are tightly connected, eta and num_rounds. Eta is essentially a learning weight,
like in gradient descent. And the num_round is the how many learning steps we want to perform or in other words how many tree's we want to build. With each iteration
a new tree is built and added to the model with
a learning rate eta. So in general,
the higher the learning rate, the faster the model fits to the train set
and probably it can lead to over fitting. And more steps model does,
the better the model fits. But there are several caveats here. I

t happens that with a too high learning
rate the model will not fit at all, it will just not converge. So first, we need to find out if we
are using small enough learning rate. On the other hand,
if the learning rate is too small, the model will learn nothing
after a large number of rounds. But at the same time, small learning rate
often leads to a better generalization. So it means that learning
rate should be just right, so that the model generalize and
doesn't take forever to train. The nice thing is that we can freeze
eta to be reasonably small, say, 0.1 or 0.01, and then find how many rounds we
should train the model til it over fits. We usually use early stopping for it. We monitor the validation loss and exit
the training when loss starts to go up. Now when we found
the right number of rounds, we can do a trick that
usually improves the score. We multiply the number of
steps by a factor of alpha and at the same time,
we divide eta by the factor of alpha. For example, we double the number
of steps and divide eta by 2. In this case, the learning will
take twice longer in time, but the resulting model
usually becomes better. It may happen that the valid parameters
will need to be adjusted too, but usually it's okay to leave them as is. Finally, you may want to
use random seed argument, many people recommend to
fix seed before hand. I think it doesn't make too much
sense to fix seed in XGBoost, as anyway every changed parameter will
lead to completely different model. But I would use this
parameter to verify that different random seeds do not
change training results much. Say [INAUDIBLE] competition,
one could jump 1,000 places up or down on the leaderboard just by training
a model with different random seeds. If random seed doesn't
affect model too much, good. In other case, I suggest you to think
one more time if it's a good idea to participate in that competition as
the results can be quite random. Or at least I suggest you to adjust
validation scheme and account for the randomness. All right,
we're finished with gradient boosting. Now let's get to RandomForest and
ExtraTrees. In fact, ExtraTrees is just a more
randomized version of RandomForest and has the same parameters. So I will say RandomForest
meaning both of the models. RandomForest and ExtraBoost build trees,
one tree after another. But, RandomForest builds each
tree to be independent of others. It means that having a lot of trees
doesn't lead to overfeeding for RandomForest as opposed
to gradient boosting. In sklearn, the number of trees for random

forest is controlled
by N_estimators parameter. At the start, we may want to determine what number
of trees is sufficient to have. That is, if we use more than that,
the result will not change much, but the models will fit longer. I usually first set N_estimators
to very small number, say 10, and see how long does it take
to fit 10 trees on that data. If it is not too long then I set
N_estimators to a huge value, say 300, but it actually depends. And feed the model. And then I plot how the validation
error changed depending on a number of used trees. This plot usually looks like that. We have number of trees on the x-axis and
the accuracy score on y-axis. We see here that about 50 trees
already give reasonable score and we don't need to use more
while tuning parameter. It's pretty reliable to use 50 trees. Before submitting to leaderboard, we can set N_estimators to
a higher value just to be sure. You can find code for this plot, actually, in the reading materials. Similarly to XGBoost, there is a parameter max_depth
that controls depth of the trees. But differently to XGBoost, it can be set to none,
which corresponds to unlimited depth. It can be very useful actually when
the features in the data set have repeated values and important interactions. In other cases, the model with unconstrained
depth will over fit immediately. I recommend you to start with a depth
of about 7 for random forest. Usually an optimal depth for
random forests is higher than for gradient boosting, so do not hesitate
to try a depth 10, 20, and higher. Max_features is similar to call
sample parameter from XGBoost. The more features I use to decipher
a split, the faster the training. But on the other hand,
you don't want to use too few features. And min_samples_leaf is a regularization parameter similar to min_child_weight from XGBoost and
the same as min_data_leaf from LightGPM. For Random Forest classifier,
we can select a criterion to eleviate a split in the tree with
a criterion parameter. It can be either Gini or Entropy. To choose one, we should just try both and
pick the best performing one. In my experience Gini is better more
often, but sometimes Entropy wins. We can also fix random seed using
random_state parameter, if we want. And finally, do not forget to set n_jobs
parameter to a number of cores you have. As by default, RandomForest from sklearn
uses only one core for some reason. So in this video, we were talking
about various hyperparameters of gradient boost and
decision trees, and random forest. In the following video, we'll

discuss neural networks and linear models. [MUSIC][MUSIC] In this video we'll briefly discuss neural network libraries and then we'll see how to tune hyperparameters for neural networks and linear models. There are so many frameworks, Keras, TensorFlow, MxNet, PyTorch. The choice is really personal, all frameworks implement more than enough functionality for competition tasks. Keras is for sure the most popular in Kaggle and has very simple interface. It takes only several dozen lines to train a network using Keras. TensorFlow is extensively used by companies for production. And PyTorch is very popular in deep learning research community. I personally recommend you to try PyTorch and Keras as they are most transparent and easy to use frameworks. Now, how do you tune hyperparameters in a network? We'll now talk about only dense neural networks, that is the networks that consist only of fully connected layers. Say we start with a three layer neural network, what do we expect to happen if we increase the number of neurons per layer? The network now can learn more complex decision boundaries and so it will over fit faster. The same should happen when the number of layers are increased, but due to optimization problems, the learning can even stop to converge. But anyway, if you think your network is not powerful enough, you can try to add another layer and see what happens. My recommendation here is to start with something very simple, say 1 or 2 layer and 64 units per layer. Debug the code, make sure the training and [INAUDIBLE] losses go down. And then try to find a configuration that is able to overfit the training set, just as another sanity check. After it, it is time to tune something in the network. One of the crucial parts of neural network is selected optimization method. Broadly speaking, we can pick either vanilla stochastic gradient descent with momentum or one of modern adaptive methods like Adam, Adadelta, Adagrad and so on. On this slide, the adaptive methods are colored in green, as compared to SGD in red. I want to show here that adaptive methods do really allow you to fit the training set faster. But in my experience, they also lead to overfitting. Plain old stochastic gradient descent converges slower, but the trained network usually generalizes better. Adaptive methods are useful, but in the settings others in classification and regression. Now here is a question for you. Just keep the size. What should we expect when increasing a batch size with other hyperparameters fixed? In fact, it turns out that huge batch size leads to more overfitting. Say a batch of 500 objects

is large in experience. I recommend to pick a value around 32 or 64. Then if you see the network is
still overfitting try to decrease the batch size. If it is under fitting, try to increase it. Know that is a the number
of outbox is fixed, then a network with a batch
size reduced by a factor of 2 gets updated twice more times
compared to original network. So take this into consideration. Maybe you need to reduce the number of
networks or at least somehow adjust it. The batch size also should not be too
small, the gradient will be too noisy. Same as in gradient boosting,
we need to set the proper learning rate. When the learning rate is too high,
network will not converge and with too small a learning rate,
the network will learn forever. The learning rate should be
not too high and not too low. So the optimal learning rate
depends on the other parameters. I usually start with a huge learning rate,
say 0.1, and try to lower it down till I find one with which network converges
and then I try to revise further. Interestingly, there is a connection
between the batch size and the learning rate. It is theoretically grounded for
a specific type of models, but people usually use it,
well actually some people use it as a rule of thumb with neural networks. The connection is the following. If you increase the batch
size by a factor of alpha, you can also increase the learning
rate by the same factor. But remember that the larger batch size, the more your network is
prone to overfitting. So you need a good regularization here. Sometime ago, people mostly use L2 and
L1 regularization for weights. Nowadays, most people use
dropout regularization. So whenever you see a network overfitting,
try first to a dropout layer. You can override dropout probability and a
place where you insert the dropout layer. Usually people add the dropout layer
closer to the end of the network, but it's okay to add some dropout
to every layer, it also works. Dropout helps network to find features
that really matters, and what never worked for me is to have dropout as the very
first layer, immediately after data layer. This way some information is lost
completely at the very beginning of the network and
we observe performance degradation. An interesting regularization
technique that we used in the [UNKOWN] competition is
static dropconnect, as we call it. So recall that, usually we have an input
layer densely connected to, say 128 units. We will instead use a

first
hidden layer with a very huge number of units, say 4,096 units.
This is a huge network for a usual
competition and it will overfeed badly. But now to regularlize it,
we'll at random drop 99% of connections between the input layer and
the first hidden layer. We call it static dropconnect
because originally in dropconnect, we need to drop random connections at
every learning iterations while we fix connectivity pattern for the network
for the whole learning process. So you see the point, we increase
the number of hidden units, but the number of parameters in the first
hidden layer remains small. Notice that anyway the weight matrix
of the second layer becomes huge, but it turns out to be
okay in the practice. This is very powerful regularizations. And more of the networks with
different connectivity patterns makes much nicer than networks
without static dropconnect. All right, last class of models
to discuss are my neuro models. Yet, a carefully tuned live GPM would
probably beat support vector machines, even on a large, sparse data set. SVM's do not require almost any tuning,
which is truly beneficial. SVM's for classification and regression
are implemented in SK learners or wrappers to algorithms from libraries
called libLinear and libSVM. The latest version of libLinear and
libSVM support multicore competitions, but unfortunately it is not possible
to use multicore version in Sklearn, so we need to compile these libraries
manually to use this option. And I've never had anyone
use kernel SVC lately, so in this video we will
talk only about linear SVM. In Sklearn we can also find logistic and
linear regression with various regularization options and also,
as your declassifier and regressor. We've already mentioned them
while discussing metrics. For the data sets that do not fit in
the memory, we can use Vowpal Wabbit. It implements learning of linear
models in online fashion. It only reads data row by row
directly from the hard drive and never loads the whole
data set in the memory. Thus, allowing to learn
on a very huge data sets. A method of online learning for linear
models called flow the regularized leader or FTRL in short was particularly
popular some time ago. It is implemented in Vowpal Wabbit but there are also lots of
implementations in pure Python. The hyperparameters we usually need
to tune linear models are L2 and L1 regularization of weights. Once again, regularization is denoted

with red color because of the higher the regularization weight is the more
model struggle to learn something. But know that, the parameter see in
SVM is inversely proportional to regularization weight, so
the dynamics is opposite. In fact, we do not need to think about the
meaning of the parameters in the case of one parameter, right? We just try several values and
find one that works best. For SVM, I usually start a very small
seed, say 10 to the power of minus 6 and then I try to increase it,
multiply each time by a factor of 10. I start from small values because
the larger the parameter C is, the longer the training takes. What type of regularization,
L1 or L2 do you choose? Actually, my answer is try both. To my mind actually they are quite similar
and one benefit that L1 can give us is weight sparsity, so the sparsity
pattern can be used for feature selection. A general advise I want to give
here do not spend too much time on tuning hyperparameters, especially
when the competition has only begun. You cannot win a competition
by tuning parameters. Appropriate features, hacks,
leaks, and insights will give you much more than carefully
tuned model built on default features. I also advice you to be patient. It was my personal mistake several times. I hated to spend more then ten minutes
on training models and was amazed how much the models could improve if I
would let it train for a longer time. And finally, average everything. When submitting, learn five models
starting from different random initializations and
average their predictions. It helps a lot actually and
some people average not only random seed, but also other parameters
around an optimal value. For example, if optimal depth for
extra boost is 5, we can average 3 digiboosts with depth 3,
4, and 5. Wow, it would be better if we could averaged 3 digiboosts with depth 4,
5, and 6. Finally, in this lecture, we discussed what is a general pipeline
for a hyperparameter optimization. And we saw, in particular,
what important hyperparameters derive for several models,
gradient boosting decision trees, random forests and extra trees,
neural networks, and linear models. I hope you found something interesting
in this lecture and see you later. [MUSIC][SOUND] Hi, to this moment,
we have already discussed all basics new things which build up to
a big solution like featured generation, validation, minimalist

codings and so on. We went through several
competitions together and tried our best to unite everything
we learn into one huge framework. But as with any other set of t
ools,
there are a lot of heuristics which people often find only with
a trial and
error approach, spending significant time on learning
how to use these tools efficiently. So to help you out here, in
this video we'll share things we
learned the hard way, by experience. These things may vary from
one person to another. So we decided that everyone on class will
present his own guidelines personally, to stress the possible
diversity in a broad issues and to make an accent on different m
oments. Some notes might seem obvious to you,
some may not. But be sure for even some of them or
at least no one involve them. Can save you a lot of time. So, le
t's start. When we want to enter a competition,
define your goals and try to estimate what you can
get out of your participation. You may want to learn more
about an interesting problem. You may want to get acquainted
with new software tools and packages, or
you may want to try to hunt for a medal. Each of these goals wil
l influence what
competition you choose to participate in. If you want to learn m
ore
about an interesting problem, you may want the competition to ha
ve
a wide discussion on the forums. For example, if you are interes
ted in
data science, in application to medicine, you can try to predict
 lung cancer
in the Data Science Bowl 2017. Or to predict seizures in long
term human EEG recordings. In the Melbourne University
Seizure Prediction Competition. If you want to get acquainted
with new software tools, you may want the competition
to have required tutorials. For example, if you want to
learn a neural networks library. You may choose any of competiti
ons with
images like the nature conservancy features, monitoring competit
ion. Or the planet, understanding
the Amazon from space competition. And if you want to try to hun
t for a medal, you may want to check how
many submissions do participants have. And if the points that pe
ople have
over one hundred submissions, it can be a clear sign of legible
problem or difficulties in validation includes an inconsistency
of
validation and leaderboard scores. On the other hand, if there a
re people
with few submissions in the top, that usually means there should
 be
a non-trivial approach to this competition or it's discovered on
ly by few people. Beside that, you may want to pay
attention to the size of the top teams. If leaderboard mostly co
nsists of
teams with only one participant, you'll probably have enough

chances if you gather a good team. Now, let's move to the next s
tep
after you chose a competition. As soon as you get familiar with
the data, start to write down your ideas about
what you may want to try later. What things could work here? Wha
t approaches you may want to take. After you're done, read forum
s and
highlight interesting posts and topics. Remember, you can get a
lot of information
and meet new people on forums. So I strongly encourage you to
participate in these discussions. After the initial pipeline is
ready and you roll down few ideas, you may want
to start improving your solution. Personally, I like to organize
these ideas into some structure. So you may want to sort
ideas into priority order. Most important and
promising needs to be implemented first. Or you may want to orga
nize
these ideas into topics. Ideas about feature generation,
validation, metric optimization. And so on. Now pick up an idea
and implement it. Try to derive some insights on the way. Especi
ally, try to understand why
something does or doesn't work. For example, you have an idea ab
out trying a deep
gradient boosting decision tree model. To your joy, it works. No
w, ask yourself why? Is there some hidden data
structure we didn't notice before? Maybe you have categorical fe
atures
with a lot of unique values. If this is the case, you as well ca
n make a conclusion that
mean encodings may work great here. So in some sense,
the ability to analyze the work and derive conclusions while
you're trying out your ideas will get you on the right track to
reveal hidden data patterns and leaks. After we checked out most
 important ideas, you may want to switch
to parameter training. I personally like the view,
everything is a parameter. From the number of features, through
gradient boosting decision through depth. From the number of lay
ers in
convolutional neural network, to the coefficient you finally
submit is multiplied by. To understand what I should tune and ch
ange first, I like to sort all
parameters by these principles. First, importance. Arrange param
eters from
important to not useful at all. Tune in this order. These may de
pend on data structure,
on target, on metric, and so on. Second, feasibility. Rate param
eters from, it is easy to tune,
to, tuning this can take forever. Third, understanding. Rate par
ameters from, I know what
it's doing, to, I have no idea. Here it is important to understa
nd
what each parameter will change in the whole pipeline. For examp
le, if you increase
the number of features significantly, you may want to change rat
io of columns
which is used to find the best split in gradient boosting decisi

on tree. Or, if you change number of layers
in convolution neural network, you will need more reports to tra
in it,
and so on. So let's see, these were some
of my practical guidelines, I hope they will prove useful for
you as well. Every problem starts with data loading and
preprocessing. I usually don't pay much attention to
some sub optimal usage of computational resources but this parti
cular
case is of crucial importance. Doing things right at the very be
ginning
will make your life much simpler and will allow you to save a lo
t of time and
computational resources. I usually start with basic data
preprocessing like labeling, coding, category recovery,
both enjoying additional data. Then, I dump resulting data into
HDF5 or
MPI format. HDF5 for Panda's dataframes,
and MPI for non bit arrays. Running experiment often require
a lot of kernel restarts, which leads to reloading all the data.
 And loading class CSC files
may take minutes while loading data from HDF5 or MPI formats
is performed in a matter of seconds. Another important matter is
 that by
default, Panda is known to store data in 64-bit arrays, which is
unnecessary in most of the situations. Downcasting everything to
 32 bits will
result in two-fold memory saving. Also keep in mind that Panda's
 support
out of the box data relink by chunks, via chunks ice parameter
in recess fee function. So most of the data sets may be
processed without a lot of memory. When it comes to performance
evaluation, I
am not a big fan of extensive validation. Even for medium-sized
datasets
like 50,000 or 100,000 rows. You can validate your models
with a simple train test split instead of full cross validation
loop. Switch to full CV only
when it is really needed. For example,
when you've already hit some limits and can move forward only wi
th
some marginal improvements. Same logic applies to
initial model choice. I usually start with LightGBM,
find some reasonably good parameters, and evaluate performance o
f my features. I want to emphasize that
I use early stopping, so I don't need to tune number
of boosting iterations. And god forbid start ESVMs,
random forks, or neural networks, you will waste too
much time just waiting for them to feed. I switch to tuning the
models,
and sampling, and staking, only when I am satisfied
with feature engineering. In some ways, I describe my approach a
s,
fast and dirty, always better. Try focusing on what is really im
portant,
the data. Do ED, try different features. Google domain-specific

knowledge. Your code is secondary. Creating unnecessary classes and personal frame box may only make
things harder to change and will result in wasting your time, so keep things simple and reasonable. Don't track every little change. By the end of competition, I usually
have only a couple of notebooks for model training and to want notebooks
specifically for EDA purposes. Finally, if you feel really uncomfortable
with given computational resources, don't struggle for weeks, just rent a larger server. Every competition I start with
a very simple basic solution that can be even primitive. The main purpose of such solution
is not to build a good model but to debug full pipeline from very beginning of the data to the very end when we write
the submit file into decided format. I advise you to start with construction of the initial pipeline. Often you can find it in baseline
solutions provided by organizers or in kernels. I encourage you to read carefully and
write your own. Also I advise you to follow from simple
to complex approach in other things. For example, I prefer to start
with Random Forest rather than Gradient Boosted Decision Trees. At least Random Forest
works quite fast and requires almost no tuning
of hybrid parameters. Participation in data science competition implies the analysis of data and generation of features and manipulations with models. This process is very similar in spirit
to the development of software and there are many good practices that I advise you to follow. I will name just a few of them. First of all, use good variable names. No matter how ingenious you are,
if your code is written badly, you will surely get confused in it and
you'll have a problem sooner or later. Second, keep your research reproducible. FIx all random seeds. Write down exactly how
a feature was generated, and store the code under version control system like git. Very often there are situation when you need to go back to the model that you built two weeks ago and edit to the ensemble width. The last and probably the most important thing, reuse your code. It's really important to use the same
code at training and testing stages. For example, features should be prepared
and transforming by the same code in order to guarantee that they're
produced in a consistent manner. Here in such places are very difficult to catch, so it's better to be very careful with it. I recommend to move reusable code into
separate functions or even separate model. In addition, I advise you to read
scientific articles on the topic of the competition. They can provide you with
information about machine and correlated things like for example

how
to better optimize a measure, or AUC. Or, provide the main
knowledge of the problem. This is often very useful for
future generations. For example, during Microsoft Mobile
competition, I read article about mobile detection and used idea
s from
them to generate new features. >> I usually start the competitio
n by
monitoring the forums and kernels. It happens that a competition
starts,
someone finds a bug in the data. And the competition data is
then completely changed, so I never join a competition
at its very beginning. I usually start a competition with
a quick EDA and a simple baseline. I tried to check the data for
various leakages. For me, the leaks are one of the most
interesting parts in the competition. I then usually do several
submissions
to check if validation score correlates with publicly the board
score. Usually, I try to come up with a list
of things to try in a competition, and I more or less try to fol
low it. But sometimes I just try to generate
as many features as possible, put them in extra boost and
study what helps and what does not. When tuning parameters,
I first try to make model overfit to the training set and only t
hen I
change parameters to constrain the model. I had situations when
I could not
reproduce one of my submissions. I accidentally changed somethin
g in
the code and I could not remember what exactly, so nowadays I'm
very
careful about my code and script. Another problem? Long executio
n history in notebooks leads
to lots of defined global variables. And global variables surely
lead to bugs. So remember to sometimes
restart your notebooks. It's okay to have ugly code, unless you
do not use this to produce a submission. It would be easier for
you to get into this code later if
it has a descriptive variable names. I always use git and
try to make the code for submissions as transparent as possible.
I usually create a separate notebook for every submission so I
can always run
the previous solution and compare. And I treat the submission
notebooks as script. I restart the kernel and
always run them from top to bottom. I found a convenient way to
validate
the models that allows to use validation code with minimal chang
es to retrain
a model on the whole dataset. In the competition, we are provide
d
with training and test CSV files. You see we load them in the fi
rst cell. In the second cell, we split
training set and actual training and validation sets, and
save those to disk as CSV files with the same structure as given
train CSV and
test CSV. Now, at the top of the notebook,

with my model, I define variables. Path is to train and test sets. I set them to create a training and validation sets when working with
the model and validating it. And then it only takes me to switch those paths to original train CSV and test CSV to produce a submission. I also use macros. At one point I was really tired of typing import numpy as np, every time. So I found that it's possible to define
a macro which will load everything for me. In my case, it takes only five symbols to type the macro name and this macro immediately loads me everything. Very convenient. And finally, I have developed my library
with frequently used functions, and training code for models. I personally find it useful, as the code,
it now becomes much shorter, and I do not need to remember how to import a particular model. In my case I just specify a model with its name, and as an output I get all the information
about training that I would possibly need. [SOUND] [MUSIC]Hello everyone. This is Marios. Today I would like to show you the Pipeline or like the approach I have used to tackle more than 100 machine learning competitions in cargo and obviously has helped me to do quite well. Before I start, let me state that I'm not claiming this is the best pipeline out there, is just the one I use. You might find some parts of it useful. So roughly, the Pipeline is, as you see it on the screen, here this is a summary and we will go through it in more detail later on. But briefly, I spend the first day in order to understand the problem and make the necessary preparations in order to deal with this. Then, maybe one, two days in order to understand a bit about the data, what are my features, what I have available, trying to understand other dynamics about the data, which will lead me to define a good cross validation strategy and we will see later why this is important. And then, once I have specified the cross validation strategy, I will spend all the days until 3-4 days before the end of the competition and I will keep iterating, doing different feature engineering and applying different machine bearing models. Now, something that I need to to highlight is that, when I start this process I do it myself, shut from the outside world. So, I close my ears, and I just focus on how I would tackle this problem. That's because I don't want to get affected by what the others are doing. Because I might be able to find something that others will not. I mean, I might take a completely different approach and this always leads me to gain, when I then combine with the rest of the people. For example, through merges or when I use other people's kernels. So, I think this is important, because it gives you the chance to create an intuitive approach about the data, and then also leverage the fact that other people have different approaches and you will get more diverse results. And in the last 3 to 4 days, I would start exploring different ways to combine all the models of field, in order to get the best results. Now, if people have seen me in competitions, you should know that you might have noticed that in the last 3-2 days I do a rapid jump in the little box and that's exactly because I leave assembling at the end. I normally don't do it. I have confidence that it will work and I spend more time in feature engineering a

nd modeling, up until this point. So, let's take all these steps one by one. Initially I try to understand the problem. First of all, what type of problem it is. Is it image classification, so try to find what object is presented on an image. This is sound classification, like which type of bird appears in a sound file. Is it text classification? Like who has written the specific text, or what this text is about. Is it an optimization problem, like giving some constraints how can I get from point A to point B etc. Is it a tabular dataset, so that's like data which you can represent in Excel for example, with rows and columns, with various types of features, categorical or numerical. Is it time series problem? Is time important? All these questions are very very important and that's why I look at the dataset and I try to understand, because it defines in many ways what resources I would need, where do I need to look and what kind of hardware and software I will need. Also, I do this sort of preparation along with controlling the volume of the data. How much is it. Because again, this will define how I need to, what preparations I need to do in order to solve this problem. Once I understand what type of problem it is, then I need to reserve hardware to solve this. So, in many cases I can escape without using GPUs, so just a few CPUs would do the trick. But in problems like image classification of sound, then generally anywhere you would need to use deep learning. You definitely need to invest a lot in CPU, RAM and disk space. So, that's why this screening is important. It will make me understand what type of machine I will need in order to solve this and whether I have this processing power at this point in order to solve this. Once this has been specified and I know how many CPUs, GPUs, RAM and disk space I'm going to need, then I need to prepare the software. So, different software is suited for different types of problems. Keras and TensorFlow is obviously really good for when solving an image classification or sound classification and text problems that you can pretty much use it in any other problem as well. Then you most probably if you use Python, you need scikit learn and XGBoost, Lighgbm. This is the pinnacle of machine learning right now. And how do I set this up? Normally I create either an anaconda environment or a virtual environment in general, and how a different one for its competition, because it's easy to set this up. So, you just set this up, you download the necessary packages you need, and then you're good to go. This is a good way, a clean way to keep everything tidy and to really know what you used and what you find useful in the particular competitions. It's also a good validation for later on, when we will have to do this again, to find an environment that has worked well for this type of problem and possibly reuse it. Another question I ask at this point is what the metric I'm being tested on. Again, is it a regression program, is it a classification program, it is root mean square error, it is mean absolute error. I ask these questions because I try to find if there's any similar competition with similar type of data that I may have dealt with in the past, because this will make this preparation much much better, because I'll go backwards, find what I had used in the past and capitalize on it. So, reuse it, improve it, or even if I don't have something myself, I can just find other similar competitions or explanations of these type of problem from the web and try to see what people had used in

order to integrate it to my approaches. So, this is what it means to understand the problem at this point. It's more about doing the screen search, this screening in order to understand what type of preparation I need to do, and actually do this preparation, in order to be able to solve this problem competitively, in terms of hardware, software and other resources, past resources in dealing with these types of problems. Then I spent the next one or two days to do some exploratory data analysis. The first thing that I do is I see all my features, assuming a tabular data set, in the training and the test data and to see how consistent they are. I tend to plot distributions and to try to find if there are any discrepancies. So is this variable in the training data set very different than the same variable in the task set? Because if there are discrepancies or differences, this is something I have to deal with. Maybe I need to remove these variables or scale them in a specific way. In any case, big discrepancies can cause problems to the model, so that's why I spend some time here and do some plotting in order to detect these differences. The other thing that I do is I tend to plot features versus the target variable and possibly versus time, if time is available. And again, this tells me to understand the effect of time, how important is time or date in this data set. And at the same time it helps me to understand which are like the most predictive inputs, the most predictive variables. This is important because it generally gives me intuition about the problem. How exactly this helps me is not always clear. Sometimes it may help me define a gross validation strategy or help me create some really good features but in general, this kind of knowledge really helps to understand the problem. I tend to create cross tabs for example with the categorical variables and the target variable and also creates unpredictability metrics like information value and you see chi square for example, in order to see what's useful and whether I can make hypothesis about the data, whether I understand the data and how they relate with the target variable. The more understanding I create at this point, most probably will lead to better features for better models applied on this data. Also while I do this, I like to bin numerical features into bands in order to understand if there nonlinear R.A.T's. When I say nonlinear R.A.T's, whether the value of a feature is low, target variable is high, then as the value increases the target variable decreases as well. So whether there are strange relationships trends, patterns, or correlations between features and the target variable, in order to see how best to handle this later on and get an intuition about which type of problems or which type of models would work better. Once I have understood the data, to some extent, then it's time for me to define a cross validation strategy. I think this is a really important step and there have been competitions where people were able to win just because they were able to find the best way to validate or to create a good cross validation strategy. And by cross validation strategy, I mean to create a validation approach that best resembles what you're being tested on. If you manage to create this internally then you can create many different models and create many different features and anything you do, you can have the confidence that is working or it's not working, if you've managed to build the cross validation strategy in a consistent way with what you're being t

ested on so consistency is the key word here. The first thing I ask is, "Is time important in this data?" So do I have a feature which is called date or time? If this is important then I need to switch to a time-based validation. Always have past data predicting future data, and even the intervals, they need to be similar with the test data. So if the test data is three months in the future, I need to build my training and validation to account for this time interval. So my validation data always need to be three months in the future and compared to the training data. You need to be consistent in order to have the most consistent results with what you are been tested on. The other thing that I ask is, "Are there different entities between the train and the test data?" Imagine if you have different customers in the training data and different in the test data. Ideally, you need to formulate your cross validation strategy so that in the validation data, you always have different customers running in training data otherwise you are not really testing in a fair way. Your validation method would not be consistent with the test data. Obviously, if you know a customer and you try to predict it, him or her, why you have that customer in your training data, this is a biased prediction when compared to the test data, that you don't have this information available. And this is the type of questions you need to ask yourself when you are at this point, "Am I making a validation which is really consistent with what am I being tested on?" The other thing that is often the case is that the training and the test data are completely random. I'm sorry, I just shortened my data and I took a random part, put it on training, the other for test so in that case, is any random type of cross validation could help for example, just do a random K-fold. There are cases where you may have to use a combination of all the above so you have strong temporal elements at the same time you have different entities, so different customers to predict for past and future and at the same time, there is a random element too. You might need to incorporate all of them do make a good strategy. What I do is I often start with a random validation and just see how it fares with the test leader board, and see how consistent the result is with what they have internally, and see if improvements in my validation lead to improvements to the leader board. If that doesn't happen, I make a deeper investigation and try to understand why. It may be that the time element is very strong and I need to take it into account or there are different entities between the train and test data. These kinds of questions in order to formulate a better validation strategy. Once the validation strategy has been defined, now I start creating many different features. I'm sorry for bombarding you with loads of information in one slide but I wanted this to be standalone. It says give you the different type of future engineering you can use in different types of problems, and also suggestions for the competition to look up which was quite representative of this time. But you can ignore these for now. Look at it later. The main point is different problem requires different feature engineering and I put everything when I say feature engineering. I put the day data cleaning and preparation as well, how you handle missing values, and the features you generate out of this. The thing is, every problem has its own corpus of different techniques you use to derive or create new features. It's not easy to k

now everything because sometimes it's too much, I don't remember it myself so what I tend to do is go back to similar competitions and see what people are using or what people have used in the past and I incorporate into my code. If I have dealt with this or a similar problem in the past then I look at my code to see what I had done in the past, but still looking for ways to improve this. I think that's the best way to be able to handle any problem. The good thing is that a lot of the feature engineering can be automated. You probably have already seen that but, as long as your cross validation strategy is consistent with the test data and reliable, then you can potentially try all sorts of transformations and see how they work in your validation environment. If they work well, you can be confident that this type of feature engineering is useful and use it for further modeling. If not, you discard and try something else. Also the combinations of what you can do in terms of feature engineering can be quite vast in different types of problems so obviously time is a factor here, and scalability too. You need to be able to use your resources well in order to be able to search as much as you can in order to get the best outcome. This is what I do. Normally if I have more time to do this feature engineering in a competition, I tend to do better because I explore more things.And the modeling is pretty much the same story. So, it's type problem has its own type of model that works best. Now, I don't want to go through that list again, I put it here so that you can use it for reference. But, again, the way you work this out is you look for literature, you sense other previous competitions that were similar and you try to find which type of problem, which type of model or best for its type of problem. And it's not surprise that for typical dataset, when I say typical dataset I mean, tabular dataset rather boosting machines in the form of [inaudible] turned to rock fest for problems like aim as classification sound classification, deep learning in the form of convolutional neural networks tend to work better. So, this is roughly what you need to know. New techniques are being developed so, I think your best chance here or what I have used in order to do well in the past was knowing what's tends to work well with its problem, and going backwards and trying to find other code or other implementations and similar problems in order to integrate it with mine and try to get a better result. I should mention that each of the previous models needs to be changed sometimes differently. So you need to spend time within this cross-validation strategy in order to find the best parameters, and then we move onto Ensembling. Every time you apply your cross-validation procedure with a different feature engineering and a different joint model, it's time, you saved two types of predictions, you save predictions for the validation data and you save predictions for the test data. So now that you have saved all these predictions and by the way this is the point that if you collaborate with others that tend to send you the predictions, and you'll be surprised that sometime that collaboration is just this. So people just sending these prediction files for the validation and the test data. So now you can find the best way to combine these models in order to get the best results. And since you already have predictions for the validation data, you know the target variable for the validation data, so you can explore different ways to combine them. The metho

ds could be simple, could be an average, or already average, or it can go up to a multilayer stacking in general. Generally, what you need to know is that from my experience, smaller data requires simple ensemble techniques like averaging. And also what tends to show is to look at correlation between predictions. So find it here that work well, but they tend to be quite diverse. So, when you use fusion correlation, the correlation is not very high. That means they are likely to bring new information, and so when you combine you get the most out of it. But if you have bigger data there are, you got pretty must try all sorts of things. What I like to think of is it is that, when you have really big data, the stacking process that impedes the modeling process. By that, I mean that you have a new set of features this time they are predictions of models, but you can apply the same process you have used before. So you can do feature engineering, you can create new features or you can remove the features/ prediction that you no longer need and you can use this in order to improve the results for your validation data. This process can be quite exhaustive, but well, again, it can be automated to some extent. So, the more time you have here, most probably the better you will do. But from my experience, 2, 3 days is good in order to get the best out of all the models you have built and depends obviously on the volume of data or volume of predictions you have generated up until this point. At this point I would like to share a few thoughts about collaboration. Many people have asked me this and I think this is a good point to share. These ideas has greatly helped me to do well in competitions. The first thing is that it makes things more fun. I mean you are not alone, you're with other people and that's always more energizing, it's always more interesting, it's more fun, you can communicate with the others through times like Skype, and yeah I think it's more collaborative as the world says, it is better. You learn more. I mean you can be really good, but, you know, you always always learn from others. No way to know everything yourself. So it's really good to be able to share points with other people, see what they do learn from them and become better and grow as a data scientist, as a model. From my experience you score far better than trying to solve a problem alone, and I think these happens for mainly for two ways. There are more but these are main two. First you can cover more ground because, you can say, you can focus on ensembling, I will focus on feature engineering or you will focus on joining this type of model and I will focus on another type of model. So, you can generally cover more ground. You can divide task and you can search, you can cover more ground in terms of the possible things you can try in a competition. The second thing is that every person sees the problem from different angles. So, that's very likely to generate more diverse predictions. So something we do is although we kind of define together by the different strategy when we form teams, then we would like to work for maybe one week separately without discussing with one another, because this helps to create diversity. Otherwise, if we over discuss this, we might generate pretty much the same things. So, in other words, our solutions might be too correlated to add more value. So, this is a good way in order to leverage the different mindset each person has in solving these problems. So, for one week, each one works separately and then after some point

, we start combining or work more closely. I would advise people to start collaborating after getting some experience, and I say here two or three competitions just because Cargo has some rules. Sometimes, it is easy to make mistakes. I think it's better to understand the environment, the competition environment well before exploring these options in order to make certain that, no mistakes are done, no violation of the rules. Sometimes new people tend to make these mistakes. So, it's good to have this experience prior to trying to collaborating. I advise people to start forming teams with people around their rank because sometimes it is frustrating when you join a high rank or a very experienced team I would say. It's bad to say experience from rank, because you don't know sometimes how to contribute, you still don't understand all the competition dynamics and it might stall your progress, if you join a team and you're not able to contribute. So, I think it's better to, in most cases, to try and find people around your rank or around your experience and grow together. This way is the best form of collaboration I think. Another tip for collaborating is to try to collaborate with people that are likely to take diverse approaches or different approaches than yourself. You learn more this way and it is more likely that when you combine, you will get a better score. So, such for people who are sort of famous for doing well certain things and in order to get the most out of it, to learn more from each other and get better results in the leader board. About selecting submissions, I have employed a strategy that many people have done. So normally, I select the best submissions I see in my internal result and the one that work best on the leader board. At the same time, I also look for correlations. So, if two submissions, they tend to be the same pretty much. So, the one that was the best submission locally, was also the best on leader boards, I try to find other submissions that still work well but they are likely to be quite diverse. So, they have low correlations with my best submission because this way, I might capture, I might be lucky, it maybe be a special type of test data set and just by having a diverse submission, I might be lucky to get a good score. So that's the main idea about this. Some tips I would like to share now in general about competitive modeling, on land modeling and in Cargo specifically. In these challenges, you never lose. [inaudible] lose, yes you may not win prize money. Out of 5000 people, sometimes it's difficult to be, almost to impossible to be in the top three or four that gives prizes but you always gain in terms of knowledge, in terms of experience. You get to collaborate with other people which are talented in the field, you get to add it to your CV that you try to solve this particular problem, and I can tell you there has been some criticists here, people doubt that doing these competitions stops your employ-ability but I can tell you that i know many examples and not want us, they really thought the Ocean Cargo like Master and Grand-master that just by having kind of experience, they have been able to find very decent jobs and even if they had completely diverse backgrounds to the science. So, I can tell you it matters. So, any time you spend here, it's definitely a win for you. I don't see how you can lose by competing in these challenges. You mean if this is something you like right. The whole predictive modeling that the science think. Coffee tempts to shop, because you tend to spend

longer hours. I tend to do this especially late at night. So it definitely tells me something to consider or to be honest any other beverage will do: depends what you like. I see it a bit like a game and I advise you to do the same because if you see it like a game, you never need to work for it. If you know what I mean. So it looks a bit like NRPT. In some way, you have some tools or weapons. These are all the algorithms and feature engineering techniques you can use. And then you have this core leader board and you try to beat all the bad guys and to beat the score and rise above them. So in a way does look like a game. You know you try to use all the tools, all the skills that you have to try to beat the score. So, I think if you see it like a game it really helps you. You don't get tired and you enjoy the process more. I do advise you to take a break though, from my experience you may spend long hours hitting on it and that's not good for your body. You definitely need to take some breaks and do some physical exercise. Go out for a walk. I think it can help most of the times by resting your mind this way can actually help to do better. You have more rested heart, more clear thinking. So, I definitely advise you to do this, generally don't overdo it. I have overnighted in the past but i advise you not to do the same. And now there is a thing that I would like to highlight is that the Cargo community is great. Is one of the most open and helpful helpful communities have experience in any social context, maybe apart from Charities but if you have a question and you posted on the forums or other associated channels like in Slug and people are always willing to help you.That's great, because there are so many people out there and most probably they know the answer or they can help you for a particular problem. And this is invaluable. So many times i have really made use of this, of this option and it really helps. You know this kind of mentality was there even before the serine was gamified. When I say gamified, now you get points by sharping in a way by sharing code or participating in discussions. But in the past, people were doing without really getting something out of it. It maybe the open source mentality of data science that the fact that many people participating are researchers. I don't know but it really is a field that sharing seems to be really important in helping others. So, I do advise you to consider this and don't be afraid to ask in these forums. Another thing that I do at shops, is that after the competition has ended irrespective of how well or not you've done, is go and look for other people and what they have done. Normally, there are threads where people share their approaches, sometimes they share the whole approach would go to sometimes it just give tips and you know this is where you can upgrade your tools and you can see what other people have done and make improvements. And in tandem with this, you should have a notebook of useful methods that you keep updating it at the end of every competition. So, you found an approach that was good, you just add it to that notebook and next time you encounter the same or similar competition you get that notebook out and you apply the same techniques at work in the past and this is how you get better. Actually, if i now start a competition without that notebook, i think it will take me three or four times more in order to get to the same score because a lot of the things that I do now depend on stuff that i have done in the past. So, it's definitely helpf

ul, consider creating this notebook or library of all the approa ches or approaches that have worked in the past in order to have an easier time going on. And that was what I wanted to share wi th you and thank you very much for bearing with me and to see yo u next time, right.Hi everyone. This video is dedicated to the f ollowing advanced feature engineering techniques. Calculating va rious statistics of one feature grouped by another and features derived from neighborhood analysis of a given point. To make it a little bit clearer, let's consider a simple example. Here we h ave a chunk of data for some CTR task. Let's forget about target variable and focus on human features. Namely, User_ID, unique i dentifier of a user, Page_ID, an identifier of a page user visit ed, Ad_price, item prices in the ad, and Ad_position, relative p osition of an ad on the web page. The most straightforward way t o solve this problem is to label and call the Ad_position and fe ed some classifier. It would be a very good classifier that coul d take into account all the hidden relations between variables. But no matter how good it is, it still treats all the data point s independently. And this is where we can apply feature engineer ing. We can imply that an ad with the lowest price on the page w ill catch most of the attention. The rest of the ads on the page won't be very attractive. It's pretty easy to calculate the fea tures relevant to such an implication. We can add lowest and hig hest prices for every user and page per ad. Position of an ad wi th the lowest price could also be of use in such case. Here's on e of the ways to implement statistical features with paid ads. I f our data is stored in the data frame df, we call groupby metho d like this to get maximum and minimum price values. Then store this object in gb variable, and then join it back to the data fr ame df. This is it. I want to emphasize that you should not stop at this point. It's possible to add other useful features not n ecessarily calculated within user and page per. It could be how many pages user has visited, how many pages user has visited dur ing the given session, and ID of the most visited page, how many users have visited that page, and many, many more features. The main idea is to introduce new information. By that means, we ca n drastically increase the quality of the models. But what if th ere is no features to use groupby on? Well, in such case, we can replace grouping operations with finding the nearest neighbors. On the one hand, it's much harder to implement and collect usef ul information. On the other hand, the method is more flexible. We can fine tune things like the size of relevant neighborhood o r metric. The most common and natural example of neighborhood an alysis arises from purposive pricing. Imagine that you need to p redict rental prices. You would probably have some characteristi cs like floor space, number of rooms, presence of a bus stop. Bu t you need something more than that to create a really good mode l. It could be the number of other houses in different neighborh oods like in 500 meters, 1,000 meters, or 1,500 meters, or avera ge price per square meter in such neighborhoods, or the number o f schools, supermarkets, and parking lots in such neighborhoods. The distances to the closest objects of interest like subway st ations or gyms could also be of use. I think you've got the idea . In the example, we've used a very simple case, where neighborh oods were calculated in geographical space. But don't be afraid to apply this method to some abstract or even anonymized feature

space. It still could be very useful. My team and I used this method in Spring Leaf competition. Furthermore, we did it in supervised fashion. Here is how we have done it. First of all, we applied mean encoding to all variables. By doing so, we created homogeneous feature space so we did not worry about scaling and importance of each particular feature. After that, we calculated 2,000 nearest neighbors with Bray-Curtis metric. Then we evaluated various features from those neighbors like mean target of nearest 5, 10, 15, 500, 2,000 neighbors, mean distance to 10 closest neighbors, mean distance to 10 closest neighbors with target 1, and mean distance to 10 closest neighbors with target 0, and, it worked great. In conclusion, I hope you embrace the main ideas of both groupby and nearest neighbor methods and you would be able to apply them in practice. Thank you for your attention.[MUSIC] Hi everyone, in this video I will
talk about the application of matrix factorization technique
in feature extraction. You will see a few application
of the approach for feature extraction and
we will be able to apply it. I will show you several examples
along with practical details. Here's a classic example
of recommendations. Suppose we have some information about
user, like age, region, interest and items like gender, year length. Also we know ratings that
users gave to some items. These ratings can be organized in a user
item matrix with row corresponding to users, and columns corresponding to items,
as shown in the picture. In a cell with coordinates i, j,
the user or agent can be chooser i, give the item j. Assume that our user
have some features Ui. And jth item have is
corresponding feature Mj. And scalar product of these
features produce a rating Rij. Now we can apply matrix factorization
to learning those features for item and users. Sometimes these features
can have an interpretation. Like the first feature in item can
be measured of or something similar. But generally you should consider them
as some extra features, which we can use to encode user in the same way as we did
before with labeling coder or coder. Specifically our assumption about
scale of product is the following. If we present all attributes
of user and items as matrixes, the matrix product will
be very close to the matrix's ratings. In other words,
which way to find matrix's U and M, such as their product
gives the matrix R. This way, this approach is called matrix
factorization or matrix composition. In previous examples, we used both row and
column related features. But sometimes we don't let
the features correspond to rows. Let's consider another example. Suppose that we are texts, do you
remember how we usually classify text? We extract features and each document
was described by a large sparse reactor. If we do matrix factori

zation over
these parse features, we will get the representation for index d
isplayed
in yellow, and terms displayed in green. Although we can somehow
use representation for jumps, we are interested only
in representations for dogs. Now every document is described
by a small, dense reactor. These are our features, and we can us
e
them in a way similar to previous example. This case is often ca
lled
dimension energy reduction. It's quite an efficient way to reduc
e
the size of feature metric, and extract real valued features
from categorical ones. In competitions we often have
different options for purchasing. For example, using text data,
you can run back of big rams and so on. Using matrix optimizatio
n technique, you are able to extract features
from all of these matrices. Since the resulting matrices will be
small, we can easily join them and use togetherness of the featu
res
in tree-based models. Now I want to make a few comments
about matrix factorization. Not just that we are not
constrained to reduce whole matrix, you can apply factorization
to a subset
of a column and leave the other as is. Besides reduction you can
use pressure boards for getting another presentation
of the same data. This is especially useful for example since it
provides velocity
of its models and leads to a better. Of course matrix factorizat
ion
is a loss of transformation, in other words we will lose some
information after the search reduction. Efficiency of this appro
ach heavily
depends on a particular task and choose a number of latent facto
rs. The number should be considered as
a hyper parameter and needs to be tuned. It's a good practice to
choose
a number of factors between 5 and 100. Now, let's switch from ge
neral idea
to particular implementations. Several matrix factorization
methods are implemented in circuit as the most famous SVD and PC
A. In addition,
their use included TruncatedSVD, which can work with sparse matr
ices. It's very convenient for example,
in case of text datasets. Also there exists a so called non-nega
tive
matrix factorization, or NMF. It impose an additional restrictio
ns that
all hidden factors are non-negative, that is either zero or a po
sitive number. It can be applied only to
non-negative matrixes. For example matrix where all represented
occurence of each word in the document. NMF has an interesting p
roperty, it transforms data in a way that makes
data more suitable for decision trees. Take a look at the pictur
e from
Microsoft Mobile Classification Challenge. It can be seen that N

MF transform data
forms lines parallel to the axis. A few more notes on matrix fac
torizations. Essentially they are very
similar to linear models, so we can use the same transformation
tricks as we use for linear models. So in addition to standard N
MF, I advise you to apply
the factorization to transform data. Here's another plot from th
e competition. It's clear that these two transformations
produce different features, and we don't have to choose the best
one. Instead, it's beneficial
to use both of them. I want to note that matrix factorization
is a trainable transformation, and has its own parameters. So we
should be careful, and
use the same transformation for all parts of your data set. Read
ing and transforming each
part individually is wrong, because in that case you will get
two different transformations. This can lead to an error
which will be hard to find. The correct method is shown below,
first we need to the data information on all data and
only then apply to each individual piece. To sum up, matrix comp
osition is a very
general approach to dimensional reduction and feature extraction
. It can be used to transform
categorical feature into real ones. And tricks for linear models
are also
suitable for matrix factorizations. Thank you for your attention
. [MUSIC] [SOUND]Hi, everyone. The main topic of this video is F
eature Interactions. You will learn how to construct them and us
e in problem solving. Additionally, we will discuss them for fea
ture extraction from decision trees. Let's start with an example
. Suppose that we are building a model to predict the best adver
tisement banner to display on a website. Among available feature
s, there are two categorical ones that we will concentrate on. T
he category of the advertising banner itself and the category of
the site the banner will be showing on. Certainly, we can use t
he features as two independent ones, but a really important feat
ure is indeed the combination of them. We can explicitly constru
ct the combination in order to incorporate our knowledge into a
model. Let's construct new feature named ad_site that represents
the combination. It will be categorical as the old ones, but se
t of its values will be all possible combinations of two origina
l values. From a technical point of view, there are two ways to
construct such interaction. Let's look at a simple example. Cons
ider our first feature, f1, has values A or B. Another feature,
f2, has values X or Y or Z, and our data set consist of four dat
a points. The first approach is to concatenate the text values o
f f1 and f2, and use the result as a new categorical feature f_j
oin. We can then apply the OneHot according to it. The second ap
proach consist of two steps. Firstly, apply OneHot and connect t
o features f1 and f2. Secondly, construct new metrics by multipl
ying each column from f1 encoded metrics to each column from f2
encoded metrics. It was nothing that both methods results in pra
ctically the same new feature representations. In the above exam
ple, we can consider as interactions between categorical feature
s, but similar ideas can be applied to real valued features. For
example, having two real valued features f1 and f2, interaction

s between them can be obtained by multiplications of f1 and f2. In fact, we are not limited to use only multiply operation. Any function taking two arguments like sum, difference, or division is okay. The following transformations significantly enlarge fea ture space and makes learning easier, but keep in mind that it m akes or frequent easier too. It should be emphasized that for th ree ways algorithms such as the random forest or gradient boost decision trees it's difficult to extract such kind of dependenci es. That's why they're buffer transformation are very efficient for three based methods. Let's discuss practical details now. Wh ere wise future generation approaches greatly increase the numbe r of the features. If there were any original features, there wi ll be n square. And will be even more features if several types of interaction are used. There are two ways to moderate this, ei ther do feature selection or dimensionality reduction. I prefer doing the selection since not all but only a few interactions of ten achieve the same quality as all combinations of features. Fo r each type of interaction, I construct all piecewise feature in teractions. Feature random forests over them and select several most important features. Because number of resulting features fo r each type is relatively small. It's possible to join them toge ther along with original features and use as input for any machi ne learning algorithm usually to be by use method. During the vi deo, we have examined the method to construct second order inter actions. But you can similarly produce throned order or higher. Due to the fact that number of features grow rapidly with order, it has become difficult to work with them. Therefore high order directions are often constructed semi-manually. And this is an art in some ways. Additionally, I would like to talk about metho ds to construct categorical features from decision trees. Take a look at the decision tree. Let's map each leaf into a binary fe ature. The index of the object's leaf can be used as a value for a new categorical feature. If we use not a single tree but an e nsemble of them. For example, a random forest, then such operati on can be applied to each of entries. This is a powerful way to extract high order interactions. This technique is quite simple to implement. Tree-based poodles from sklearn library have an ap ply method which takes as input feature metrics and rituals corr esponding indices of leaves. In xgboost, also support to why a p arameter breed leaf in predict method. I suggest we need to coll aborate documentations in order to get more information about th ese methods and IPIs. In the end of this video, I will tackle th e main points. We examined method to construct an interactions o f categorical features. Also, we extend the approach to real-val ued features. And we have learned how to use trees to extract hi gh order interactions. Thank you for your attention.Hi, everyone . Today, we will discuss this new method for visualizing data in tegrating features. At the end of this video, you will be able t o use tSNE in your products. In the previous video, we learned a bout metaphysician technique that is predatory very close to lin ear models. In this video, we will touch the subject of non-line ar methods of dimensionality reduction. That says in general are called manifold learning. For example, look at the data in form of letter S on the left side. On the right, we can see results of running different manifold learning algorithm on the data. Th is new result is placed at the right bottom corner on the slide.

This new algorithm is the main topic of the lecture, as it tells of how this really works won't be explained here. But you will come to look at additional materials for the details. Let's just say that this is a method that tries to project points from high dimensional space into small dimensional space so that the distances between points are approximately preserved. Let's look at the example of the tSNE on the MNIST dataset. Here are points from 700 dimensional space that are projected into two dimensional space. You can see that such projection forms explicit clusters. Coolest shows that these clusters are meaningful and corresponds to the target numbers well. Moreover, neighbor clusters corresponds to a visually similar numbers. For example, cluster of three is located next to the cluster of five which in chance is adjustment to the cluster of six and eight. If data has explicit structure as in case of MNIST dataset, it's likely to be reflected on tSNE plot. For the reason tSNE is widely used in exploratory data analysis. However, do not assume that tSNE is a magic want that always helps. For example, a misfortune choice of hyper parameters may lead to poor results. Consider an example, in the center is the least presented a tSNE projection of exactly the same MNIST data as in previous example, only perplexity parameter has been changed. On the left, for comparison, we have plots from previous right. On the right, so it present a tSNE projection of random data. We can see as a choice of hybrid parameters change projection of MNIST data significantly so that we cannot see clusters. Moreover, new projection become more similar to random data rather than to the original. Let's find out what depends on the perplexity hyperparameter value. On the left, we have perplexity=3, in the center=10, and on the right= 150. I want to emphasize that these projections are all made for the same data. The illustration shows that these new results strongly depends on its parameters, and the interpretation of the results is not a simple task. In particular, one cannot infer the size of original clusters using the size of projected clusters. Similar proposition is valid for a distance between clusters. Blog distill.pub contain a post about how to understand and interpret the results of tSNE. Also, it contains a great interactive demo that will help you to get into issues of how tSNE works. I strongly advise you to take a look at it. In addition to exploratory data analysis, tSNE can be considered as a method to obtain new features from data. You should just concatenate the transformers coordinates to the original feature matrix. Now if you've heard this about practical details, as it has been shown earlier, the results of tSNE algorithm, it strongly depends on hyperparameters. It is good practice to use several projections with different perplexities. In addition, because of stochastic of this methods results in different projections even with the same data and hyperparameters. This means the train and test sets should be projected together rather than separately. Also, tSNE will run for a long time if you have a lot of features. If the number of features is greater than 500, you should use one of dimensionality reduction approach and reduce number of features, for example, to 100. Implementation of tSNE can be found in the sklearn library. But personally, I prefer to use another implementation from a separate Python package called tSNE, since it provide a way more efficient implementation. In conclusion, I want to remind you the basic

points of the lecture. TSNE is an excellent tool for visualizing data. If data has an explicit structure, then it likely be [inaudible] on tSNE projection. However, it requires to be cautious with interpretation of tSNE results. Sometimes you can see structure where it does not exist or vice versa, see none where structure is actually present. It's a good practice to do several tSNE projections with different perplexities. And in addition to EJ, tSNE is working very well as a feature for feeding models. Thank you for your attention.Hello everyone, this is Marios Michailidis, and this will be the first video in a series that we will be discussing on ensemble methods for machine learning. To tell you a bit about me, I work as Research Data Scientist for H2Oai. In fact, my PhD is about assemble methods, and they used to be ranked number one in cargo and ensemble methods have greatly helped me to achieve this spot. So you might find the course interesting. So what is ensemble modelling? I think with this term, we refer to combining many different machine learning models in order to get a more powerful prediction. And later on we will see examples that this happens, that we combine different models and we do get better predictions. There are various ensemble methods. Here we'll discuss a few, those that we encounter quite often, in predictive modelling competitions, and they tend to be, in general, quite competitive. We will start with simple averaging methods, then we'll go to weighted averaging methods, and we will also examine conditional averaging. And then we will move to some more typical ones like bagging, or the very, very popular, boosting, then stacking and StackNet, which is the result of my research. But as I said, these will be a series of videos, and we will initially start with the averaging methods. So, in order to help you understand a bit more about the averaging methods, let's take an example. Let's say we have a variable called age, as in age years, and we try to predict this. We have a model that yields prediction for age. Let's assume that the relationship between the two, the actual age in our prediction, looks like in the graph, as in the graph. So you can see that the model boasts quite a higher square of a value of 0.91, but it doesn't do so well in the whole range of values. So when age is less than 50, the model actually does quite well. But when age is more than 50, you can see that the average error is higher. Now let's take another example. Let's assume we have a second model that also tries to predict age, but this one looks like that. As you can see, this model does quite well when age is higher than 50, but not so well when age is less than 50,

nevertheless, it scores again 0.91. So we have two models that h
ave
a similar predictive power, but they look quite different. It's
quite obvious that they do
better in different parts of the distribution of age. So what wi
ll happen if we
were to try to combine this two with a simple averaging method,
in other words, just say (model 1 + model two) / 2,
so a simple averaging method. The end result will look
as in the new graph. So, our square has moved to 0.95,
which is a considerable improvement versus the 0.91 we had befor
e,
and as you can see, on average, the points tend to
be closer with the reality. So the average error is smaller. How
ever, as you can see, the model doesn't
do better as an individual models for the areas where the models
were doing really well, nevertheless, it does better on average.
 This is something we need to understand, that there is potentia
lly a better
way to combine these models. We could try to take a weighting av
erage. So say, I'm going to take 70% of
the first model prediction and 30% of the second model predictio
n. In other words,
(model 1x0.7 + model 2x0.3), and the end result would
look as in the graph. So you can see their square is no better
and that makes sense, because the models have quite similar pred
ictive power and
it doesn't make sense to rely more in one. And also it is quite
clear that
it looks more with model 1, because it has better predictions
when age is less than 50, and worse predictions
when age is more than 50. As a theoretical exercise, what is the
theoretical best we could get out of this? We know we have a mod
el that scores
really well when age is less than 50, and another model that sco
res really
well when age is more than 50. So ideally, we would like to
get to something like that. This is how we leverage the two
models in the best possible way here by using a simple
conditioning method. So if less than 50 is one I'll just
use the other, and we will see later on that there are ensemble
methods
that are very good at finding these relationships of two or more
 predictions
in respect to the target variable. But, this will be a topic for
another discussion. Here we discuss simple averaging methods, ho
pefully you found it useful, and
stay here for the next session to come. Thank you very much.Hell
o everyone. This is Marios Michailidis and we will continue our
discussion in regards to ensemble methods. Previously, we saw so
me simple averaging methods. This time, we'll discuss about bagg
ing, which is a very popular and efficient form of ensembling. W
hat is bagging? bagging refers to averaging slightly different v
ersions of the same model as a means to improve the predictive p
ower. A common and quite successful application of bagging is th
e Random Forest. Where you would run many different versions of

decision trees in order to get a better prediction. Why should w e consider bagging? Generally, in the modeling process, there ar e two main sources of error. There are errors due to bias often referred to as underfitting, and errors due to variance often re ferred to as overfitting. In order to better understand this, I' ll give you two opposite examples. One with high bias and low va riance and vice versa in order to understand the concept better. Let's take an example of high bias and low variance. We have a person who is let's say young, less than 30 years old and we kno w this person is quite rich and we're trying to find him, this p erson who'll buy a racing or an expensive car. Our model has hig h variance, has high bias if it says that this person is young a nd I think he's not going to buy an expensive car. What the mode l has done here is that it hasn't explore very deep relationship within the data. It doesn't matter that this person is young if it has a lots of money when it comes to buying a car. It hasn't explored different relationships. In other words, it has been u nderfitted. However, this is also associated with low variance b ecause this relationship, the fact that a young person generally doesn't buy an expensive car is generally true so we would expe ct this information to generalize well enough in a foreseen data . Therefore, the variance is low in this example. Now, let's try to see the other way around, an example with high variance and low bias. Let's assume we have a person. His name is John. He li ves in a green house, has brown eyes, and we want to see he will buy a car. A model that has gone so deep in order to find these relationships actually has a low bias because it has really exp lored a lots of information about the training data. However, it is making the mistake that every person that has these characte ristics is going to buy a car. Therefore, it generalizes for som ething that it shouldn't. In other words, it has already exhaust ed the information in the training data and the results are not significant. So, here, we actually have high variance but low bi as. If we were to visualize the relationship between prediction error and model complexity, it would look like that. When we beg in the training of the model, we can see that the training error make the error in that training data gets reduced and the same happens in the test data because the predictions are easily gene ralizable. They are simple. However, after a point, any improvem ents in the training error are not realized into test data. This is the point where the model starts over exhausting information , creates predictions that are not generalizable. This is where bagging actually comes into play and offers it's utmost value. B y making slightly different or let say randomized models, we ens ure that the predictions do not read very high variance. They're generally more generalizable. We don't over exhaust the informa tion in the training data. At the same time, we saw before that when you average slightly different models, we are generally abl e to get better predictions and we can assume that in 10 models, we are still able to find quite significant information about t he training data. Therefore, this is why bagging tends to work q uite well and personally, I always use bagging. When I say, "I f it a model," I have actually not fit a model I have fit a baggin g version of this model so probably that different models. Which parameters are associated with bagging? The first is the seed. We can understand that many algorithms have some randomized proc

edures so by changing the seed you ensure that they are made sli ghtly differently. At the same time, you can run a model with le ss rows or you could use bootstrapping. Bootstrapping is differe nt from row sub-sampling in the sense that you create an artific ial dataset so you might let's say data row the training data th ree or four times. You create a random dataset from the training data. A different form of randomness can be imputed with shuffl ing. There are some algorithms, which are sensitive to the order of the data. By changing the order you ensure that the models b ecome quite different. Another way is to dating a random sample of columns so bid models on different features or different vari ables of the data. Then you have model-specific parameters. For example, in a linear model, you will try to build 10 different l et's say logistic regression with slightly different regularizat ion parameters. Obviously, you could also control the number of models you include in your ensemble or in this case we call them bags. Normally, we put a value more than 10 here but, in princi ple, the more bags you put, it doesn't hurt you. It makes result s better but after some point, performance start plateauing. So there is a cost benefit with time but, in principle, more bags i s generally better and optionally, you can also apply parallelis m. Bagging models are independent to each other, which means you can build many of them at the same time and make full use of yo ur computation power. Now, we can see an example about bagging b ut before I do that, just to let you know that a bagging estimat ors that scikit-learn has in Python are actually quite cool. The refore, I recommend them. This is a typical 15 lines of code tha t I use quite often. They seem really simple but they're actuall y quite efficient. Assuming you have a training at the test data set and to target variable, what you do is you specify some bagg ing parameters. What is the model I'm going to use at random for est? How many bags I'm going to run? 10. What will be my seed? O ne. Then you create an object, an empty object that will save th e predictions and then you run a loop for as many bags as you ha ve specified. In this loop, you repeat the same. You change the seed, you feed the model, you make predictions in the test data and you save these predictions and then, you just take an averag e of these predictions. This is the end of the session. In this session, we discussed bagging as a popular form of ensembling. W e saw bagging in association with variants and bias and we also saw in the example about how to use it. Thank you very much. The next session we will describe boosting, which is also very popu lar so stay in tune and have a good day.Hello, everyone. This is Marios Michailidis. And today, we'll continue our discussion wi th ensemble methods, and specifically, with a very popular form of ensembling called boosting. What is boosting? Boosting is a f orm of weighted averaging of models where each model is built se quentially in a way that it takes into account previous model pe rformance. In order to understand this better, remember that bef ore, we discussed about biking, and we saw that we can have it a t many different models, which are independent to each other in order to get a better prediction. Boosting does something differ ent. It says, now I tried to make a model, but I take into accou nt how well the previous models have done in order to make a bet ter prediction. So, every model we add sequentially to the ensem ble, it takes into account how well the previous models have don

e in order to make better predictions. There are two main boosti ng type of algorithms. One is based on weight, and the other is based on residual error, and we will discuss both of them one by one. For weight boosting, it's better to see an example in orde r to understand it better. Let's say we have a tabular data set, with four features. Let's call them x0, x1, x2, and x3, and we want to use these features to predict a target variable, y. What we are going to do in weight boosting is, we are going to fit a model, and we will generate predictions. Let's call them pred. These predictions have a certain margin of error. We can calcula te these absolute error, and when I say absolute error, is absol ute of y minus our prediction. You can see there are predictions which are very, very far off, like row number five, but there a re others like number six, which the model has actually done qui te well. So what we do based on this is we generate, let's say, a new column or a new vector, where we create a weight column, a nd we say that this weight is 1 plus the absolute error. There a re different ways to calculate this weight. Now, I'm just giving you this as an example. You can infer that there are different ways to do this, but the overall principle is very similar. So w hat you're going to do next is, you're going to fit a new model using the same features and the same target variable, but you're going to also add this weight. What weight says to the model is , I want you to put more significance into a certain role. You c an almost interpret weight has the number of times that a certai n row appears in my data. So let's say weight was 2, this means that this row appears twice, and therefore, has bigger contribut ion to the total error. You can keep repeating this process. You can, again, calculate a new error based on this error. You calc ulate new weights, and this is how you sequentially add models t o the ensemble that take into account how well each model has do ne in certain cases, maximizing the focus from where the previou s models have done more wrong. There are certain parameters asso ciated with this type of boosting. One is the learning rate. We can also call it shrinkage or eta. It has different names. Now, if you recall, I explained boosting as a form of weighted averag ing. And this is true, because normally what this learning rate. So what we say is, every new model we built, we shouldn't trust it 100%. We should trust it a little bit. This ensures that we don't have one model generally having too much contribution, and completely making something that is not very generalizable. So this ensures that we don't over-trust one model, we trust many m odels a little bit. It is very good to control over fitting. The second parameter we look at is the number of estimators. This i s quite important. And normally, there is an inverse relationshi p, an opposite relationship, with the learning rate. So the more estimators we add to these type of ensemble, the smaller learni ng rate we need to put. This is sometimes quite difficult to fin d the right values, and we do it with the help of cross-validati on. So normally, we start with a fixed number of estimators, let 's say, 100, and then, we try to find the optimal learning rate for this 100 estimators. Let's say, based on cross-validation pe rformance, we find this to be 0.1. What we can do then is, let's say, we can double the number of estimators, make it 200, and d ivide the learning rate by 2, so we can put 0.05, and then we ta ke performance. It may be that the relationship is not as linear

as I explained, and the best learning rate may be 0.04 or 0.06 after duplicating the estimators, but this is roughly the logic. This is how we work in order to increase estimators, and try to see more estimators give us better performance without losing so much time, every time, trying to find the best learning rate. Another thing we look at is the type of input model. And generally, we can perform boosting with any type of estimator. The only condition is that it needs to accept weight in it's modeling process. So I weigh to say how much we should rely in each role of our data set. And then, we have various boosting types. As I said, I roughly explained to you how we can use the weight as a means to focus on different rows, different cases the model has done wrong, but there are different ways to express this. For example, there are certain boosting algorithm that do not care about the margin of error, they only care if you did the classification correct or not. So there are different variations. One I really like is the AdaBoost, and there is a very good implementation in sklearn, where you can choose any input algorithm. I think it's really good. And another one I really like is, normally, it's only good for logistic regression, and there is a very good implementation in Weka for Java if you want to try. Now, let's move onto the our time of boosting, which has been the most successful. I believe that in any predictive modeling competition that was not image classification or predicting videos. This has been the most dominant type of algorithm that actually has one most in these challenges so this type of boosting has been extremely successful, but what is it? I'll try to give you again a similar example in order to understand the concept. Let's say we have again the same dataset, same features, again when trying to predict a y variable, we fit a model, we make predictions. What we do next, is we'll calculate the error of these predictions but this time, not in absolute terms because we're interested about the direction of the error. What we do next is we take this error and we make it adding new y variable so the error now becomes the new target variable and we use the same features in order to predict this error. It's an interesting concept and if we wanted, let's say to make predictions for Rownum equals one, what we would do is we will take our initial prediction and then we'll add the new prediction, which is based on the error of the first prediction. So initially, we have 0.75 and then we predicted 0.2. In order to make a final prediction, we would say one plus the other equals 0.95. If you recall, the target for this row, it was one. Using two models, we were able to get closer to the actual answer. This form of boosting works really, really well to minimize the error. There are certain parameters again which are associated with this type of boosting. The first is again the learning rate and it works pretty much as I explained it before. What you need to take into account is how this is applied. Let's say we have a learning rate of 0.1. In the previous example, where the prediction was 0.2 for the second model, what you will say is I want to move my prediction towards that direction only 10 percent. If you remember the prediction was 0.2, 10 percent of this is 0.02. This is how much we would move towards the prediction of the error. This is a good way to control over fitting. Again, we ensure we don't over rely in one model. Again, how many estimators you put is quite important. Normally, more is better but

you need to offset this with the right learning rate. You need t o make certain that every model has the right contribution. If y ou intent to put many, then you need to make sure that your mode ls have very, very small contribution. Again, you decide these p arameters based on cross-validation and the logic is very simila r as explained before. Other things that work really well is tak ing a subset of rows or a subset of columns when you build its m odel. Actually, there is no reason why we wouldn't use this with the previous algorithm. The way its based, it is more common wi th this type of boosting, and internally works quite well. For i nput model, I have seen that this method works really well with this increase but theoretically, you can put anything you want. Again, there are various boosting types. I think the two most co mmon or more successful right now in a predictive modeling conte xt is the gradient based, which is actually what I explained wit h you how the prediction and you don't move 100 percent with tha t direction if you apply the learning rate. The other very inter esting one, which I've actually find it very efficient especiall y in classification problems is the dart. Dart, it imposes a dro p out mechanism in order to control the contribution of the tree s. This is a concept derived from deep learning where you say, " Every time I make a new prediction in my sample, every time I ad d a new estimate or I'm not relying on all previous estimators b ut only on a subset of them." Just to give you an example, let's say we have a drop out rate of 20 percent. So far, we have buil t 10 trees, we want to or 10 models and then we try to see, we t ry to build a new, an 11th one. What we'll do is we will randoml y exclude two trees when we generate a prediction for that 11th tree or that 11th model. By randomly excluding some models, by i ntroducing this kind of randomness, it works as a form of regula rization. Therefore, it helps a lot to make a model that general izes quite well enough for same data. This concept tends to work quite well because this type of boosting algorithm has been so successful. There have been many implementations to try to impro ve on different parts of these algorithms. One really successful application especially in the comparative predictive modeling w orld is the Xgboost. It is very scalable and it supports many lo ss functions. At the same time, is available in all major progra mming languages for data science. Another good implementation is Lightgbm. As the name connotes, it is lightning fast. Also, it is supported by many programming languages and supports many los s functions. Another interesting case is the Gradient Boosting M achine from H2O. What's really interesting about this implementa tion is that it can handle categorical variables out of the box and it also comes with a real set of parameters where you can co ntrol the modeling process quite thoroughly. Another interesting case, which is also fairly new is the Catboost. What's really g ood about this is that it comes with the strong initial set of p arameters. Therefore, you don't need to spend so much time tunin g. As I mentioned before, this can be quite a time consuming pro cess. It can also handle categorical variables out of the box. U ltimately, I really like the Gradient Boosting Machine implement ation of Scikit-learn. What I really like about this is that you can put any scikit-learn estimator as a base. This is the end o f this video. In the next session, we will discuss docking, whic h is also very popular, so stay tuned.Continuing our discussion

with ensemble
methods, next one up is stacking. Stacking is a very, very popular form of ensembling using
predictive modeling competitions. And I believe in most competitions,
there is a form of stacking in the end in order to boost
your performance as best as you can. Going through the definition of stacking,
it essentially means making several predictions
with hold-out data sets. And then collecting or
stacking these predictions to form a new data set,
where you can fit a new model on it, on this newly-formed data set from predictions. I would like to take you through
a very simple, I would say naive, example to show you how,
conceptually, this can work. I mean, we have so far seen that you
can use previous models' predictions to affect a new model, but
always in relation with the input data. This is a new concept because we're
only going to use the predictions of some models in order
to make a better model. So let's see how these could
work in a real life scenario. Let's assume we have three kids,
let's name them LR, SVM, KNN, and
they argue about a physics question. So each one believes the answer to
a physics question is different. First one says 13, second 18, third 11, they don't know how to
solve this disagreement. They do the honorable thing,
they say let's take an average, which in this case is 14. So you can almost see the kids, there's different models here,
they take input data. In this case,
it's the question about physics. They process it based on
historical information and and they are able to output an estimate,
a prediction. Have they done it optimally, though? Another way to say this is
to say there was a teacher, Miss DL, who had seen this discussion,
and she decided to step up. While she didn't hear the question,
she does know the students quite well, she knows the strength and
weaknesses of each one. She knows how well they have done
historically in physics questions. And from the range of values they have
provided, she is able to give an estimate. Let's say that in this concept, she knows
that SVM is really good in physics, and her father works in the department
of Physics of Excellence. And therefore she should have
a bigger contribution to this ensemble than every other kid,
therefore the answer is 17. And this is how a meta model works,
it doesn't need to know the input data. It just knows how the models
have done historically, in order to find the best
way to combine them. And this can work quite well in practice. So, let's go more into

the methodology of stacking. Wolpert introduced stacking in 1992
, as a meta modeling technique
to combine different models. It consists of several steps. The f
irst step is,
let's assume we have a train data set, let's divide it into two
parts; so
a training and the validation. Then you take the training part,
and you train several models. And then you make predictions for
the second part, let's say the validation data set. Then you col
lect all these predictions,
or you stack these predictions. You form a new data set and
you use this as inputs to a new model. Normally we call this a m
eta model,
and the models we run into, we call them base model or base lear
ners. If you're still confused about stacking,
consider the following animation. So let's assume we have three
data sets A,
B, and C. In this case, A will serve
the role of the training data set, B will be the validation data
 set, and C will be the test data sets where we
want to make the final predictions. They all have similar archit
ectural,
four features, and one target variable we try to predict. So in
this case,
we can choose an algorithm to train a model based on data set 1,
 and then we make predictions for
B and C at the same time. Now we take these predictions, and
we put them into a new data set. So we create a data set to stor
e the
predictions for the validation data in B1. And a data set called
 C1 to save
predictions for the test data, called C1. Then we're going to re
peat the process, now we're going to choose
another algorithm. Again, we will fit it on A data set. We will
make predictions on B and
C at the same time, and we will save these predictions
into the newly-formed data sets. And we essentially append them,
we stack them next to each other, this is where stacking takes i
ts name. And we can continue this even more,
do it with a third algorithm. Again the same, fit on A,
predict on B and C, same predictions. What we do then is we take
 the target
variable for the B data set, or the validation datadset,
which we already knew. And we are going to fit a new model on B1
with the target of the validation data, and then we will make pr
edictions from C1. And this is how we combine
different models with stacking, to hopefully make better predict
ions for
the test or the unobserved data. Let us go through an example,
a simple example in Python, in order to understand better,
as in in code, how it would work. It is quite simple, so even pe
ople not very experienced with
Python hopefully can understand this. The main logic is that we
will use
two base learners on some input data, a random forest and a line
ar regression. And then, we will try to combine

the results, starting with a meta learner, again, it will be linear regression. Let's assume we again have
a train data set, and a target variable for
this data set, and a test data set. Maybe the code seems a bit intimidating,
but we will go step by step. What we do initially is we take the train
data set and we split it in two parts. So we create a training and
a valid data set out of this, and we also split the target variable. So we create ytraining and
yvalid, and we split this by 50%. We could have chosen something else,
let's say 50%. Then we specify our base learners,
so model1 is the random forest in this case, and
model2 is a linear regression. What we do then is we fit
the both models using the training data and the training target. And we make predictions for
the validation data for both models, and at the same time we'll make
predictions for the test data. Again, for both models,
we save these as preds1, preds2, and for the test data,
test_preds1 and test_preds2. Then we are going to
collect the predictions, we are going to stack the predictions and
create two new data sets. One for validation,
where we call it stacked_predictions, which consists of preds1 and preds2. And then for the data set for
for the test predictions, called stacked_test_predictions, where we stack test_preds1 and test_preds2. Then we specify a meta learner, let's call it meta_model,
which is a linear regression. And we fit this model on the predictions
made on the validation data and the target for the validation data, which
was our holdout data set all this time. And then we can generate predictions for the test data by applying this model
on the stacked_test_predictions. This is how it works. Now, I think this is a good
time to revisit an old example we used in the first session,
about simple averaging. If you remember,
we had a prediction that was doing quite well to predict age when
the age was less than 50, and another prediction that was doing
quite well when age was more than 50. And we did something tricky,
we said if it is less than 50, we'll use the first one, if age is more
than 50, we will use the other one. The reason this is tricky is
because normally we use the target information to make this decision. Where in an ideal world, this is what
you try to predict, you don't know it. We have done it in order to show what
is the theoretical best we could get, or yeah, the best. So taking the same predictions and applying stacking, this is what the
end

result would actually look like. As you can see,
it has done pretty similarly. The only area that there is some
error is around the threshold of 50. And that makes sense, because the model
doesn't see the target variable, is not able to identify
this cut of 50 exactly. So it tries to do it only
based on the input models, and there is some overlap around this area. But you can see that stacking
is able to identify this, and use it in order to
make better predictions. There are certain things you need to
be mindful of when using stacking. One is when you have time-sensitive data,
as in let's say, time series, you need to formulate your
stacking so that you respect time. What I mean is, when you create
your train and validation data, you need to make certain that your train
is in the past and your validation is in the future, and ideally your
test data is also in the future. So you need to respect this
time element in order to make certain your model generalizes well. The other thing you need to look at is,
obviously, single model performance is important. But the other thing that is
also very important is model diversity, how different
a model is to each other. What is the new information each
model brings into the table? Now, because stacking, and depending
on the algorithms you will use for stacking, can go quite deep
into exploring relationships. It will find when a model is good, and when a model is actually bad or
fairly weak. So you don't need to worry too much
to make all the models really strong, stacking can actually extract
the juice from each prediction. Therefore, what you really need to focus
is, am I making a model that brings some information,
even though it is generally weak? And this is true, there have been many
situations where I've made, I've had some quite weak models in my ensemble,
I mean, compared to the top performance. And nevertheless, they were actually
adding lots of value in stacking. They were bringing in new information
that the meta model could leverage. Normally, you introduce
diversity from two forms, one is by choosing a different algorithm. Which makes sense, certain algorithms capitalize on
different relationships within the data. For example, a linear model will
focus on a linear relationship, a non-linear model can capture
better a non-linear relationships. So predictions may come a bit different. The other thing is you can
even run the same model, but you try to run it on different
transformation of input data, either less features or
completely different transformation. For example, in one data se

t you may treat categorical
features as one whole encoding. In another,
you may just use label in coding, and the result will probably p
roduce
a model that is very different. Generally, there is no limit to
how many models you can stack. But you can expect that
there is a plateauing after certain models have been added. So i
nitially, you will see some
significant uplift in whatever metric you are testing on every
time you run the model. But after some point, the incremental
uplift will be fairly small. Generally, there's no
way to know this before, exactly what is the number of models
where we will start plateauing. But generally, this is a affecte
d by how
many features you have in your data, how much diversity you mana
ged
to introduce into your models, quite often how many
rows of data you have. So it is tough to know this beforehand,
but generally this is
something to be mindful of. But there is a point where adding mo
re
models actually does not add that much value. And because the me
ta model, the meta model will only use predictions of other mode
ls. We can assume that the other
models have done, let's say, a deep work or
a deep job to scrutinize the data. And therefore the meta model
doesn't need to be so deep. Normally, you have predictions with
are correlated with the target. And the only thing it needs to d
o is
just to find a way to combine them, and that is normally not so
complicated. Therefore, it is quite often that
the meta model is generally simpler. So if I was to express this
in a random forest context, it will have lower depth than what w
as the
best one you found in your base models. This was the end of the
session,
here we discussed stacking. In the next one, we will discuss a v
ery
interesting concept about stacking and extending it on multiple
levels,
called stack net. So stay in tune.We can continue our discussion
 with StackNet. StackNet is a scalable meta modeling methodology
 that utilizes stacking to combine multiple models in a neural n
etwork architecture of multiple levels. It is scalable because w
ithin the same level, we can run all the models in parallel. It
utilizes stacking because it makes use of this technique we ment
ioned before where we split the data, we make predictions so som
e hold out data, and then we use another model to train on those
 predictions. And as we will see later on, this resembles a lot
in neural network. Now let us continue that naive example we gav
e before with the students and the teacher, in order to understa
nd what conceptually, in a real world, would need to add another
 layer. So in that example, we have a teacher that she was tryin
g to combine the answers of different students and she was outpu
tting an estimate of 17 under certain assumptions. We can make t
his example more interesting by introducing one more meta learne

r. Let's call him Mr. RF, who's also a physics teacher. Mr. RF b elieves that LR should have a bigger contribution to the ensembl e because he has been doing private lessons with him and he know s he couldn't be that far off. So he's able to see the data from slightly different ways to capitalize on different parts of the se predictions and make a different estimate. Whereas, the teach ers could work it out and take an average, we could create or we can introduce a higher authority or another layer of modeling h ere. Let's call it the headmaster, GBM, in order to shop, make b etter predictions. And GBM doesn't need to know the answers that the students have given. The only thing he needs to know is the input from the teachers. And in this case, he's more keen to tr ust his physics teacher by outputting a 16.2 prediction. Why wou ld this be of any use to people? I mean, isn't that already comp licated? Why would we want to ever try something so complicated? I'm giving you an example of a competition my team used, four l ayer of stacking, in order to win. And we used two different sou rces of input data. We generated multiple models. Normally, exit boost and logistic regressions, and then we fed those into a fo ur-layer architecture in order to get the top score. And althoug h we could have escaped without using that fourth layer, we stil l need it up to level three in order to win. So you can understa nd the usefulness of deploying deep stacking. Another example is the Homesite competition organized by Homesite insurance where again, we created many different views of the data. So we had di fferent transformations. We generated many models. We fed those models into a three-level architecture. I think we didn't need t he third layer again. Probably, we could have escaped with only two levels but again, deep stacking was necessary in order to wi n. So there is your answer, deep stacking on multiple levels rea lly helps you to win competitions. In the spirit of fairness and openness, there has been some criticism about large ensembles t hat maybe they don't have commercial value, they are confidentia lly expensive. I have to add three things on that. The first is, what is considered expensive today may not be expensive tomorro w and we have seen that, for example, with the deep learning, wh ere with the advent of GPUs, they have become 100 times faster a nd now they have become again very, very popular. The other thin g is, you don't need to always build very, very deep ensembles b ut still, small ensembles would still really help. So knowing ho w to do them can add value to businesses, again based on differe nt assumptions about how fast they want the decisions, how much is the uplift you can see from stacking, which may vary, sometim es it's more, sometime is less. And generally, how much computin g power they have. We can make a case that even stacking on mult iple layers can be very useful. And the last point is that these are predictive modeling competitions so it is a bit like the Ol ympics. It is nice to be able to see the theoretical best you ca n get because this is how innovation takes over. This is how we move forward. We can express StackNet as a neural network. So no rmally, in a neural network, we have these architecture of hidde n units where they are connected with input with the form of lin ear regression. So actually, it looks pretty much like a linear regression. So whether you have a set of coefficients and you ha ve a constant value where you call it bias in neaural networks, and this is how your output predictions which one of the hidden

units which are then taken, collected, to create the output. The concept of StackNet is actually not that much different. The only thing we want to do is, we don't want to be limited to that linear regression or to that perception. We want to be able to use any machine learning algorithm. Putting that aside, the architecture should be exactly the same, could be fairly similar. So how to train this? In a typical neural network, we use bipropagation. Here in this context, this is not feasible. I mean in the context of trying to make this network work with any input model because not all are differentiable. So this is why we can use stacking. Stacking here is a way to link the output, the prediction, the output of the node, with target variable. This is how the link also is made from the input features with a node. However, if you remember the way that stacking works is you have some train data. And then, you need to divide it into two halves. So, you use the first part called, training, in order to make predictions to the other part called, valid. If we, assuming that adding more layers gives us some uplift, if we wanted to do this again, we would have re-split the valid data into two parts. Let's call it, mini train, and mini valid. And you can see the problem here. I mean, assuming if we have really big data, then this may not really be an issue. But in certain situations where we don't have that much data. Ideally, we would like to do this without having to constantly re-split our data. And therefore minimizing the training data set. So, this is why we use a K-Fold paradigm. Let's assume we have a training data set with four features x0, x1, x2, x3, and the y variable, or target. If we are use k-fold where k = 4, this is a hyper-parameter which is what to put here. We would make four different parts out of these datasets. Here I have put different colors, colors to each one of these parts. What we would do then in order to commence the training, is we will create an empty vector that has the same size as rows, as in the training data, but for now is empty. And then, for each one of the folds, we would start, we will take a subset of the training data. In this case, we will start with red, yellow, and green. We will train a model, and then we will take the blue part, and will make predictions. And we will take these predictions, and we will put them in the corresponding location in the prediction array which was empty. Now, we are going to repeat the same process always using this rotation. So, we are now going to use the blue, the yellow, and the green part, and we will keep to create a model, and we will keep the red part for prediction. Again, we will take these predictions and put it into the corresponding part in the prediction array. And we will repeat again with the yellow, and the green. Something that I need to mention is that the K-Fold doesn't need to be sequential as a date. So, it would have been shuffled. I did it as this way in order to illustrate it better. But once we have finished and we have generated a whole prediction for the whole training data, then we can use the whole training data, in order to fit one last model and make now predictions for the test data. Another way we could have done this is for each one of the four models we were making predictions for the validation data. At the same time, we could have been making predictions for the whole test data. And after four models, we will just take an average at the end. We'll just divide the test predictions by four. But a different way to do it

, I have found this way I just explained better with neural networks, and the method where you use the whole training data to generate predictions for test better with tree-based methods. So, once we finish the predictions with the test, you can start again with another model this time. So you will generate an empty prediction, you will stack it next to your previous one. And you will repeat the same process. You will essentially repeat this until you're finished with all models for the same layer. And then, this will become your new training data set and you will generally begin all over again if you have a new layer. This is generally the concept. Though we could say this, in order to extend on many layers, we use this K-Fold paradigm. However, normally, neural networks we have this notion of epochs. We have iterations which help us to re-calibrate the weights between the nodes. Here we don't have this option, the way stacking is. However, we can introduce this ability of revisiting the initial data through connections. So, a typical way to connect the nodes is the one we have already explored where you have it input nodes, each node is directly related with the nodes of the previous layer. Another way to do this is to say, a node is not only affected, connected with the nodes of the directly previous layer, but from all previous nodes from any previous layer. So, in order to illustrate this better, if you remember the example with the headmaster where he was using predictions from the teachers, he could have been using also predictions from the students at the same time. This actually can work quite well. And you can also refit the initial data. Not just the predictions, you can actually put your initial x data set, and append it to your predictions. This can work really well if you haven't made many models. So that way, you get the chance to revisit that initial training data, and try to capture more informations. And because we already have meta l-models present, the model tries to focus on where we can explore any new information. So in this kind of situation it works quite well. Also, this is very similar to target encoding or many encoding you've seen before where you use some part of the data, let's say, a code on a categorical column, given some cross-validation, you generate some estimates for the target variable. And then, you insert this into your training data. Okay, you don't stack it, as in you don't create a new column, but essentially you replace one column with hold out predictions of your target variable which is essentially very similar. You have created the logic for the target variable, and you are essentially inserting it into your training data idea.

.

.

[MUSIC] Hello everyone. In this video we will analyze
the Crowdflower competition. We, I mean me, Stanislav Semenov an
d
Dmitry Altukhov participated as a team and took second place. I
will explain most important parts of
our solution along with some details. The outline of the video i
s as follows. First, I will tell you about the contest
itself, what kind of data and metrics were provided. After that
we will discuss are approach,
features and tricks. And then I will summarize what is worth
to kind of look for on the competition. Some of the competition
were
organized by CrowdFlower Company. The goal of this competition i
s to measure
relevance of the search results given the queries and resulting
product
descriptions from living e-commerce sites. The task is to evalua
te the accuracy
of their search algorithms. The challenge of the competition is
to predict the relevance score of a given query and problem desc
ription. On the picture we see assessor's
user interface, which has a query, a search query, and
some information about an item. Assessors were asked to give eac
h
query-product pairing a score of 1, 2, 3, or 4, with 4 indicatin
g the item
completely satisfied the search query, and 1 indicating the item
 doesn't match. As the training data, we have only
median and variance of these scores. Data set consists of three
text fields,
request query, result and products title,
and product description, and two columns related to the target,
median and variance of scores. To ensure that the algorithm is r
obust
enough to handle any HTML snippets in the real world, the data p
rovided in
the program description field is raw and sometimes contain permi
ssions
that is relevant to the product. For example,
a string of text or object IDs. To discourage hand-labeling, the
 actual
set was augmented with extra data. This additional data was
ignored to when the public and private leaderboard
scores were calculated. And of course, campaign scores have
no idea which objects we had already used to calculate the score
s. So we have 10,000 samples in train, and
20,000 in test, but it's good data. I mean, validation works wel
l and local scores are close enough
to leaderboard scores. Effective solutions, non-standard metric
was used, quadratic weighted kappa. Let's take a closer look at
it. You can find a detailed description of the
metric on the competition evaluation page. We have already discu
ssed the metric
in our course, but let me recall it. Submissions are scored base
d on the
quadratic weighted kappa which measures the agreement between tw

o ratings. This metric typically will rise from 0,
random agreement between raters, to 1, complete agreement betwee
n raters. In case there is less agreement between
the raters than expected by random, the metric may go below 0. I
n order to understand the metric, let's consider an example
of how to calculate it. First, we need N by n confusion matrix C
,
which constructed and normalized. Our vertical axis by its failu
res,
or horizontal predicted failures. In our case, N is equal to 4 a
s
the number of possible ratings. Second we need N by n histogram
matrix of expected matrix E, which is calculated assuming that t
here
is no correlation between ratings cost. This is calculated as wi
thin
histogram vectors of ratings. Also we need N by N matrix of weig
hts, W, which is calculated based
on its elements positions. This particular formula for weights u
ses
square distance between indexes i and j, so this is why the meth
od is
called quadratic weighted kappa. Finally, kappa can be calculate
d as one
minus a fraction of this weighted sum of confusion matrix in the
 numerator, and weighted sum of expectation
matrix in the denominator. I want to notice that kappa has prope
rties
similar both to classification loss and regression loss. The mor
e distant predicted and true ratings are,
the more penalties there will be. Remember that thing,
we will use it later in our video. Okay, let's now talk
about my team solution. It's turned out to be quite complex,
consisting of several parts. Like extending of queries, per-quer
y
models, bumper features, and so on. I will tell you about main p
oints. So let's start with text features. We have three text fie
lds, a query,
a title, and a description. You can apply all techniques that
we discuss in our course and calculate various measures of simil
arity. That's exactly what we did. That is for query title and q
uery
description pair, we calculated the number of magic words, cosin
e distance
between TF-IDF representations, distance between the average wor
d2vec
vectors, and Levensthein distance. In fact, this is a standard s
et of
features that should be used for similar task. And there is noth
ing outstanding. The support should be
considered as a baseline. And now we turn to the interesting thi
ngs. In addition, we found it was
useful to use symbolic n-grams. You can work with them in
the same way as with word-based, if each letter is interpreted
as a separate word. We use symbolic n-grams from one letter,
to five letters. After getting a large parse metrics of

n-grams, we apply it as to them, and took only close to 300
combinations as features. You remember we discussed this portion
of our course, there is an example. Looking at the data we notic
e
three interesting properties. Queries are very short,
numbers of unique queries is 261, and queries are the same in tr
ain and
test sets. We decided to use these observations
to build extended versions of query as follows. For each query,
we get the most
relevant corresponding items, those with relevance equal to four
. We join all words from the title
of the relevant items and take ten most popular words. This is w
hat we called extended query and it's used to build all these te
xt
features that I mentioned earlier. Notice that this trick is app
licable
only within the context framework. Due to the fact that queries
in test
are exactly the same as in train. In real life we couldn't do so
because it's unrealistic to ignore
relevant results for every search query. The fact that sets of q
ueries in train and
test are the same, give us an opportunity to split our
problem into many small subtasks. Specifically, for each query,
you can build a separate model that will only predict relevance
of corresponding items. Again, in real life, such tricks can't b
e
applied, but within the context framework, it's totally fine. So
, for each unique query,
we get corresponding samples, calculate work to work similaritie
s,
back and forth presentation, and fit a random fourth classifier.
Finally, we have 261 model
which predictions were used as a feature in final example. Do re
member that every product
item is by several people. Median in a variance of the ratings.
The variance show in ratings
are inconsistent or not. Intuitively, if the variance is large,
then such an object need to be taken
into account with a smaller weight. One way to do it is to assig
n items
weight depending on query answers, which was a heuristics 1 /
1 + variance as the weight. Another method is to restore
individual observation using median and variance statistics. We
found that supposing
there are three assessors, we can almost always certainly
restore our original labels. For example,
if we have a median equal to three, and variance equal to 0.66,
there of course are two, three, and four, which by this approach
and for
each source sample, generated three once. However, using them as
training
data took longer to train, and did not improve the score. And si
mple heuristic works quite well,
and they use it in final solution. In the competition, you need

to choose
a metric, we need to predict class, but penalty for
the error is not symmetric. We decided to take it into account,
adding several artificially created binary
delimiters between classes as features. In other words, we're tr
ying to
classify to answer the question, is it true that target class nu
mber is
greater than 1, greater than 2, and so on. We call these feature
s bumpers, since they
are kind of separators between classes. We build them in fashion
, similar to
how we construct predictions instead. It was very useful for the
 final solution. All mentioned features will
be used in an ensemble. We build several models on
different subsets of features. Some feel relatively simple like
SVM and
some look quite complex. You can see the part of
complex model created by me. It uses bumper features,
all sorts of similarities and query features in different combin
ations. Also there is a lot of frostage models, which are mixed
up with
second stage random forest. In fact, each participant of the tea
m
made his own model, and then for competition we simply mixed our
model linearly for final submission. Let's remember that the met
ric on the one
hand has some proper classification It's necessary to predict th
e class. But for regression answer,
we can analyze more. We have built models for regression, but we
 have had to somehow turn
real-valued predictions into classes. A simple approach with wou
ld work poorly,
so we decided to pick up thresholds. For the purpose, we were ri
ght. Thresholds and choose the best one
weighted on cross-validation score. The buff procedure gave us a
 very
significant increase in quality. in fact it often happens in com
petitions
with non-standard metrics that [INAUDIBLE] grades symmetric opti
mization
gives a significant improvement. So let's sum up. In the competi
tion it was really
important to use the [INAUDIBLE] ideas. First symbolic and grams
, once since
they give a significant increase in the score and it was not sol
ved
that you should use that. Second, expansion of queries led to
significant increase in this course. Also optimization of thresh
olds was
a crucial part of our solution. I hope you will re-use some of t
hese
tricks in your competitions, though. Thank you for the attention
. [MUSIC][MUSIC] Hi, throughout the course, we use
the Springleaf competition as a useful example of EDA, mean enco
dings and

features based on nearest neighbors. Back then, we took the third place in
this competition together with and. And now in this video, I will describe
the last part of our solution, which is the usage of stacking and
ensembles. On this picture, you can see the final
stacking scheme we produced on the level 0 features, on the first level,
predictions by basic models. On the level one plus combination. So these predictions and
some accurately chosen features on the second level models
on this new set of features. And finally, on the third level,
their linear combination. In this video, we will go through each level as it builds
up to this non-trivial ensembled scheme. But first, let's quickly remind
ourselves about the problem. This was a binary classification
task with area under curve metric. We had 145,000 samples in training data
and about 2,000 anonymized features. These were useful insights
derived by us while doing EDA. And you can check out EDA done by earlier
in our course to refresh your memory. So now let's start with features. Here we have four packs of features. First two are the basic dataset and
the processed dataset. To keep it simple,
we just used insights derived from EDA to clean data [INAUDIBLE] and
to generate new features. For example,
we remove duplicated features and edit some feature interaction based
on scatter plots and correlations. Then, we mean-encoded all categorical
features using growth relation loop and sign data and smoothing. We further used the mean-encoded dataset
to calculate features based on nearest neighbors. Like, what is the least in
closest object of the class zero? And how many objects out of ten
nearest neighbors belong to class one? You can review how this could be done in related topics introduced
by Dmitri Altihof. So finally, these four packs of
feature were level 0 of our solution. And the second level was represented
by several different gradient within decision tree models,
and one neural network. The main idea here is that meta
features should be diverse. Each meta feature should bring
new information about the target. So we use both distinct parameters and
different sets of features for our models. For the neural network, we additionally
pre-processed features with common scalars, ranks and
power transformation. The problem there was in huge outliers
which skew network training results. So ranks and power transformation

helped to handle this problem. After producing meta features who is
gradual in boosting decision to it and neural networks, we calculated pay rise differences
on them to help next level models. Note that this is also an interesting
trick to force the model to utilize the differences in
the first level models predictions. Here we edit two datasets of features based on nearest neighbors. One was taken directly from level 0 and
they contain the same features. But it was calculated on the mean-encoded
dataset to the power of one-half. The point here was that these features
were not completely utilized by the first level models. And indeed, they brought new pieces
of information to this level. Now we already have autofold
predictions from the first level and we will train with the models on them. Because we could have target leakage
here because of other folk, and also because features not very good and there are almost no patterns left
in the data for models to discover. We chose simple classifiers, keeping in
mind that predictions should be diverse. We used four different models. Gradient boosted decision tree,
neural networks, random forest and logistic regression. So this is all with
the second level models. And finally, we took a linear in your combination of the second level models. Because a linear model is not inclined
to that we estimated coefficients directly using these four predictions and
our target for throwing in data. So, this is it. We just went through each level of this
stacking scheme and then the student. Why we need this kind of complexity? Well, usually it's because different
models utilize different patterns in the data and we want to unite all
of this patterns in one mighty model. And stacking can do exactly that for us. This may seem too complicated. Of course, it takes time to move up to
this kind of scheme in a competition. But be sure that after completion our course, you already have enough
knowledge about how to do this. These schemes never appear in the final
shape at the beginning of the competition. Most work here usually is
done on the first level. So you try to create diverse meta features
and unite them in one simple model. Usually, you start to create the high
grade second level of stacking, when you have only a few days left. And after that, you mostly work on
the improvement of this scheme. That said, you already have the required knowledge and now you just need to get
some practice out there. Be diligent, and without a doubt,

you will succeed. [SOUND] [MUSIC][MUSIC] Hi, in this video, I will tell you about Microsoft
Malware Classification Challenge. We were participating in a team with
other lecturers of this course, Mikhail Trofimov and Stanislav Semenov. We got the third place
in this competition. The plan for
the presentation is the following. We will start with the data description and a little bit of EGA. We'll then talk about feature extraction, then we will discuss how we did
feature processing and selection. We will see how we did the modeling,
and finally we will explain several tricks that allowed us
to get higher on the league of board. So let's start with problem description. In this competition, we will provided about half of terabyte
of malicious software executables. Each executable was
represented in two forms. The first is, HEX dump. HEX dump is just a representation of file
as a sequence of bytes, it is row format. The file on the disk is
stored as a sequence or bytes, that is exactly
what we see in HEX dump. The second form is a listing generated by
interactive disassembler, or IDA in short. IDA tries to convert low level
sequence of bytes from the HEX dump to a sequence of assembler columns. Take a look at the bottom screenshot. On the left,
we see sequences of bytes from HEX dump. And on the right,
we see what IDA has converted them to. The task was to classify malware
files into nine families. We will provide with train sets of about
10,000 examples with known labels, and a testing set of singular size. And the evaluation metric was
multi-class logarithmic loss. Note that the final loss in this competition was very, very low. The corresponding accuracy was more than
99.7%, it's enormously high accuracy. Remember, it is sometimes beneficial to examine metadata. In this competition, we were given
seven zip archives with files. And the filenames look like hash values. Actually, we did not find any
helpful metadata for our models, but it was interesting how train test
split was done by organizers. The organizers were using the first
letter of the file names for the split. On this plot, on the x axis, we see
the first characters of the file names. And on the y axis, we have the number of files with their
names, starting with the given character. The plots actually look cool, but
the only information we get from them is the train test split is in fact random,
and we cannot use them to improve our models. So in this competi

tion we
were given a raw data, so we needed to extract
the features ourselves. Let's see what features we extracted. We ll, in fact, no one from our
team had domain knowledge or previous experience in
malware classification. We did not even know
what IDA disassembly is. So we started really simple. Remember i ssue executable in the dataset
was represented as two files. So our first two features were
the sizes of those files, or equivalently their length. And surp risingly we got 88% accuracy
just by using these two features. On the plot you see x axis cor respond to
an index of an object in the train set. We actually sorted the o bjects
by their class here. And the y axis shows the file sizes
of aHEX dump file for every object. And we see this feature
is quite demonstrative. The most simple features we
could derive out of sequence are account of their elements, righ t? So this is basically what function
value comes from pandas does. So it is what we did. We counted b ytes in HEX dump files,
and that is how we get 257 features. And 257 is because we
have 256 byte values, and there was one special symbol. We achie ved almost 99%
accuracy using those features. At the same time that we started to
read about this assembly format, and papers on malware classific ation. And so, we got an idea what
feature is to generate. We looked into this assembly file. We us e regular expressions
to extract various things. Many of the features we extracted
were thrown away immediately, some were really good. And what we did actually first,
we counted system calls. You see that on the slide, they are als o called API calls in
Windows as we read in the paper. And here is the error rate we g ot
with this feature, it's pretty good. We counted assembler common s like move,
push, goal, and assembler,
they work rather well. We also try to extract common
sequences like common end grams and extract features out of them , but
we didn't manage to get a good result. The best features we
had were section count. Just count the number of
lines in this assembly file, which start with .text or
.data, or something like that. The classification accuracy with
that feature was more than 99% using these features. By incorpor ating all of these, we were
able to get an error rate less than 0.5%. From our text mining e xperience, we knew that n-grams could be
a good discriminative feature. So we computed big grams. We foun d a paper which
proposed to use 4-grams. We computed them too, and we even
went further and extracted 10-grams. Of course, we couldn't work

with 10
grams directly, so we performed a nice feature selection, which was one of the
most creative ideas for this competition. We will see later in this
video how exactly we did it. Interestingly, just by applying
the feature selection to 10-grams and fitting in XGBoost,
we could get a place in the top 30. Another feature we found
interesting is an entropy. We computed entropy of small
segments of wide sequence by moving the sliding window
over the sequence. And computing entropy inside each window. So we've got another sequence that could contain an explicit information about
the encrypted parts of the executable. See, we expect the entropy to be
high if the data is encrypted and compressed, and
low if the data is structured. So the motivation is that some malwares
are injected into a normal executables, and they are stored in encrypted format. When the program starts, the malware
extracts itself in background first and then executes. So entropy features could
kind of help us to detect and encrypt the trojans in the executables,
and thus, detect some classes of malware. But we got an entropy
sequence of variable length. We couldn't use those
sequences as features, right? So we generated some statistics of entropy
distribution like mean and median. And also we computed a 20 percentiles and
inverse percentiles, and use them as features. The same features were extracted out
of entropy changes, that is we first apply diff function to entropy sequence
and then compute the statistics. It was possible to extract three
things from hex dump by looking for printable sequences that
ends with 0 element. We didn't use strings themselves,
but we computed strings lengths. Distribution for each file and
extract the similar statistics, the statistics that we extracted
from entropy sequences. Okay, we finished with feature extraction. So let's see how we pre-process them and
perform feature selection. Those moment when we
generated a lot of features. We wanted to incorporate all of them
in the classifier, but we could not fit the classifier efficiently when
we got say 20,000 features. Most features will probably useless, so we tried different feature selection
method and transformation techniques. Let's consider the buys counts features. There are 257 of them, not much, but
probably there is still a redundancy. We tried non-negative
matrix factorization, NMF. And the principal component analysis,
PCA, in order to remove this redundancy. I will remind you that both NMF and
PCA are trying to factorize object feature matrix x into

the product of two low-rank matrices. You can think of that of a
s finding a small number of basis vectors in the
feature space, so that every object can be approximated by a lin
ear
combination of those basis vectors. And this coefficients of
approximation can be treated as new features for each object. So
 the only difference between NMF and
PCA is that NMF requires all the components of floor rank
matrices to be non-negative. We set the number of basis vectors
to 15,
and here we see plots between
the first two extracted features. One for NMF and one for PCA. S
o these are the coefficient for
most important basis vectors actually. We used 3D based model an
d it is obvious that NMF features
were a lot better for trees. So NMF works good when the non-nega
tivity
of the data is essential. And in our case we worked with counts,
which are non-negative by nature. Simple trick to get another pa
ck of
features by doing almost nothing is to apply a log
transform to the data and calculate NMF on
the transformed data again. Let's think what happened here. NMF
protest originally uses MSE laws to measure the quality of
approximation it build. This trick implicitly changes the laws
NMF optimizes from MSE to RMSLE. Just recall that RMSLE
is MSE in the log space. So now the decomposition process
pays attention to different things due to different loss, and
thus produces different features. We used the small pipeline
to select 4-grams features. We removed rare features, applied
linear SVM to the L1 regularization, as such model tends to sele
ct features. And after that,
we thresholded random forest feature importances to get final fe
ature
set of only 131 feature. The pipeline was a little bit
more complicated with 10-grams. First, we used the hashing to re
use
the dimensionality of original features. We actually did it onli
ne
while computing 10-grams. We then selected about 800 features
based on L1 regularize SVM and the RandomForest importances. Thi
s is how we got about 800 features
instead of 2 to the power of 28. But we went even further. This
is actually the most
interesting part for me. The main problem with feature selection
, that I've just described,
is that we've done it for 10-grams independently of other featur
es
that we already had to that moment. After selection,
we could end up with really good features. But those features co
uld
contain the same information that we already had in other featur
es. And we actually wanted to improve
our features with 10-grams. So here is what we did instead. We g
enerated out of fold prediction for the train set using all
the features that we had. We sorted the object by their true cla

ss
predicted probability and try to find the features that would se
parate the most
error prone object from the others. So actually, we use another
model for it. We've created a new data set with
error prone objects having label 1, and others having label 0. W
e trained random forest and selected 14 10-grams, well,
actually hashes to be precise. We had a nice performance
increase on the leaderboard when incorporating those 14 features
. This method actually could
lead us to sever overfitting, but it fortunately worked out nice
ly. All right, let's get to modeling. We didn't use stacking in
this
competition as we usually do now. It became popular slightly
after this competition. We first used Random Forest everywhere,
but it turned out it needs to calibrated for
log loss. So we switched to XGBoost and
used it for all our modeling. Every person in the team
extracted his own features and trained his own models
with his own parameters. So our final solution was
a combination of diverse models. We found bagging worked
quite well in this data set. We even did the bagging
in very peculiar way. We concatenated our twin set
with a seven times larger set of object sampled from train
set with the replacement. And we used the resulting data set tha
t is eight times larger than
original train set, to train XGBoost. Of course, we averaged abo
ut 20
models to account for randomness. And finally, let's talk about
several
more tricks we used in this competition. The accuracy of the mod
els was quite high,
and we all know that the more
data we have the better. So we've decided to try to use
testing data for training our models. We just need some labels f
or
the testers, right? We either use predicted class or we sample t
he label from
the predicted class distribution. Generated test set labels
are usually called pseudo labels. So how do we perform cross
validation with pseudo labels? We split our train set into fold
as we usually do well performing cross validation. In this examp
le,
we split data in two folds. But what's different in cross valida
tion
is that now, before training the model in a particular fold, we
can concatenate
this fold with the test data. Then we switch the faults and
do the same thing. See we trained the objects to
denoted with green color and predict the once shown with red. Ok
ay, and to get predictions for
the test set, we again do kind of cross validation
thing, like on the previous slide. But now we divide the test
set in faults and concatenate train set to each
fold of the the test set. In the end, we get out of all predicti
ons

for the test set, that is our submission. One of the crucial things to
understand about this method, that sometimes we need two
different models for it. And this is the case where one model
is very confident in its predictions. That is, the predictive probabilities
are very close to 0 and 1. In this case, if we train a model,
predict task set, sample labels or
take it the most probable label and retrim same model with the same
features or get no improvement at all. We just did not introduce any information
with pseudo labeling in this way, but If I ask say, Stanislav,
to predict that with his model. And then I use his labels for a
test and create my model, that is where
I will get a nice improvement. This actually becomes just another way to incorporate knowledge
from two diverse models. And the last thing that helped
us a lot is per-class mixing. At the time of the competition,
people usually mixed models linearly. We went further and mixed models joining
coefficients for each class separately. In fact, if you think about it,
it's very similar to stacking with a linear model of a special
kind at a second level. But the second became popular in
a month after this competition, and what we did here is a very
simplest manual form of stacking. We published our source code online, so if
you are interested you can check it out. All right,
this was an interesting competition. It was challenging from technical
view point as we needed to manipulate more than half of terabyte of data,
but it was very interesting. The data was given in a raw format
and the actual challenge was to
come up with nice features. And that is where we could be creative,
thank you. [MUSIC]Hi. In this video, I'm going to talk about Walmart trip type classification challenge which was held in Kaggle couple of years ago. I won the first place in that competition. And now, I will tell you about most interesting parts of the problem and about my solution. That said, this presentation consists of four parts. First, we will state the problem. Second,we will understand what data format and data reprocessing. And third, we will talk about models, their relative quality and their relation to the general staking scheme. And finally, we will overview some possibilities to generate new features here. So, let's start. In our data, we had purchases people made in Walmart visiting their shop in two weeks, and we had to classify them into 38 visiting trip types or classes. Let's take a quick look at features in the data. Trip type column represents the target, visit number represents ID which unites purchases made by one customer in one shopping trip. For example, a customer which made visit number seven, purchased two items which are located in the second and in the third lines of this data frame. Notice that all rows with the same visit number have the same trip type. An importa

nt moment, is that we have to predict a trip type for visit numb er, and not for each row in the train data. And as you can see, in the train we have around 647,000 rows and only 95,000 visits. Back to the features, next feature is weekday which obviously r epresents the weekday of the visit. Next is UPC. UPC is an exact ID of a purchased item. Then, scan count. Scan count is the exa ct number of items purchased. Note that minus one here represent s not the purchase but the return. Next features, department des cription, with 68 unique values is a broad category for an item. And finally, fineline number, with around 5000 unique values, i s a more refined category for an item. So, after we understood w hat this feature represents, let's recall that we have to make o ne prediction for each visit number. Let's take a look at the da ta for the visit number eight. We can see here that this particu lar visit has a lot of purchases in category paint and accessori es, which means that trip type number 26 may represent a visit w ith most purchases in that category. Now, how to approach model train in here. Let's take another look at the data and assess ou r possibilities. Should we predict trip type for each item on th e list or should we choose another way? Of course both of them a re possible, but in the first one, we'll predict trip type for e ach row with each data set, we'll miss important interactions be tween items which belong to the same visit. For example, trip ty pe may have a number of 26, if more than half of its items are f rom paint and accessories. But, if we will not account for inter action between these items, it can be quite hard to predict. So, the second option of uniting all purchase in the visit and maki ng a data set where each row represents a complete visit, seems more reasonable. And, as can be expected, this approach leads to more significant benefits in the competition. I'm going to show you the easiest way to change the data format to the desired on e. Let's choose the department description feature for the purpo se of an example. First, let's group the data frame by visit num ber and calculate how many times each department description is present in a visit. Then, let's unstack last group by column so we will get a unique column for each department description valu e. Now, this is the format we wanted. Each row represents a visi t and each column is a feature described in that visit. We can u se this group by approach for other features besides department description. Also note that items in the visit are actually very similar to words in a text. After our confirmation, each featur e here represents counts, so we could apply ideas which usually works with text, for example, tf-idf transformation. As you can guess, a lot of possibilities emerge here. Great. After this is done and we process data in the desired format, let's move to ch oosing a model. Based on what we already have discussed, can you guess if we should expect the significant difference in scores between linear models and tree-based models here? Think about th is a bit. For example, is there a reason why linear models will under perform in comparison to tree based-models? Yes, there is. Again, I'm talking about interactions here. Indeed, tree-based models in neural network have significant superiority in quality in this competition for this very reason. But still, one can us e linear models and TNN to produce useful method features here. Despite the fact that they didn't imply interactions, they were a valuable asset in my general staking scheme. I will not go int

o further details of staking here because we already covered mos t ideas in other videos about competitions. Instead, we'll talk a bit about feature generation. Except for interactions between items purchased in one visit, one could try to exploit interacti ons between features. The interesting and unexpected result here was that one fineline number can belong to multiple department descriptions, which means that fineline number is not a more det ailed department description as you can think. Using this intera ction, one can further improve his model. Another interesting fe ature generation idea was connected to the time structure of the data. Take a look at this plot, it represents the change in the weekday feature relative to the row number. It looks like the d ata is ordered by time here. And the data appears to consist of 31 days, but train test split wasn't time based. So, you could d erive features like day number in the data set, number of a visi t in a day, and the total amount of visits in a day. So, this is it. We just discussed the most interesting parts of this compet ition. Changing the data format to a more suitable, generating f eatures while doing sold, working with models while doing stacki ng. And finally, doing some for additional feature engineering. The challenge itself proved useful and interesting. And I would recommend you to check it out and try approaches we have talked about.Hello everyone. Today, I will explain to you how, me and m y team mate Gert, we won a very special Kaggle competition calle d the Acquired Valued Shoppers Challenge. First let me give you some background about the competition. It was a recommender's ch allenge. And when I say a recommender's challenge, I mean you ha ve a list of products and a list of customers, and you try to br ing them together. So we target them so we recommend which custo mer in order to increase sales loyalty or something else. There were around 1,000 teams and for back then, at least they were qu ite a lot. Now Kaggle has become much more popular. But back the n, given the size of the data, which was quite big, I think this competition attracted a lot of attention. And as I said, we att ained first place with my teammate Gert, for what it was, I thin k, a very clear win because we took a lead early in the competit ion, and we maintained it. And that solution was not actually ve ry machine learning Kebbi, but it was focused on really trying t o understand the problem well and find ways to validate properly . And in general, it was very, very focused on getting sensible results. And that's why I think it's really valuable to explain what we did. So what was the actual problem we tried to solve? I magine you have 310,000 shoppers, almost equally split. I'm on a train and test. You have all their transactions from a point wh ere they were given an offer, and these were about 350 million t ransactions. So, one year of all the transactions of all the cus tomers involved in this challenge, and you have 37 different off ers. When I say offer here, it is actually a coupon. So, a shopp er is sent normally. It's a piece of paper that says, "If you bu y this item, you can get a discount." So it recommends to certai n item. Now, we don't know exactly what discount is. Maybe disco unts, maybe it says, "You can buy another item for free." So, ma ybe a different promotion, but in principle is some sort of ince ntive for you to buy this item. I have mentioned items so far or products, but the notion of product was not actually present in this challenge, but we could infer it. We could say that a uniq

ue combination of a brand, category, and company that were prese nt could form a product, at least for this competition. Let me g ive you a bit more details about the actual problem we are tryin g to solve. Imagine you have a timeline, starts from one year in the past until one year after. So, in the past, a customer make s a visit to the shop, buys certain items, leaves, comes back an other day, makes another visit, buys more items. Then at some po int, he or she is targeted with an offer, a coupon as I said. Al l transactional history for this customer stops there. You don't know anything else. Up to this point, you know everything for o ne year back up to this. The only thing you know is, you know th at the customer has indeed redeemed that coupon. So he did buy t he offer product. And for that training data only, you also have a special flag that tells you whether he or she bought that ite m again. And you can see why this is valuable. Because normally, when you target someone with a coupon, you give a discount, and you don't make that much profit actually, but you aim in establ ishing a long-term relationship with the customer. And that's wh y they were really interested in getting a model here that could predict which recommendation, which offer will have that effect , will make a customer develop a habit in buying this item. And what we were being tested on was AUC. By this point, I expect yo u roughly know what AUC is as a metric, but in principle, you're interested in how well your score discriminates between those t hat bought or can buy the item again and those that will not. So , when you have the highest score, you expect higher chance to b uy the item, and a lower score, lower attempts to buy the item a gain. So, higher AUC means that this discrimination is stronger with your score. This was a challenging challenge. And it was ch allenging because, first of all, the datasets were quite big. As I said, 350 million transactions. Me and Gert, we didn't have c razy resources back then. I have to admit that I have personally improved my hardware since then, but actually back then, I was working only with a laptop that had 32-bit Windows and only four gigabytes of RAM. So, really small and mainly challenging that we had to deal with these client files. And then, we didn't have features. So, what we knew is this customer was given this offe r, which is consistent by these three elements I mentioned befor e, category, brand, and company, and the time that this recommen dation was made nothing else. Then, you had to go to the transac tional history and try to generate features. And you know, anybo dy could create really anything they wanted. There was not a cle ar answer about which features would work best. So this is not y our typical challenge where you're normally given the thesis. Bu t it is quite difficult for the type of the recommender's challe nge. And what really makes this competition difficult, interesti ng, and what I think at the end of the day gave us the win was t he fact that the testing environment was very irregular. And we can define irregular, in this context, as an environment where t he train data and the test data had different customers. So, no overlaps. Different customers, and one different in the other. A lso, the training this data had in general different offers. It was showing you a graph that shows that the distribution of its offer and whether it appears in the train or in the test data or both. And you can see that most offers, either appear only in t est or they appear only in train with minimal overlap. So, that

makes it a bit difficult because you basically have to make a mo
del with soft products. They were offering the train, but in the
test data, you have completely other offers. So you don't know
how they would behave as these products have never been offered
before. And the last element is, the test data is obviously in t
he future. That is expected. But given the other elements, this
makes it more difficult, especially in some cases were well in t
he future. And some of it is not as important elements, but stil
l crucial was that this challenge was focusing on acquisition. S
o, there is not that much history between the customer and the o
ffered product. And I say this is irregular because grocery sale
s are in principle based on what the customer already like and h
as bought many times in the past. So we referred to these type o
f acquisition problem, where we don't have much history, as the
cold start problem, and it is much more challenging because you
don't have that direct link. That's, the customer really like th
is product I made an offer because we don't have a past history
that can verify this or we don't have much history. And the last
element is that if you actually see the propensity of an offer
to be bought, again in the training data, the results were quite
different. And here, I give you the offer by shortened propensi
ty, and you can see some offers had much success to be bought ag
ain. It's like offer two that somehow this had much less. For ex
ample, 20 percent, and this is just a sample. There were some ot
her offers that had around five percent. So, if you put now ever
ything into the context, you have different customer, different
offers, different buyer, different time periods. In principle, y
ou don't have that much information about the customer and the o
ffer product, and you know that the offers of the training data
are actually quite different. It's really difficult to get a sta
ndard pattern here. And you know that the offers in the test dat
a are going to be different. So, all this made it a difficult pr
oblem to solve or in irregular environment. How did we handle bi
g data? We did it with indexing. And the way I did the indexing
was, I saw that the data were already shorted by customer and ti
me. So, I passed through these big data file of transactions, an
d every time I encountered a new customer, I created a new file.
So I created a different file for each customer that contained
all his or her transactions, and that made it really easy to gen
erate features, because if I have to generate features for a spe
cific customer, I would just access this file and create all the
features I wanted at the will. This is also very scalable. So I
could create threads to do this in parallel. So, access many cu
stomers in parallel. And I did this not only for every customer,
but also for every category, brand, and company. So, every time
I wanted to access information, I would just access the right c
ategory, the right brand, or the right customer, and I will get
the information I wanted, and that made it very quick to handle
all these big chunks of data. But what I think was the most cruc
ial thing is how we handle this irregularity. I think at the end
of the day, this is what determines our victory because once we
got this right and we were able to try all sorts of things and
we had the confidence that it will work in the test data. The fi
rst thing we tried to do and this is something that I want you t
o really understand, is how we can replicate internally what we
are being tested on. That's really important. I'll give you the

room to try all these things. Try all different permutations and combinations of data techniques, anything you have you can put in mind, and really understand what works and what's not. So, we tried to do that. The first of attempt didn't go very well. So we try to randomly split the data between train and validation and we're trying to make certain that each offer is represented equally in each one of this train and validation data set proportionately equally. But was that correct? I mean, if you think about it. What we were doing there, we were saying I'm building a model with some offers and I'm validating in the same offers. That's good. Maybe we can do well here. But is this what we're really being tested on? No. Because in the test data, we'll have completely different offers. So, this didn't work very well. This was giving very nice internal results but not very good results in the test data. So, we tried something else. Can we leave one offer out? And I'm showing you roughly what this look like. So, for every offer, can we put an offer in the validation data and use all the cases of every other offer to train a model? So, if we were to predict offer 16, we will use all customers that received offer 1 to 15 and 17 to 24 to build the model and then we'll make predictions for all those customers that received offer 16. And you can see that this actually is quite close to what you're being tested on because you know you're building a model with some offers but, you're being tested on some other offer that is not there. And you can take the average of all these 24 users and I put 24 because this is how many offers you really have in the training data. You can take that average and that average may be much more close to the reality, close to what you were being tested on. And this was true. This gave better results, but we were still not there. And I'll show you why we were not there. Consider the following problem. Here, I'll give you a small sample of predictions for offer two and what was the actual target? What we see here is a perfect AUC score. Why? Because all our positive cases that are labeled with one and they have the green color, have higher score than the red ones, where the target is zero. So, the discrimination here is perfect. We have a point, a cutoff point. We can set 0.5 here where all cases that have score higher than this. We can safely say they are one and that is true and everything that has a score lower than this are zero. So, you see here one discrimination is perfect. Let's now take a sample from offer four. If you remember offer four, had in general lower propensity. Offer two had around 0.5 and offer four had around 0.2. So, it's mean we're center much lower and what you can see here is that, again, AUC is perfect for this sample because again, all the higher scores that are labeled with green have a target of one. And then the lower scores, everything that has a score less than 0.18 has a target of 0. The discrimination is perfect. We can find this cutoff point. We can say 0.8, where everything that has a score higher than this can safely be set to one. And that is always true. And vice versa, everything that's less than 18, then it's a 0. And that is always true. So, we have two scores. They discriminate really well between the good and the bad cases. However, we are not tested on the AUC of one offer. We are tested on the AUC of all offers together. So the test data have many offers. So, you are interested in the score that generalizes well against any offer. So, what happens if we try

to merge this table? AUC is no longer perfect and why this happens? Because some of the negative cases of the first offer had higher score than the positive cases, those that have a target equal to one from the second offer. So you can see, although the discrimination internally is really good, they don't blend that well. You lose something from that ability of your score to discriminate between ones and zeros. And the moment we saw this, we knew that just leaving one offer out was not going to be enough. We had to make certain that when we merge all those scores together, the score is still good. The ability of our model to discriminate is still powerful or it doesn't lose. And that's why we use a combination of the previous average AUC of all the offers and the AUC after doing this concatenation. So, the average of the two AUCs which really the metric we try to optimize because we thought that this is actually very close to what we were being tested on. And here I can show you the result of all our attempts and this is with a small subset of features because by that point, we were not interested to create the best features, we were interested to test which approach works best. So, you can see if you do it standard stratified K-fold, you can get much nicer results in internal cross-validation but in the test data, the relationship is almost opposite. So, highest score in cross-validation leads to worse results in the test data. And you can see why because you're not internally modeling or internally validating or on what you are actually being tested on. Doing the one-offer out keep obviously lower internal cross-validation results and better performance in the test data, but even better was doing this leave-one-offer plus one concatenation in the end. And this AUC was lower but actually had better test results. I believe we could get even better results if we made certain that we are also validating in new customers. But we didn't actually do this because we saw that this approach had already good results. But as a means to improve, we could have also made certains that we validate on different customers because this is what the test was like.

.

.

This tutorial is part of the Learn Machine Learning series. In t
his step, you will learn what data leakage is and how to prevent
 it.
What is Data Leakage

Data leakage is one of the most important issues for a data scie
ntist to understand. If you don't know how to prevent it, leakag
e will come up frequently, and it will ruin your models in the m
ost subtle and dangerous ways. Specifically, leakage causes a mo
del to look accurate until you start making decisions with the m
odel, and then the model becomes very inaccurate. This tutorial
will show you what leakage is and how to avoid it.

There are two main types of leakage: Leaky Predictors and a Leak
y Validation Strategies.
Leaky Predictors

This occurs when your predictors include data that will not be a
vailable at the time you make predictions.

For example, imagine you want to predict who will get sick with
pneumonia. The top few rows of your raw data might look like thi
s:

got_pneumonia	age	weight	male	took_antibiotic_medicine	...
False	65	100	False	False	...
False	72	130	True	False	...
True	58	100	False	True	...

-

People take antibiotic medicines after getting pneumonia in orde
r to recover. So the raw data shows a strong relationship betwee
n those columns. But took_antibiotic_medicine is frequently chan
ged after the value for got_pneumonia is determined. This is tar
get leakage.

The model would see that anyone who has a value of False for too
k_antibiotic_medicine didn't have pneumonia. Validation data com
es from the same source, so the pattern will repeat itself in va
lidation, and the model will have great validation (or cross-val
idation) scores. But the model will be very inaccurate when subs
equently deployed in the real world.

To prevent this type of data leakage, any variable updated (or c
reated) after the target value is realized should be excluded. B
ecause when we use this model to make new predictions, that data
 won't be available to the model.

Leaky Data Graphic
Leaky Validation Strategy

A much different type of leak occurs when you aren't careful dis
tinguishing training data from validation data. For example, thi

s happens if you run preprocessing (like fitting the Imputer for missing values) before calling train_test_split. Validation is meant to be a measure of how the model does on data it hasn't considered before. You can corrupt this process in subtle ways if the validation data affects the preprocessing behavoir.. The end result? Your model will get very good validation scores, giving you great confidence in it, but perform poorly when you deploy it to make decisions.

Preventing Leaky Predictors

There is no single solution that universally prevents leaky predictors. It requires knowledge about your data, case-specific inspection and common sense.

However, leaky predictors frequently have high statistical correlations to the target. So two tactics to keep in mind:

To screen for possible leaky predictors, look for columns that are statistically correlated to your target.
If you build a model and find it extremely accurate, you likely have a leakage problem.

Preventing Leaky Validation Strategies

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 use pipelines and do your preprocessing inside the pipeline.

Example

We will use a small dataset about credit card applications, and we will build a model predicting which applications were accepted (stored in a variable called card). Here is a look at the data:

```
import pandas as pd

data = pd.read_csv('../input/AER_credit_card_data.csv',
                   true_values = ['yes'],
                   false_values = ['no'])
print(data.head())
```

```
   card  reports       age  income     share  expenditure  owner
  selfemp  \
0  True        0  37.66667  4.5200  0.033270   124.983300   True
    False
1  True        0  33.25000  2.4200  0.005217     9.854167  False
    False
2  True        0  33.66667  4.5000  0.004156    15.000000   True
    False
3  True        0  30.50000  2.5400  0.065214   137.869200  False
    False
4  True        0  32.16667  9.7867  0.067051   546.503300   True
    False
```

```
   dependents  months  majorcards  active
0           3      54           1      12
1           3      34           1      13
2           4      58           1       5
3           0      25           1       7
4           2      64           1       5
```

We can see with data.shape that this is a small dataset (1312 ro
ws), so we should use cross-validation to ensure accurate measur
es of model quality

```
data.shape
```

```
(1319, 12)
```

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

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

# Since there was no preprocessing, we didn't need a pipeline he
re. Used anyway as best practice
modeling_pipeline = make_pipeline(RandomForestClassifier())
cv_scores = cross_val_score(modeling_pipeline, X, y, scoring='ac
curacy')
print("Cross-val accuracy: %f" %cv_scores.mean())
```

```
Cross-val accuracy: 0.979528
```

With experience, you'll find that it's very rare to find models
that are accurate 98% of the time. It happens, but it's rare eno
ugh that we should inspect the data more closely to see if it is
 target leakage.

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

    card: Dummy variable, 1 if application for credit card accep
ted, 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 in
come
    expenditure: Average monthly credit card expenditure
    owner: 1 if owns their home, 0 if rent
    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

A few variables look suspicious. For example, does expenditure m

```
ean expenditure on this card or on cards used before appying?

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

expenditures_cardholders = data.expenditure[data.card]
expenditures_noncardholders = data.expenditure[~data.card]

print('Fraction of those who received a card with no expenditure
s: %.2f' \
      %(( expenditures_cardholders == 0).mean()))
print('Fraction of those who received a card with no expenditure
s: %.2f' \
      %((expenditures_noncardholders == 0).mean()))

Fraction of those who received a card with no expenditures: 0.02
Fraction of those who received a card with no expenditures: 1.00

Everyone with card == False had no expenditures, while only 2% o
f those with card == True had no expenditures. It's not surprisi
ng that our model appeared to have a high accuracy. But this see
ms a data leak, 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, majorcards are a little les
s clear, but from the description, they sound concerning. In mos
t situations, it's better to be safe than sorry if you can't tra
ck down the people who created the data to find out more.

We would run a model without leakage as follows:

potential_leaks = ['expenditure', 'share', 'active', 'majorcards
']
X2 = X.drop(potential_leaks, axis=1)
cv_scores = cross_val_score(modeling_pipeline, X2, y, scoring='a
ccuracy')
print("Cross-val accuracy: %f" %cv_scores.mean())

Cross-val accuracy: 0.806677

This accuracy is quite a bit lower, which on the one hand is dis
appointing. However, we can expect it to be right about 80% of t
he time when used on new applications, whereas the leaky model w
ould likely do much worse then that (even in spite of it's highe
r apparent score in cross-validation.).
Conclusion

Data leakage can be multi-million dollar mistake in many data sc
ience applications. Careful separation of training and validatio
n data is a first step, and pipelines can help implement this se
paration. Leaking predictors are a more frequent issue, and leak
ing predictors are harder to track down. A combination of cautio
n, common sense and data exploration can help identify leaking p
redictors so you remove them from your model.
Exercise
```

Review the data in your ongoing project. Are there any predictors that may cause leakage? As a hint, most datasets from Kaggle competitions don't have these variables. Once you get past those carefully curated datasets, this becomes a common issue.

Click here to return the main page for Learning Machine Learning.

Leakage
Introduction

Leakage is one of the scariest things in machine learning (particularly competitions). Leakage makes your models look good, until you put them into production and realize that they're actually roundly terrible. To quote the Kaggle wiki entry on the subject:

> Data Leakage is the creation of unexpected additional information in the training data, allowing a model or machine learning algorithm to make unrealistically good predictions.
>
> Leakage is a pervasive challenge in applied machine learning, causing models to over-represent their generalization error and often rendering them useless in the real world. It can caused by human or mechanical error, and can be intentional or unintentional in both cases.

Leakage is particularly bad because it invalidates or weakens cross validation scoring. The accuracy of cross validation as a prediction for how well our model will do in validation or on production data is incredibly important; so much so that it's often said that "above all, trust your CV". If we undermine that, we undermine most of the tools and techniques in our toolbox!
Target leakage

The most obvious form of leakage is when a variable in a dataset is derived from the target variable in some way. For example, if we are predicting annual_gdp, a column with GDP in 2016 dollars, standardized_gdp, would be an example of a leak, because it's just the same data transformed a little bit. In order to build a real model and not a linear transform, we would need to remove this column from our model entirely. Again from the Kaggle wiki:

> One concrete example we've seen occurred in a prostrate cancer dataset. Hidden among hundreds of variables in the training data was a variable named PROSSURG. It turned out this represented whether the patient had received prostate surgery, an incredibly predictive but out-of-scope value.
>
> The resulting model was highly predictive of whether the patient had prostate cancer but was useless for making predictions on new patients.

With some practice working with and inspecting machine learning

features, this kind of "variable leak" is catchable, but it becomes tedious when the feature matrix has enough predictors in it. Domain knowledge helps a ton here.

Out-of-core leakage

Leakage is the number one problem in machine learning competitions because it can be weaponized by model-makers in a way that would never make sense in a production system. This is "out-of-core leakage". For an example of what this looks like, see this old Kaggle post explaining why one leak caused a competition identifying right whales to be reset. They're very challenging to catch because even experienced competition-runners (like the Kaggle team) can't match the time and depth competitors can bring to probing datasets for weaknesses.

Knowledge leakage

Which brings us to knowledge leakage, which is what I want to cover in more depth in this notebook. I'll actually just be going over the information presented in this fantastic blog post on the subject, so you should probably read that first.

To guard against overfitting, machine learning relies heavily on cross validation and related holdout and parameter search schemes. The effectiveness of the technique relies on our building a model on a training data, then testing it for fitness on training data that it's never seen before.

This is only an effective technique if we can prevent information about our test data from leaking into our training data. In theory this is easy: just don't use observations from the test data in the training data. However, there are things we can do during the pre-processing before we train a model that injects information about our test data into the training process! Doing this will increase our cross validation accuracy on the data we train on, but will worsen our accuracy in practice on validation or production data.

Let's demo how this can happen (NB: we're reimplementing the blog post code here; some things have changed in the library in the meanwhile however, so this code is a little different from that which originally ran).

We'll build a 100×10000

feature matrix: that is, 100 observations across 10000 synthetic features. This is a massively overdetermined feature matrix. Then we'll perform feature selection: we'll measure the correlation of each of the columns with the target column, and take the top two scorers as our model inputs. We'll train on those, and measure what our mean squared error (MSE) is (for more on model fit metrics click here).

```
import numpy as np
import pandas as pd
import scipy.stats as st
```

```
np.random.seed(0)
df = np.random.randint(0,10,size=[100,10000])
y = np.random.randint(0,2,size=100)
df = pd.DataFrame(df)
X = df.values

corr = np.abs(
    np.array([st.pearsonr(X[:, i], y)[0] for i in range(X.shape[
1])])
)
corrmax_indices = np.argpartition(np.abs(corr), -2)[-2:]

X_selected = X[:, corrmax_indices]

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

clf = LogisticRegression()
clf.fit(X_selected, y)
mse = cross_val_score(clf, X_selected, y, cv=10, scoring='neg_me
an_squared_error')

pd.Series(mse).abs().mean()

0.24989898989898984
```

Our mean squared error is pretty good, and we trust our CV, so w e think this is a result reflective of practical performace. How ever, is it really? Can you spot the error?

It's subtle. The reason we picked a matrix with so many features is because it accentuates the error we've made with the procedu re here. By measuring the correlation of the columns and taking the two highest scorers before doing cross validation, we actual ly injected incidental information about which variables are mos t highly correlated in both the train and test sets. Hence when we run the cross validation, we've "pre-selected" incidental cor relation that we know beforehand performs well in the test set.

We picked a lot of variables to make this effect easily noticabl e (with 10000 variables, some of them are going to end up quite correlated with the target). We can see how strong of an effect this creates by doing this same variable selection after a train -test split:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_test, y_train, y_test = train_test_split(X, y, test_s
ize=0.4)

corr = np.abs(
    np.array([st.pearsonr(X_train[:, i], y_train)[0] for i in ra
nge(df.shape[1])])
)
corrmax_indices = np.argpartition(np.abs(corr), -2)[-2:]
```

```
X_selected = X_train[:, corrmax_indices]

clf = LogisticRegression()
clf.fit(X_selected, y_train)
y_hat = clf.predict(X_test[:, corrmax_indices])
mean_squared_error(y_test, y_hat)
```

```
0.45000000000000001
```

It looks like knowledge leaking almost halved our mean squared e
rror!

The correct approach to dealing with this problem is to think ha
rder about how we will structure our pipeline. Best-fit variable
 selection like this should live inside of our cross validation;
 that is, it should only be done after we've already done train-
test splitting. This will at least give us a more realistic inde
x on performance:

```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import StratifiedKFold

kf = StratifiedKFold(n_splits=10)

mse_results = []

for train_index, test_index in kf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    corr = np.abs(
        np.array([st.pearsonr(X_train[:, i], y_train)[0] for i i
n range(df.shape[1])])
    )
    corrmax_indices = np.argpartition(np.abs(corr[:-1]), -2)[-2:
]

    X_train_selected = X_train[:, corrmax_indices]

    clf = LogisticRegression()
    clf.fit(X_train_selected, y_train)
    mse = mean_squared_error(clf.predict(X_test[:, corrmax_indic
es]), y_test)
    mse_results.append(mse)

mse = pd.Series(mse).mean()
mse
```

```
0.6666666666666666
```

Conclusion

Knowledge leakage is a difficult problem to address completely.
The one thing I recommend doing to avoid this problem is being c
onscientious about using pipelines, like the one scikit-learn pr
ovides, to hande pre-processing and training as one contiguous u

nit (the scikit-learn user guide in fact lists "safety" in this regard as one of the three reasons to use pipelining).

For small to moderately-sized datasets, I do not think that know ledge leakage is a huge problem. Pipelining over feature selecti on has its own problems (it introduces overfitting into cross va lidation?). The amount of error you introduce into your model vi a knowledge leaking is relatively small: maybe even a rounding e rror on your overall model accuracy.

However, it becomes a problem when there are lots of variables, especially when the feature matrix is overdetermined (more varia bles than observations). In these cases you do want to be carefu l about how you design your pre-processing.

When in doubt, I recommend running an exercise like the one I de monstrated here on your dataset. See how much of a difference kn owledge leaking makes for a dataset shaped like yours!