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

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

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

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

features based on nearest neighbors. Back then, we took the thir
d place in
this competition together with and. And now in this video, I wil
l describe
the last part of our solution, which is the usage of stacking an
d
ensembles. On this picture, you can see the final
stacking scheme we produced on the level 0 features, on the firs
t level,
predictions by basic models. On the level one plus combination.
So these predictions and
some accurately chosen features on the second level models
on this new set of features. And finally, on the third level,
their linear combination. In this video, we will go through each
 level as it builds
up to this non-trivial ensembled scheme. But first, let's quickl
y remind
ourselves about the problem. This was a binary classification
task with area under curve metric. We had 145,000 samples in tra
ining data
and about 2,000 anonymized features. These were useful insights
derived by us while doing EDA. And you can check out EDA done by
 earlier
in our course to refresh your memory. So now let's start with fe
atures. Here we have four packs of features. First two are the b
asic dataset and
the processed dataset. To keep it simple,
we just used insights derived from EDA to clean data [INAUDIBLE]
 and
to generate new features. For example,
we remove duplicated features and edit some feature interaction
based
on scatter plots and correlations. Then, we mean-encoded all cat
egorical
features using growth relation loop and sign data and smoothing.
 We further used the mean-encoded dataset
to calculate features based on nearest neighbors. Like, what is
the least in
closest object of the class zero? And how many objects out of te
n
nearest neighbors belong to class one? You can review how this c
ould be done in related topics introduced
by Dmitri Altihof. So finally, these four packs of
feature were level 0 of our solution. And the second level was r
epresented
by several different gradient within decision tree models,
and one neural network. The main idea here is that meta
features should be diverse. Each meta feature should bring
new information about the target. So we use both distinct parame
ters and
different sets of features for our models. For the neural networ
k, we additionally
pre-processed features with common scalars, ranks and
power transformation. The problem there was in huge outliers
which skew network training results. So ranks and power transfor
mation

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

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

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

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

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

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

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

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

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

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

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

room to try all these things. Try all different permutations and
 combinations of data techniques, anything you have you can put
in mind, and really understand what works and what's not. So, we
 tried to do that. The first of attempt didn't go very well. So
we try to randomly split the data between train and validation a
nd we're trying to make certain that each offer is represented e
qually in each one of this train and validation data set proport
ionately equally. But was that correct? I mean, if you think abo
ut it. What we were doing there, we were saying I'm building a m
odel with some offers and I'm validating in the same offers. Tha
t's good. Maybe we can do well here. But is this what we're real
ly being tested on? No. Because in the test data, we'll have com
pletely different offers. So, this didn't work very well. This w
as giving very nice internal results but not very good results i
n the test data. So, we tried something else. Can we leave one o
ffer out? And I'm showing you roughly what this look like. So, f
or every offer, can we put an offer in the validation data and u
se all the cases of every other offer to train a model? So, if w
e were to predict offer 16, we will use all customers that recei
ved offer 1 to 15 and 17 to 24 to build the model and then we'll
 make predictions for all those customers that received offer 16
. And you can see that this actually is quite close to what you'
re being tested on because you know you're building a model with
 some offers but, you're being tested on some other offer that i
s not there. And you can take the average of all these 24 users
and I put 24 because this is how many offers you really have in
the training data. You can take that average and that average ma
y be much more close to the reality, close to what you were bein
g tested on. And this was true. This gave better results, but we
 were still not there. And I'll show you why we were not there.
Consider the following problem. Here, I'll give you a small samp
le of predictions for offer two and what was the actual target?
What we see here is a perfect AUC score. Why? Because all our po
sitive cases that are labeled with one and they have the green c
olor, have higher score than the red ones, where the target is z
ero. So, the discrimination here is perfect. We have a point, a
cutoff point. We can set 0.5 here where all cases that have scor
e higher than this. We can safely say they are one and that is t
rue and everything that has a score lower than this are zero. So
, you see here one discrimination is perfect. Let's now take a s
ample from offer four. If you remember offer four, had in genera
l lower propensity. Offer two had around 0.5 and offer four had
around 0.2. So, it's mean we're center much lower and what you c
an see here is that, again, AUC is perfect for this sample becau
se again, all the higher scores that are labeled with green have
 a target of one. And then the lower scores, everything that has
 a score less than 0.18 has a target of 0. The discrimination is
 perfect. We can find this cutoff point. We can say 0.8, where e
verything that has a score higher than this can safely be set to
 one. And that is always true. And vice versa, everything that's
 less than 18, then it's a 0. And that is always true. So, we ha
ve two scores. They discriminate really well between the good an
d the bad cases. However, we are not tested on the AUC of one of
fer. We are tested on the AUC of all offers together. So the tes
t data have many offers. So, you are interested in the score tha
t generalizes well against any offer. So, what happens if we try

to merge this table? AUC is no longer perfect and why this happ
ens? Because some of the negative cases of the first offer had h
igher score than the positive cases, those that have a target eq
ual to one from the second offer. So you can see, although the d
iscrimination internally is really good, they don't blend that w
ell. You lose something from that ability of your score to discr
iminate between ones and zeros. And the moment we saw this, we k
new that just leaving one offer out was not going to be enough.
We had to make certain that when we merge all those scores toget
her, the score is still good. The ability of our model to discri
minate is still powerful or it doesn't lose. And that's why we u
se a combination of the previous average AUC of all the offers a
nd the AUC after doing this concatenation. So, the average of th
e two AUCs which really the metric we try to optimize because we
 thought that this is actually very close to what we were being
tested on. And here I can show you the result of all our attempt
s and this is with a small subset of features because by that po
int, we were not interested to create the best features, we were
 interested to test which approach works best. So, you can see i
f you do it standard stratified K-fold, you can get much nicer r
esults in internal cross-validation but in the test data, the re
lationship is almost opposite. So, highest score in cross-valida
tion leads to worse results in the test data. And you can see wh
y because you're not internally modeling or internally validatin
g or on what you are actually being tested on. Doing the one-off
er out keep obviously lower internal cross-validation results an
d better performance in the test data, but even better was doing
 this leave-one-offer plus one concatenation in the end. And thi
s AUC was lower but actually had better test results. I believe
we could get even better results if we made certain that we are
also validating in new customers. But we didn't actually do this
 because we saw that this approach had already good results. But
 as a means to improve, we could have also made certains that we
 validate on different customers because this is what the test w
as like.