[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, Alexander 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 knowledge with other people and some of us even started to read lectures at the university. So now we have decided to summarize everything and make it available for everyone. Together. We've finished and procesed in about 20 different competitions only on Kaggle and just as many on other not so famous platforms. All of us have a tremendous amount of skill and experience in competitive data science and now we want to share this experience with you. For all of us, competitive data science opened a number of opportunities as the competitions we took part were dedicated to a large variety of tasks. Mikhail works in e-commerce. Alexander builds predictive model for taxi services, Dmitri works with financial 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 comparative record will bring success to you as well. We hope you will find something interesting in this course and wish you good luck.Hello and welcome to our course. In this video, I want to give you 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 assignments and our course schedule. So, at the broadest level, this course is about getting the required knowledge and expertise to successfully 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 they, how they work, how they are different from real-life industrial data analysis. Then, we're moving to recap of main machine learning models. Besides this, we're going to review software and hardware requirements and common Python libraries for data analysis. After this is done, we'll go through various feature types, how we preprocess these features and generate new ones. Now, because we sometimes need to extract features from text and images, 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 set in the exploratory data analysis topic or EDA for short. We'll discuss ways to build intuition about the data, explore anonymized features and clean the data set. Our main instrument here will be logic and visualizations. Okay, now, after making EDA, we switch to validation. And here, we'll spend some time talking about different validation strategies, identifying how data is split 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 problem. We will define data leakage and understand what are leaks, how to discover various leaks and how to utilize them. So basically, this week, we set up the main pipeline for our final project. And at this point, you should have intuition about the data, reliable 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 metrics for regression and classification and figure out ways to optimize them both while training the model and afterwards. After we will check that we are correct in measure and improvements of our models, we'll define mean-encodings and work on the encoded features. So here, we start with categorical features, how mean-encoded features lead to overfitting and how we balance overfitting with regularization. Then, we'll discuss several extensions to this approach including applying mean-encodings to numeric features and time series, and this is the point where we move on to other advanced features in the week four. Basically, this include statistics and distance-based features, metrics factorizations, feature interactions and t-SNE. These features often are the key to superior performance in competition, so you should implement and optimize them here for the final project. After this, we'll get to hyperparameters optimization. Here, we will revise your knowledge about model tuning in a systematic way and let you apply to the competition. Then, we move onto the practical guide where all of us have summarized most important moments about 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 advice, how to set up your pipeline, what to do first and so on. Finally, we'll conclude this week by working on ensembles with Kaz Anova, the Kaggle top one. We'll start with simple linear ensemble, then we continue with bagging and boosting, and finally we'll cover stacking and stacked net approach. And here by the end of this week, you should already have all required knowledge to succeed in a competition. And then finally, we've got the last week. Here we will work to analyze some of our winning solutions in competitions. But all we are really doing in the last week is wrapping up the course, working on and submitting the final project. So, this basic structure of this course. Now, we move through those sections so that you can practice your skills in the course assignments and there are three basic types of assignments in this class: quizzes, programming assignments and the final project. You don't have to do all of these in order to pass the class, you only need to complete the required assignments and you can see which ones those are by looking on the course website. But 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 EDA, exploratory data analysis, set up main pipeline that you'll use for the rest of the course and check the competition for leakages. Then in week three we update our solution by optimizing given metric and adding mean-encoded features. After that, in the week four, we further improve our solution by working on advanced features, tune your hyperparameters and uniting models in ensemble. 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 Coursera 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 about this program and this competition assignment, but we also have quizzes and programming assignments for you. We include these t o give you an opportunity to refine your knowledge about specifi c parts of this course: how to check data for leakages, how to i mplement mean encodings, how to produce an ensemble and so on. Y ou can do them at Coursera site directly but you also can downlo ad these notebooks and complete them at your local computer or y our server. And this basically is an overview of the course goal s, course schedule and course assignments. So, let's go ahead an d get started.Hi everyone. We are starting course about machine learning competitions. In this course, you will learn a lot of t ricks and best practices about data science competitions. Before we start to learn advanced techniques, we need to understand th e basics. In this video, I will explain the main concept of comp etitions and you will become familiar with competition mechanics . A variety of machinery competition is very high. In some, part icipants are asked to process texts. In others, to classify pict ure 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 i n this video. Let's start with a data. Data is what the organize rs give us as training material. We will use it in order to prod uce our solution. Data can be represented in a variety of format s. 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 r ead it in order to understand what we'll work with and which fea ture can be extracted. Here is an example from Kaggle. From the top, we see several files with data, and below, is their descrip tion. Sometimes in addition to data issued by organizers, we can use other data. For example, in order to improve image classifi cation model, one may use a publicly available data set of image s. But this depends on a particular competition and you need to check the rules. The next concept is a model. This is exactly wh at we will build during the competition. It's better to think ab out model not as one specific algorithm, but something that tran sforms data into answers. The model should have two main propert ies. It should produce best possible prediction and be reproduci ble. In fact, it can be very complicated and contain a lot of al gorithms, handcrafted features, use a variety of libraries as th is model of the winners of the Homesite competition shown on thi s slide. It's large and includes many components. But in the cou rse, we will learn how to build such models. To compare our mode l with the model of other participants, we will send our predict ions to the server or in other words, make the submission. Usual ly, you're asked about predictions only. Sources or models are n ot required. And also there are some exceptions, cool competitio ns, where participants submit their code. In this course, we'll focus on traditional challenges where a competitor submit only p rediction outputs. Often, I can not just provide a so-called sam ple submission. An example of how the submission file should loo k like, look at the sample submission from the Zillow competitio n. In it is the first column. We must specify the ID of the obje ct and then specify our prediction for it. This is typical forma t that is used in many competitions. Now, we move to the next co ncept, evaluation function. When you submit predictions, you nee d 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 characterizes the performance of the solution. The simplest example of such a function is the accurate score. This is just a rate of correct answers. In general, there are a lot of such functions. In our course, we will carefully consider some of them. The description of the competition always indicates which evaluation function is used. I strongly suggest you to pay attention to this 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 competitors. So we move to the last point we are considering, the leaderboard. The leaderboard is the rate which provides you with information about performance of all participating teams. Most machine learning competition platforms keep your submission history, but the leaderboard usually shows only your best score and position. They cannot as that submission score, reveal some information about data set. And, in extreme cases, one can obtain ground truth targets after sending a lot of submissions. In order to handle this, the set is divided into two parts, public and private. This split is hidden from users and during the competition, we see the score calculated only on public subset of the data. The second 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 routine looks like that. You as the competition, you analyze the data, 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 scoring. Usually, you are allowed to select two final submissions. Choose wisely. Sometimes public leaderboard scores might be misleading. After the competition deadline, public leaderboard is revealed, and its used for the final rating and defining the winners. That was a brief overview of competition mechanics. Keep in mind 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. Strongly suggest you to read the rules carefully before joining the competition. Now, I want to say a few words about competition platforms. Although Kaggle is the biggest and most famous one, there is a number of smaller platforms or even single-competition sites 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 reason 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 offer you non-trivial problems and state-of-the-art approaches. It allows you to broaden the horizons and look at some everyday task from a different point of view. It's also a great way to become recognizable, get some kind of frame inside data science community and receive a nice job offer. The last reason to participate 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 questio
n 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 ab
out 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 the
m. 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 shar
e code
with other participants or teammates. This is what Kernels are f
or. 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 Not
ebook. It doesn't matter how your
predictions were produced, locally or by Kernel, you should subm
it
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 overvi
ew
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, th
e 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 Progr
ession
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 related 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 column. The reason for that, that in some competitions you need to take care of these things. But usually it's not the case. I want to emphasize that as competitors, the only thing we should take care about is target metrics value. Speed, complexity and memory consumption, all this doesn't matter as long as you're able to calculate it and re-produce your own results. Let's highlight key points. Real world machine learning pipelines are very complicated and consist of many stages. Competitions, add weight to a lot of things about modeling and data analysis, but in general they don't address the questions of formalization, deployment and testing. Now, I want to say a few words about philosophy on competitions, in order to form a right impression. We'll cover these ideas in more details later in the course along with examples. The first thing I want to show you is that, machine learning competitions are not only about algorithms. An algorithm is just a tool. 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, without any sophisticated machine learning techniques. In this course, we will show you the importance of understanding your data, tools 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 yourself. Keep in mind that the only thing you should care about is target metric. It's totally fine to use heuristics or manual data analysis in order to construct golden feature and improve your model. Besides, don't be afraid of using complex solutions, advance feature engineering or doing the huge gritty calculation overnights. Use all the ways you can find in order to improve your model. After passing this course, you will able to get the maximum gain from your data. And now the important aspect is creativity. You need to know traditional approaches of solid machine learning problems but, you shouldn't be bounded by them. It's okay to modify or hack existing algorithm for your particular task. Don't be afraid to read source codes and change them, especially for deploying stuff. In our course, we'll show you examples of 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 getting money. Experience and fun you get are much more valuables than the price. Also, networking is another great advantage of participating in data science competition. I hope you find this course interesting.Hi, everyone. In this video, I want to do a brief overview of basic machine learning approaches and ideas behind them. There are several famous of machine learning algorithms which I want to review. It's a Linear Model, Tree-Based Methods, k-Nearest Neighbors, and Neural Nets. For each of this family, I will give a short intuitive explanation with examples. If you don't remember any of these topics, I strongly encourage you to learn it using links from additional materials. Let's start with Linear Models. Imagine that we have two sets of points, gray points belong to one class and green ones to another. It is very intuitive 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 approach 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 remember several examples from this model class like logistic regression or SVM. They all are Linear Models with different loss functions. I want to emphasize that Linear Models are especially good for sparse high dimensional data. But you should keep in mind the limitations of Linear Models. Often, point cannot be separated by such a simple approach. As an example, you can imagine two sets 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 find implementations of Linear Models in almost every machine learning 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 family, Tree-Based Methods. Tree-Based Methods use decision tree as a basic block for building more complicated models. Let's consider an example of how decision tree works. Imagine that we have two sets of points similar to a linear case. Let's separate one class from the other by a line parallel to the one of the axes. We use such restrictions as it significantly reduces the number of possible lines and allows us to describe the line in a simple way. After setting the split as shown at that picture, we will 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 the 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 approximating data with a constant inside of these boxes. The way of true axis splits and corresponding constants produces several approaches for building decision trees. Moreover, such trees can be combined together in a lot of ways. All this leads to a wide variety of tree-based algorithms, most famous of them being random 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 method 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 of splits. We can imagine two sets of points which can be separated 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 tree could be inaccurate near decision border, as shown on the picture. Similar to Linear Models, you can find implementations of tree-based models in almost every machine learning library. Scikit-Learn contains quite good implementation of random forest which I personally prefer. All the Scikit-Learn contain implementation 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. Next, is that Linear Model can be imagined as splitting space into two sub-spaces separated by a hyper plane. Tree-Based Methods split space into boxes and use constant the predictions in every box. k-NN methods are based on the assumptions that close objects are likely to have same labels. So we need to find closest objects and pick their labels. Also, k-NN approach heavily relies on how to measure point closeness. Feed-forward Neural Nets are harder to interpret but they produce smooth non-linear decision boundary. 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 your attention.Hi, everyone. In this video, I want to do an overview
of hardware and software requirements. You will know what is typical stuff for
data science competitions. I want to start from
hardware related things. Participating in competitions, you generally don't need a lot
of computation resources. A lot of competitions, except imaged based,
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 have
the more or faster experiments you can do. I find it comfortable to
work with fixed cores, but sometimes even 32 are not enough. Next 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 to 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 lot 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 pricing, 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 significantly. 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, or neural networks. Furthermore, one -hot encoding feature is already scaled because minimum this feature is zero, and maximum is one. Note that if you care for a fewer important numeric features, 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 always 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 nutshell, 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 features or text data. Most of the popular libraries can work with these 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 of feature generation is feature interaction between several categorical features. This is usually useful for non tree based models namely, linear model, kNN. For example, let's hypothesize that target depends on both Pclass feature, and sex feature. If this is true, linear model could adjust its predictions for every possible combination of these two features, and get a better result. How can we make this happen? Let's add this interaction by simply concatenating strings from both columns and one-hot encoding 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 advanced features topic. Now, let's summarize this features. First, ordinal is a special case of categorical feature but with values sorted in some meaningful order. Second, label encoding, basically replace this unique values of categorical features with numbers. Third, frequency encoding in this term, maps unique values to their frequencies. Fourth, label encoding and frequency encoding are often used for tree-based methods. Fifth, One-hot encoding is often used for non-tree-based-methods. And finally, applying One-hot encoding combination one heart and chords into combinations of categorical features allows non-tree- based-models to take into consideration interactions between features, and improve. Fine. We just sorted out it feature pre-process for categorical features, and took a quick look on feature generation. Now, you will be able to apply these concepts in your next competition and get better results.Hi. In this video, we will discuss basic
visual generation approaches for datetime and coordinate features. They both differ significantly from
numeric and categorical features. Because we can interpret the meaning of
datetime and coordinates, we can came up with specific ideas about future
generation which we'll discuss here. Now, let's start with datetime. 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 coo rdinates 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, a nd engine
differences between two datetime features. For coordinates, we s hould recall
extracting interesting samples from trained test data, using pla ces from
additional data, calculating distances to centers of clusters, a nd adding aggregated
statistics for surrounding area. Knowing how to effectively hand le datetime
and coordinates, as well as numeric and categorical features, wi ll 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 t op 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 co ntain useful
information by themselves, like what was the reason of
missing value occurring here? How to use them effectively? How t o engineer new features from them? We'll do the topic for this v ideo. 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 f eatures. 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 fo und 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 v alues. So if there are no not numbers there, we
can assume that they were replaced by -1. Or the feature distrib ution 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 lear ned this,
missing values can be hidden from us. And by hidden I mean repla ced by some
other value beside not a number. Great, let's talk about
missing value importation. The most often examples are first, re placing 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 perfo rmance
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 mom ent. 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 tr ees 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 va lues
importation methods. The third one, and the last one we will dis cuss here,
is to reconstruct each value if possible. One example of such po ssibility is
having missing values in time series. For example,
we could have everyday temperature for a month but several value s in
the middle of months are missing. Well of course, we can approxi mate
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 an y
proper logic to reconstruct them. Great, to this moment we alrea dy learned
that we can construct new feature, isnull indicating which
rows contains not numbers. What are other important moments abou t
feature generation we should know? Well there's one general conc ern 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. N
ow we want to add feature like
difference between temperature today and yesterday, let's do thi
s. 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 e
rror by points,
great. But unfortunately, we usually don't
have enough time to be so careful here. And more importantly, th
ese 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 h
ave a data set
with independent rows. And we want to encode the categorical
feature with the numeric feature. To achieve that we calculate m
ean
value of numeric feature for every category, and
replace categories with these mean values. What happens if we fi
ll 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 correspondi
ng to
particular category will have missing values. The closer mean va
lue 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 i
mportation
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 ide
a
of these two examples. You should be very careful with early non
e
importation if you want to generate new features. There's one mo
re 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, so
metimes 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 t
rain 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 Wikipe
dia. Otherwise, remember, the training of Word2vec doesn't requi
re target values from your text. It only requires text to extrac
t context for each word. Note, that all pre-processing we had di
scussed earlier, namely lowercase stemming, lemmatization, and t
he usage of stopwords can be applied to text before training Wor
d2vec models. Now, we're ready to summarize difference between B
ag of Words and the Word2vec approaches in the context of compet
ition. With Bag of Words, vectors are quite large but is a nice
benefit. Meaning of each value in the vector is known. With Word
2vec, vectors have relatively small length but values in a vecto
r can be interpreted only in some cases, which sometimes can be
seen as a downside. The other advantage of Word2vec is crucial i
n competitions, is that words with similar meaning will have sim
ilar vector representations. Usually, both Bag of Words and Word
2vec approaches give quite different results and can be used tog
ether in your solution. Let's proceed to images now. Similar to
Word2vec for words, convolutional neural networks can give us co
mpressed representation for an image. Let me provide you a quick
explanation. When we calculate network output for the image, be
side getting output on the last layer, we also have outputs from
inner layers. Here, we will call these outputs descriptors. Des
criptors from later layers are better way to solve texts similar
to one network was trained on. In contrary, descriptors from ea
rly layers have more text independent information. For example,
if your network was trained on images and data set, you may succ
essfully use its last layer representation in some Kar model cla
ssification text. But if you want to use your network in some me
dical specific text, you probably will do better if you will use
an earlier for connected layer or even retrain network from scr
atch. Here, you may look for a pre-trained model which was train
ed on data similar to what you have in the exact competition. So
metimes, we can slightly tune network to receive more suitable r
epresentations using targets values associated with our images.
In general, process of pre-trained model tuning is called fine-t
uning. As in the previous example, when we are solving some medi
cal 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 netw
ork 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 para
meters and thus extract more effective image representations. On
the other hand, fine-tuning is better than training network fro
m scratch if we have too little data, or if the text we are solv
ing is similar to the text model was trained on. In this case, m
odel can you use the my knowledge already encoded in networks pa
rameters, which can lead to better results and the faster retrai
ning procedure. Lets discuss the most often scenario of using th
e 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 m
etric. 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.