**EDA**

**validation.**

We will see that the validation design depends on the competition setup and that the correct validation scheme is one of the essential bricks for any winning solution.

**GETTING DOMAIN KNOWLEDGE**

If you find mistakes in data. Maybe you can use this as another feature like boolean “incorrect row” and train the model. Which can also lead to better accuracy.

**Some good code snippet:**

print 'Test min/max date: %s / %s' % ( test.Date.min().date(), test.Date.max().date())

print 'Number of days in train: %d' % ((train.Date.max() - train.Date.min()).days + 1)

import itertools

# This function looks for a combination of elements

# with product of 639360

def find\_prod(data):

# combinations of not more than 5 features

for n in range(1, 5):

# iterate through all combinations

for c in itertools.combinations(range(len(data)), n):

if data[list(c)].prod() == 639360:

print test\_nunique.index[c]

return

print 'Nothing found'

find\_prod(test\_nunique.values)

To find a combination of n features whose product of unique elements can be equal to 639360.

# from absolute dates to relative

train['date\_diff'] = (train.Date - train.Date.min()).dt.days

df.size() on group by can give a nice list of group sizes to use maybe in things like scatter plots nd all.

**pandas.factorize()** method helps to get the numeric representation of an array by identifying distinct values. This method is available as both pandas.factorize() and Series.factorize().

In random forest we can get feature importances of the fit tree as well and can directly give it for plotting

plt.plot(rf.feature.importances\_)

**Anonymized data:**

When companies dont want to show data in the exact from and replace it with lets say the hash or some other form to give it to competitors.

Try to know the type of the feature.

You can manually see and guess but if the number of features is too many we have

Df.dtypes:

**Float**: numerical,

**int**: binary or categorical,

**object**: can be anything even a numeric feature with text values can be this.

df.info()  
x.value\_counts()

x.isnull()

**TODO: OFFICE ADD**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**VISUALIZATION:**

**Of a single feature first:**

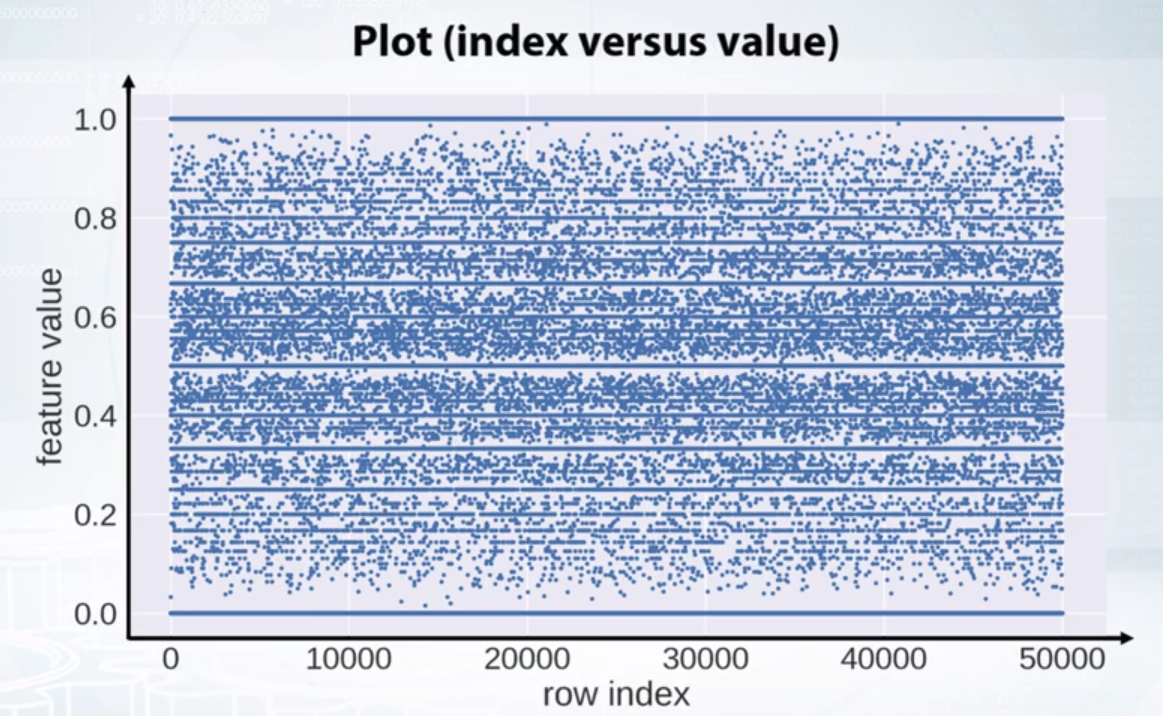
Histograms:

plt.hist(x)

Histograms can be misleading. Try to build several plots around your hypothesis. Make sure

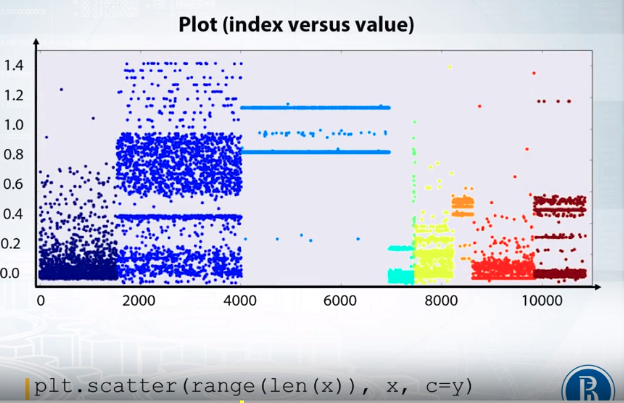
The plot is scaled to ensure correctness.

Things like a peak close to normal distribution center means that the missing values were replaced by the mean. So now you can again make this NA or maybe build another new column feature to it or maybe give it to XGBoost which take cares of nans on its own or maybe provide your own set of values for missing data.



plt.plot(x, ‘.’)

We can also color code the points according to their labels



**FEATURE STATS:**

**Pandas**

**df.describe()**

**x.mean()  
x.var()**

**x.value\_counts()**

**x.isnull()**

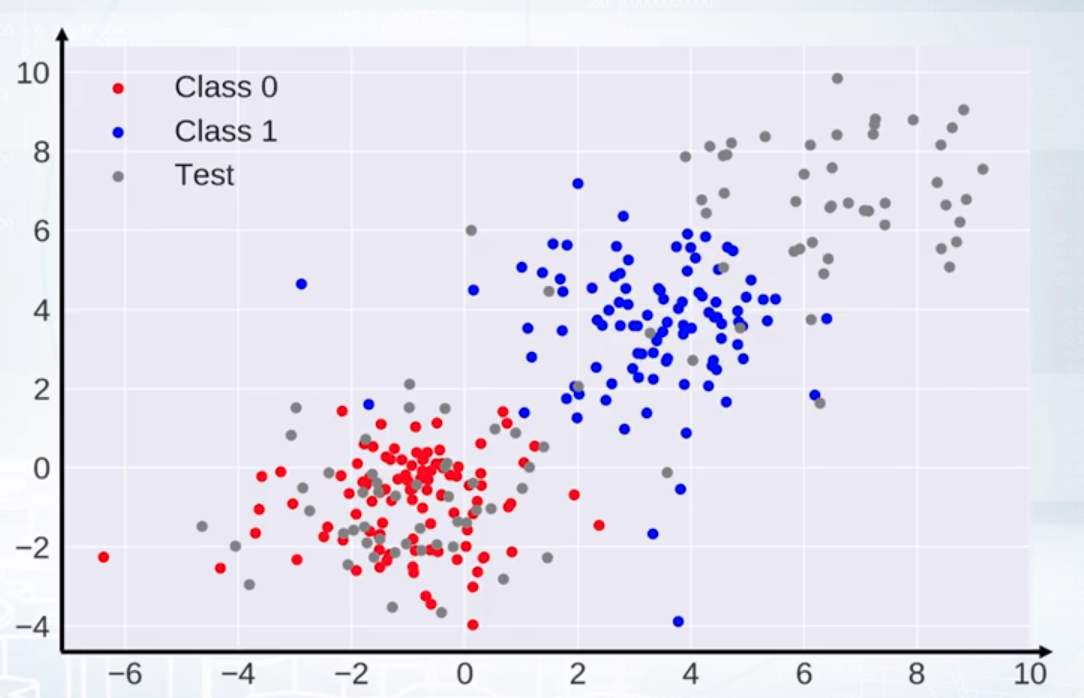
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Exploring feature relations

Scatter plot

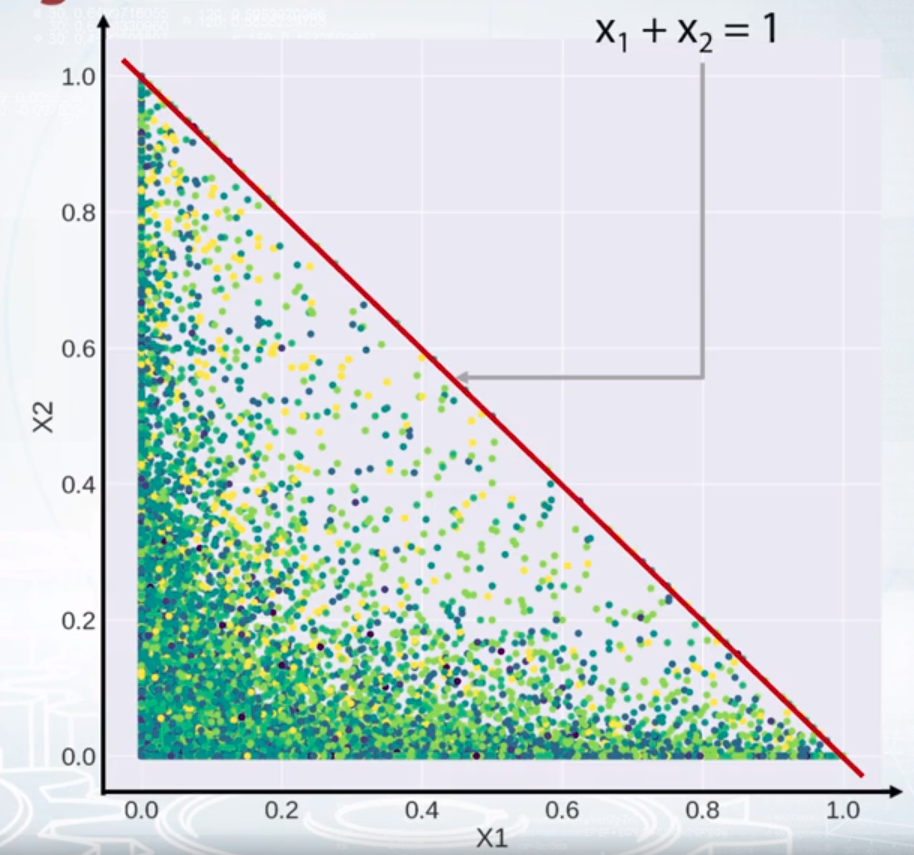
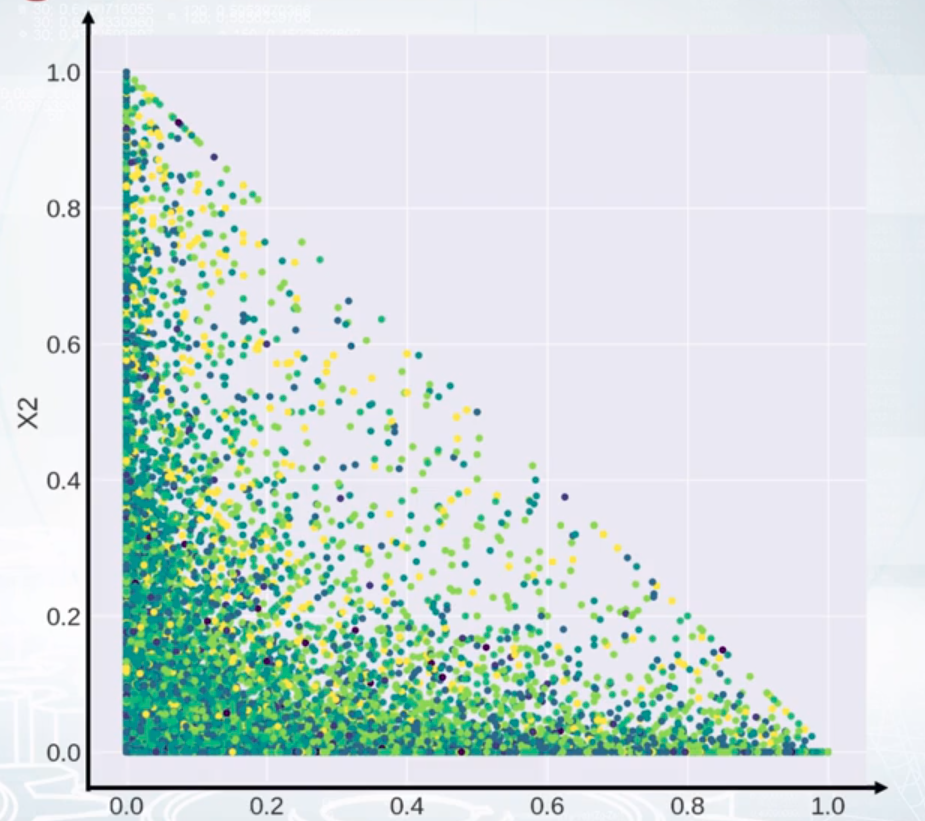
plt.scatter(p1, p2)

Also make sure the type of data distribution in the train and test set is the same using these plots and colour coding accordingly.



Like here for the test set and train set (class zero and one), we can see that the train is not equally spread like the test and can lead to bad predictions.

Have a close examination. Like here the relation is x1 + x2 < 1 as the points are left to the line



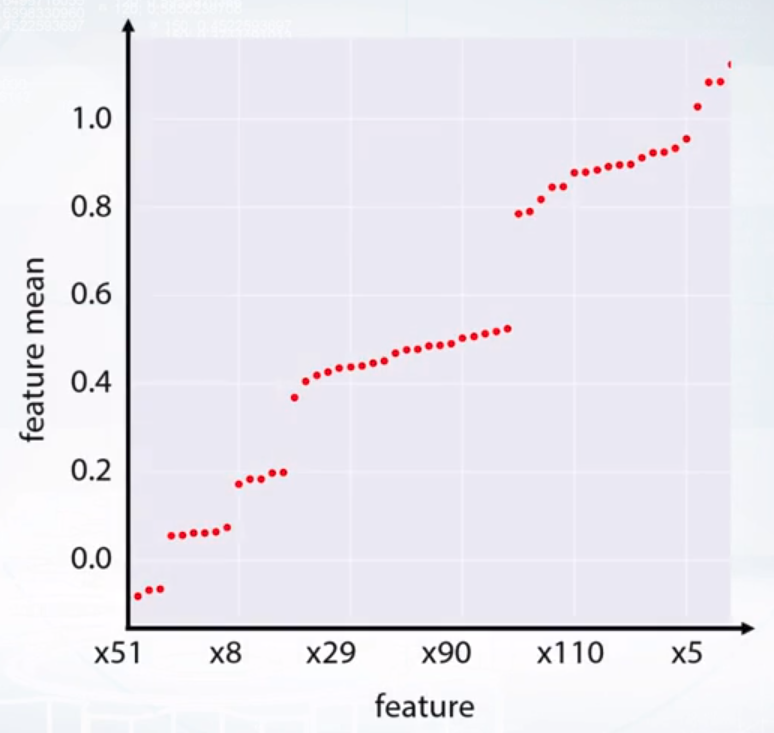
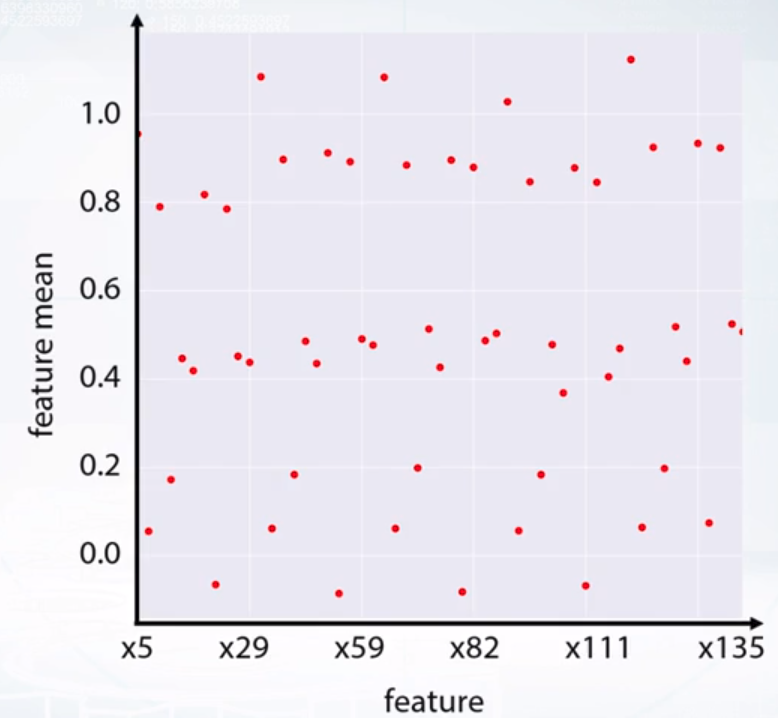
To plot all the feature plots with each other

U can seaborn or pandas

pd.scatter\_matrix(df)

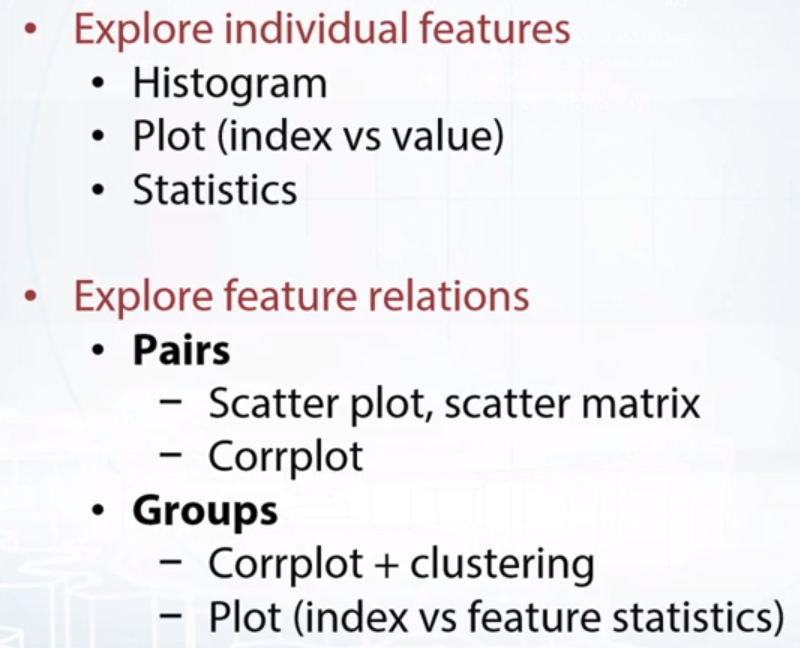
feature vs mean plot

df.mean().plot(style=’.’) modified to df.mean().sort\_values().plot(style=’.’)



Now we can take a closer look to each group and then draw some statistics from it.

We can also use correlation plots to find relations of how similar the columns are and if we reorder the columns and rows of the matrix we can find some good feature groups as well



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DATA SET CLEANING AND OTHER THINGS TO CHECK**

DUPLICATED AND CONST FEATURES

df.nunique(axis=1) == 1 remove the constant feature. As it'll have weight values later on to the model and of no use and will be bad for the model.

Drop one of the two duplicate columns as well. df.T.drop\_duplicates()

We can also have duplicate columns with different labels and doesn't look different. For this df.factorize() both the columns and then use df.T.drop\_duplicates()

THE label encoding function starts to give labels as they arrive. So if a column starts with A it gets 1 and if other column starts with C it gets the label as 1 and so both will look different even being the same column. So make sure you do label encodings carefully.

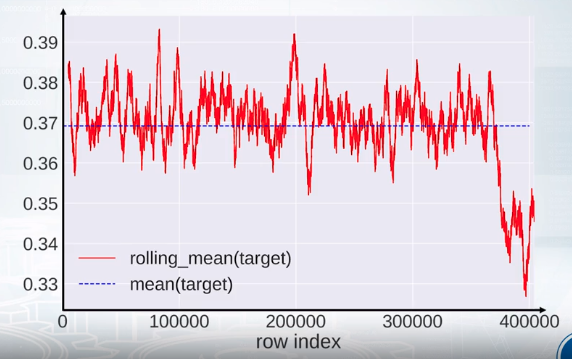
See if rows are duplicated but their targets are different. Take the decision as to remove it or it gives some other important understanding.

It is very useful to check that the dataset is shuffled and if it's not there is a high chance of data leakage.

The index vs feature value plot can help.

Having rolling mean and mean values plotted of the target.

Below as we can see there is some different behaviour in the end dataset and should be made sure to do a reshuffling



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Visualization tools

* [Seaborn](https://seaborn.pydata.org/)
* [Plotly](https://plot.ly/python/)
* [Bokeh](https://github.com/bokeh/bokeh)
* [ggplot](http://ggplot.yhathq.com/)
* [Graph visualization with NetworkX](https://networkx.github.io/)

## Others

* [Biclustering algorithms for sorting corrplots](http://scikit-learn.org/stable/auto_examples/bicluster/plot_spectral_biclustering.html)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**EDA EXAMPLES**

**Spring leaf challenge. Dealing with 2000 feature columns**

Number of nulls in rows

train.isnull().sum(axis=1).head(15)

rows they look like integer-typed too, since fractional part is zero, but pandas treats them as float since there are NaN values in that features.

### **Remove constant features[¶](https://mrswzenwnufdumipqzjcjz.coursera-apps.org/notebooks/readonly/reading_materials/EDA_Springleaf_screencast.ipynb#Remove-constant-features)**

It is usually convenient to concatenate train and test into one dataframe and do all feature engineering using it.

traintest = pd.concat([train, test], axis = 0)

First we schould look for constant features, such features do not provide any information and only make our dataset larger.

# `dropna = False` makes nunique treat NaNs as a distinct value

feats\_counts = train.nunique(dropna = False)

Then print the unique number and find the index till 1 nunique

feats\_counts.sort\_values()[:10]

### **Remove duplicated features[¶](https://mrswzenwnufdumipqzjcjz.coursera-apps.org/notebooks/readonly/reading_materials/EDA_Springleaf_screencast.ipynb#Remove-duplicated-features)**

Fill NaNs with something we can find later if needed.

In [19]:

traintest.fillna('NaN', inplace=**True**)

Now let's encode each feature, as we discussed.

In [32]:

train\_enc = pd.DataFrame(index = train.index)

​

**for** col **in** tqdm\_notebook(traintest.columns):

train\_enc[col] = train[col].factorize()[0]

We could also do something like this:

In [33]:

*# train\_enc[col] = train[col].map(train[col].value\_counts())*

The resulting data frame is very very large, so we cannot just transpose it and use .duplicated. That is why we will use a simple loop.

In [34]:

dup\_cols = {}

​

**for** i, c1 **in** enumerate(tqdm\_notebook(train\_enc.columns)):

**for** c2 **in** train\_enc.columns[i **+** 1:]:

**if** c2 **not** **in** dup\_cols **and** np.all(train\_enc[c1] == train\_enc[c2]):

dup\_cols[c2] = c1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import cPickle as pickle

pickle.dump(dup\_cols, open('dup\_cols.p', 'w'), protocol=pickle.HIGHEST\_PROTOCOL)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

nunique = train.nunique(dropna=False)

and build a histogram of those values

In [44]:

plt.figure(figsize=(14,6))

\_ = plt.hist(nunique.astype(float)**/**train.shape[0], bins=100)

See the groups in these histograms and try to figure out the most and the least unique columns and see if you can find some pattern around it. For eg.

The values are not float, they are integer, so these features are likely to be even counts. Let's look at another pack of features.

In [64]:

mask = (nunique.astype(float)**/**train.shape[0] **<** 0.8) **&** (nunique.astype(float)**/**train.shape[0] **>** 0.4)

train.loc[:25, mask]

After the hist plot we can see columns which looks similar to each other and so we should make some new features based on this. Like if all values are the same give 1 or

Else give 0. Another categorical encoding of the number of similar columns.

We can also get to see the dataset representation of NaNs like 9999999.

First thing to notice is the 23th line: 99999.., -99999 values look like NaNs so we should probably built a related feature. Second: the columns are sometimes placed next to each other, so the columns are probably grouped together and we can disentangle that.

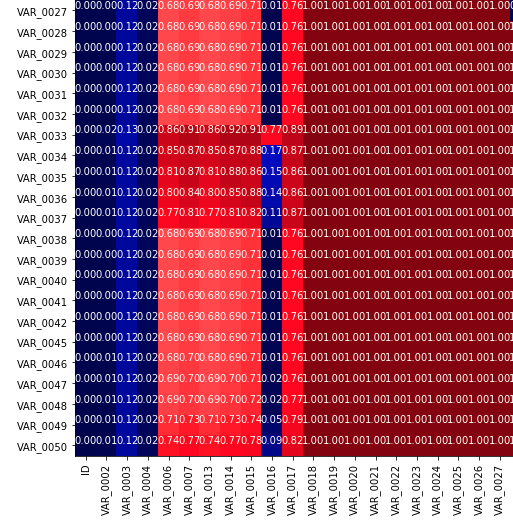
**Assuming categories have dtypes object. Can be misleading at times too. U can change this using astypes() also**

cat\_cols = list(train.select\_dtypes(include=['object']).columns)

num\_cols = list(train.select\_dtypes(exclude=['object']).columns)

YOU CAN BUILD MEAN PLOT:

# build 'mean(feat1 > feat2)' plot



So value greator than 0.5 means that majority of values came out to be greator than the other column and so the mean was bigger. As the > than symbol works like taking 1 for bigger values and 0 for smaller. Mean is the (number of greators in the column)/ (total column)

The figure above is just a partial clipping of the whole matrix

def gt\_matrix(feats,sz=16):

a = []

for i,c1 in enumerate(feats):

b = []

for j,c2 in enumerate(feats):

mask = (~train[c1].isnull()) & (~train[c2].isnull())

if i>=j:

b.append((train.loc[mask,c1].values>=train.loc[mask,c2].values).mean())

else:

b.append((train.loc[mask,c1].values>train.loc[mask,c2].values).mean())

a.append(b)

plt.figure(figsize = (sz,sz))

plt.imshow(a, interpolation = 'None')

\_ = plt.xticks(range(len(feats)),feats,rotation = 90)

\_ = plt.yticks(range(len(feats)),feats,rotation = 0)

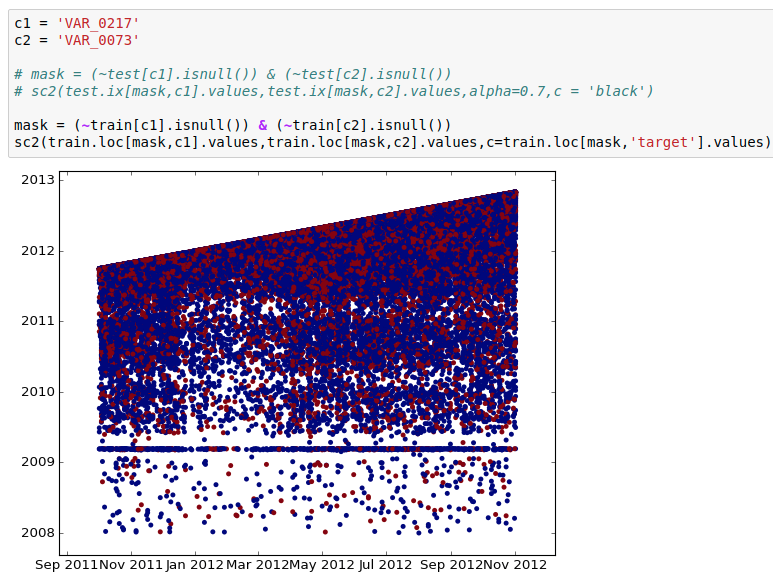
autolabel(a)

Indeed, we see interesting patterns here. There are blocks of geatures where one is strictly greater than the other. So we can hypothesize, that each column correspondes to cumulative counts, e.g. feature number one is counts in first month, second -- total count number in first two month and so on. So we immediately understand what features we should generate to make tree-based models more efficient: the differences between consecutive values.

U can see a group of features in mean matrix plot and can create new features from it for example difference between two columns.

Hist plots can show spikes. LIke the one showed in notebook. These spikes were at 12 24 36 60 and so they look like time values and so we can divide or modulo by 12 and as we can see other data in histogram they can be noise added to make things more challenging where as the column itself was of time only. So adding new features can really help.

We can scatter plot two date features



We see that one date is strictly greater than the other, so the difference between them can be a good feature. Also look at horizontal line there -- it also looks like NaN, so I would rather create a new binary feature which will serve as an idicator that our time feature is NaN.

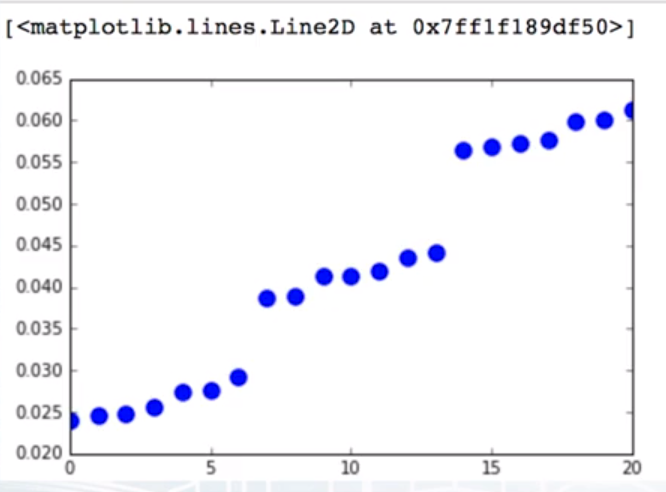
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

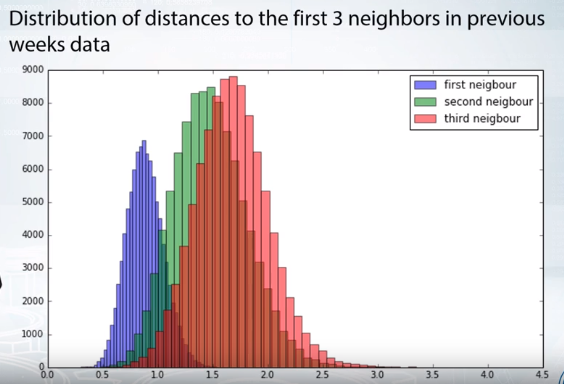
NumerAI figuring out that the updated data use to be some sort of timestamp and that 21 features, when added with each point adding its 21 k nearest neighbours into it, gave a leader board rank of top 10.



correlation matrix sorted based on the highest correlation coefficient.

You can create new features of highly correlated data. This will also help with competitions changing dataset every week with random shuffling and so that shuffling will not affect us.

here as you can see we get 3 groups of features. Where this is the plot of some highly correlated data taken from the correlation matrix. Due to these 3 groups. Multiplying group 1 by 3 and group 2 by 2 so that all come at the same level and then again computing things and predicting improved the results.



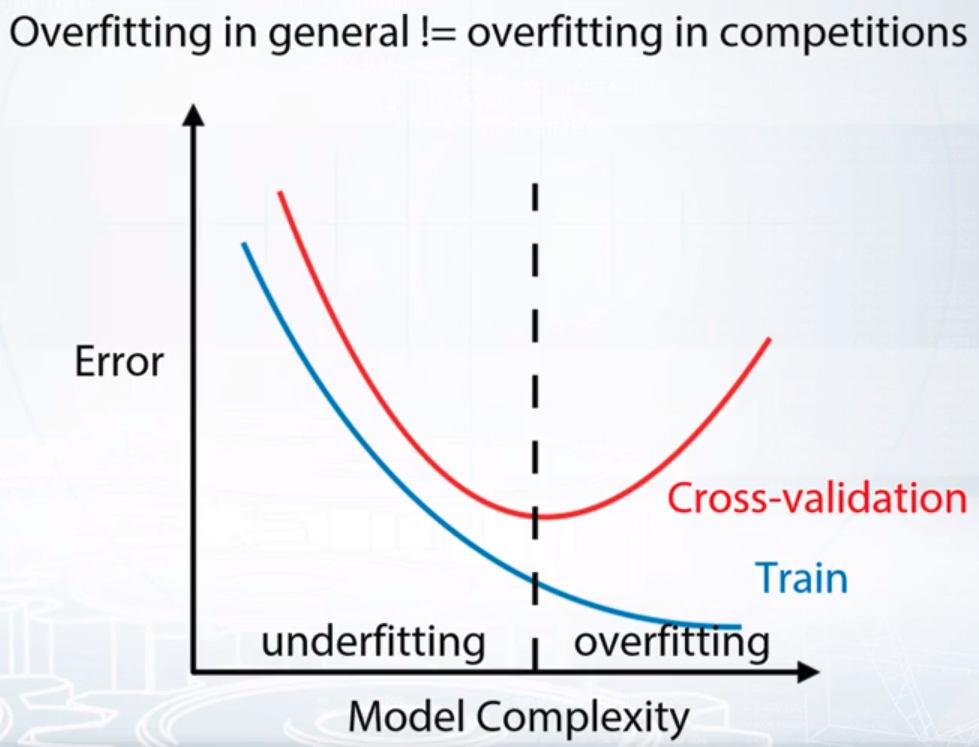
The 3 datasets looks slightly equal but shifted a lil bit and we can look to create a mapping between them.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**VALIDATION**

Without a good validation strategy we can overfit the model on the public leaderboard and later on end up with bad score on the private learderboard.

So well have to select the best model that lies between under and overfitting which can generalize the model well.



**VALIDATION STRATEGIES**

1. HOLD OUT
2. K FOLD OUT
3. LEAVE ONE OUT

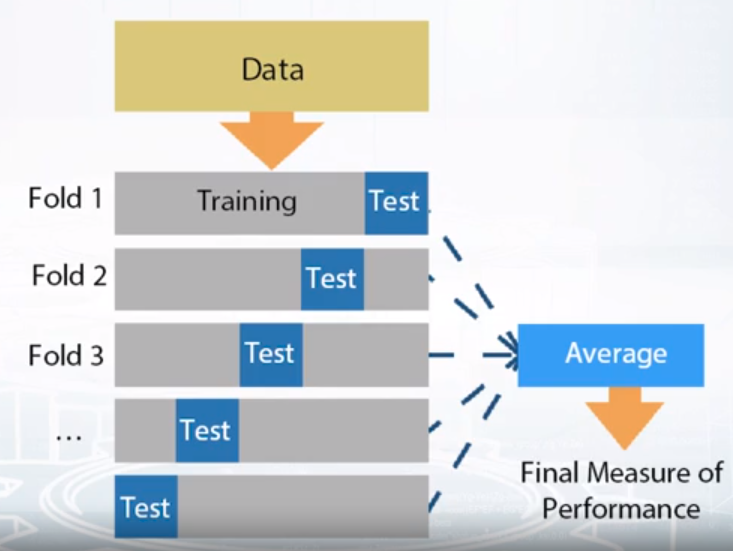
**Hold out:**

Divides dataset into train and test.

Samples in train and test must not overlap or be duplicated or else the model will fit better in the test deceiving us that the model is performing well. Whereas it's just overfitting on the train.

Only to be done when we have a lot of data. Because test data will go wasted and could have helped training the model.

**KFOLD**

 this helps as all the data is used in training.

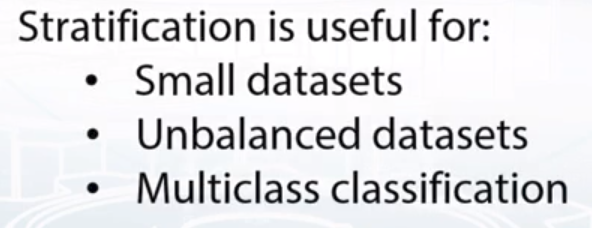
K fold: sklearn.model\_selection.Kfold

Hold out : when k fold has one 1 fold: sklearn.model\_selection.ShuffleSplit

**Leave one out:** when k fold is equal to number of samples in the data: .LeaveOneOut. When we have too little data. We should use this.

CLASS IMBALANCED data in a random split can fail by not taking the few sample datasets and dividing it correctly. So to solve this we have :

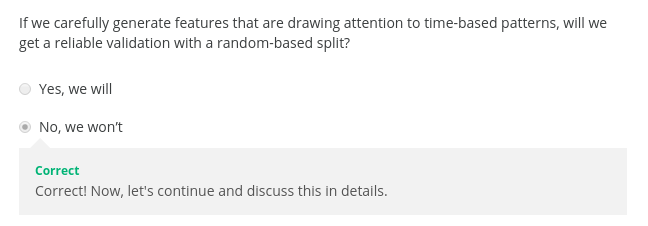
**STRATIFICATION** which ensures similar target distribution over different folds. A average of each folds target value is constant.

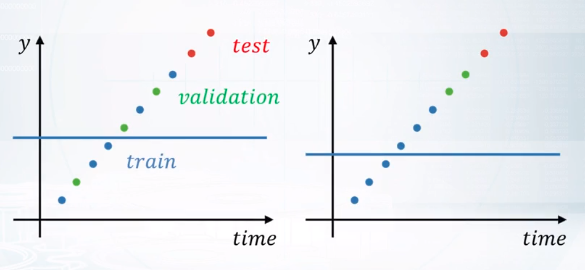


**Notice, that these are *validation* schemes are supposed to be used to estimate the quality of the model. When you found the right hyper-parameters and want to get test predictions don't forget to retrain your model using all training data.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-**

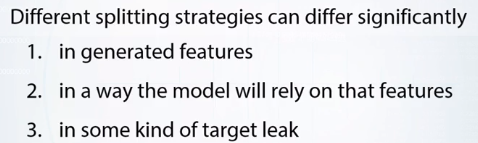
**DATA SPLITTING STRATEGY**





In the above image one is random and other is time based split. 1st focuses on predicting values based on the the exact datapoint without much knowledge of past and future points and tries to find out connection based on that.

The other model learns to take into consideration the split and tries to predict future values most of the time as the validation values were higher than the training values.



We can’t say one is better than the other,

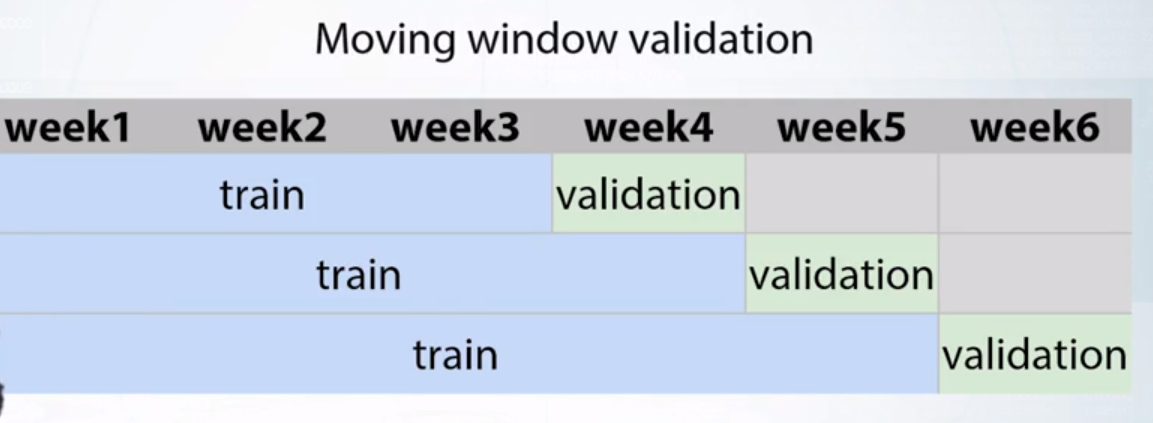
That means, to be able to find smart ideas for feature generation and to consistently improve our model, we absolutely want to identify train/test split made by organizers, including the competition, and reproduce it.

Split categories:

1. **Random, rowwise:** means rows are independent of each other .

2. **Time wise:** based on a given time stamp. Before goes to training and after goes to test. This can be used to check the performance of models that rely a lot on predicting future time series data. For example next months sale.

A new type of validation can also be used here called the moving window validation.



3 **by id:** id can be unique identifier of a user, shop or any other entity. See if train and test overlap in terms of same id so that we can use it to create new features. Like if we have the same user in train and test we can recommend songs by using train data on the test data. Mostly these overlaps are hidden from the participants. Another id example can be fishing boat pictures in train and test and based on the similarity of pics taken on the same boat due to hardly any picture dis-similarities, knn was used to group similar pics together and identify the same boat uniquely carrying a type of fish. When the private set came out , it is like finding which boat it was to predict the fish it’ll have. And in this way they derived the ids which in this case is unique boats.

4) **COMBINED:** split in time shop wise, rather than choosing one data for all the shops in the dataset.

Or another example, if we have search queries from multiple users, is using several search engines, we can split the data by a combination of user ID and search engine ID. Examples of competitions with combined splits include the Western Australia Rental Prices competition by Deloitte and their qualification phase of data science game 2017. In the first competition,train/test was split by a single date, but the public/private split was made by different dates for different geographic areas. In the second competition, participants had to predict whether a user of online music service will listen to the song. The train/test split was made in the following way.For each user, the last song he listened to was placed in the test set, while all other songs were placed in the train set. Fine. These were the main splitting strategies employed in the competitions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-

**VALIDATION STRATEGY MATTERS A LOT**

Again, the main idea I want you to take away from this lesson is that your validation should always mimic train/test split made by organizers. It could be something non-trivial.

For example, in the Home Depot Product Search Relevance competition, participants were asked to estimate search relevancy. In general, data consisted of search terms and search results for those terms, but test set contained completely new search terms. So, we couldn't use either a random split or a search term-based split for validation. First split favored more complicated models,which led to overfitting while second split, conversely, to underfitting. So, in order to select optimal models, it was crucial to mimic the ratio of new search terms from train/test split.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-

SO A GOOD VALIDATION STRATEGY WILL FOR SURE TAKE US HIGH ON THE FINAL PRIVATE LEADERBOARD AND IT MAY TAKE US DOWN IN PUBLIC LEADERBOARD.

Validation problems:

1. **Validation stage:** local validation caused due to inconsistency of data.
2. **Submission stage:** seeing that the score on the leaderboards is going down with improvement in local validation.

**VALIDATION STAGE**

If you see different optimal parameter and scores for different K-folds in the data, chances are your validation way is wrong. Other reasons can be

1. **Too little data:** if we have too much patterns in the data but too less data for the model to learn the pattern, then the model will learn some partial patterns and also it will differ in terms of learned pattern in different K-folds giving different scores as well. Also the data is too small and most of the time the new batch is completely different in terms of validation due to its diversity owing to more less score.
2. **Too diverse and inconsistent data**: For example, if you have very similar samples with different target variance, a model can confuse them. Consider two cases, first, if one of such examples is in the train while another is in the validation. We can get a pretty high error for the second sample.

**SOLUTION:**

If we are facing this kind of problem, it can be useful to make more thorough validation. You can increase K in KFold, but usually 5 folds are enough. Make KFold validation several times with different random splits.And average scores to get a more stable estimate of model's quality.

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

**Submission stage:**

Two types are:

1. leaderboard score is consistently higher/lower than validation score

2. LB score is not correlated with validation score at all.

**Reasons can be:**

too little data in public leaderboard

Train and test data are from different distributions.

**GOLDEN RULE**

Now remember that the main rule of making a reliable validation, is to mimic a train tests pre made by organizers. Because of that, I highly you to start submitting your solutions right after you enter the competition.

There could be two more reasons for this problem. The first reason, we have too little data in public leaderboard, which is pretty self explanatory. Just trust your validation, and everything will be fine. And the second train and test data are from different distributions. Let me explain what I mean when I talk about different distributions. Consider a regression test of predicting people's height by their photos on Instagram. The blue line represents the distribution of heights for man,while the red line represents the distribution of heights for women. As you can see, these distributions are different. Now let's consider that the train data consists only of women, while the test data consists only of men. Then all model predictions will be around the average height for women. And the distribution of these predictions will be very similar to that for the train data. No wonder that our model will have a terrible score on the test data. Now, because our course is a practical one, let's take a moment and think what you can do if you encounter these in a competition. Okay, let's start with a general approach to such problems. At the broadest level, we need to find a way to tackle different distributions in train and test.

Sometimes, these kind of problems could be solved by adjusting your solution during the training procedure. But sometimes, this problem can be solved only by adjusting your solution through the leaderboard.That is through leaderboard probing. The simplest way to solve this particular situation in a competition is to try to figure out the optimal constant prediction for train and test data. And shift your predictions by the difference. Right here we can calculate the average height of women from the train data. Calculating the average height of men is a bit trickier. If the competition's metric is means squared error, we can send two constant submissions, write down the simple formula. And find out that the average target value for the test is equal to 7 inches. In general, this technique is known as leaderboard probing. And we will discuss it in the topic about. So now we know the difference between the average target values for the train and the test data, which is equal to 7 inches. And as the third step of adjusting our submission to the leaderboard we could just try to add 7 to all predictions. But from this point it is not validational it is a leaderboard probing and leaks.Yes we probably could discover this during exploratory data analysis and try to make a correction in our validation scheme. But sometimes it is not possible without leaderboard probing, just like in this example. BUT this is rare enough. What happens more is:

Class imbalance: where the train has more men data than women but the public leader board has vice versa. In this case force the validation to have the same distribution as the leader board one.

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

Ensuring the same distribution in test and validation helps us get scores and parameters relevant to test.

And second, competition with CTR prediction which we discussed earlier in the data topic. Let's start with the second one, do you remember the problem, we have a test of predicting CTR. So, the train data, which basically was the history of displayed ads obviously didn't contain ads which were not shown. On the contrary, the test data consisted of every possible ad. Notice this is the exact case of different distributions in train and test.

And again, we need to set up our validation to mimic test here. So we have this huge bias towards showing that in the train and to set up a correct validation. We had to complete the validation set with rows of not shown ads.

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

**Causes of validation problems in submission stage:**

1. Too little data in public leaderboard: trust your gut in this case.
2. Check if you made correct test/train split based on mimic of public leaderboard.
3. Different distribution in train and test.

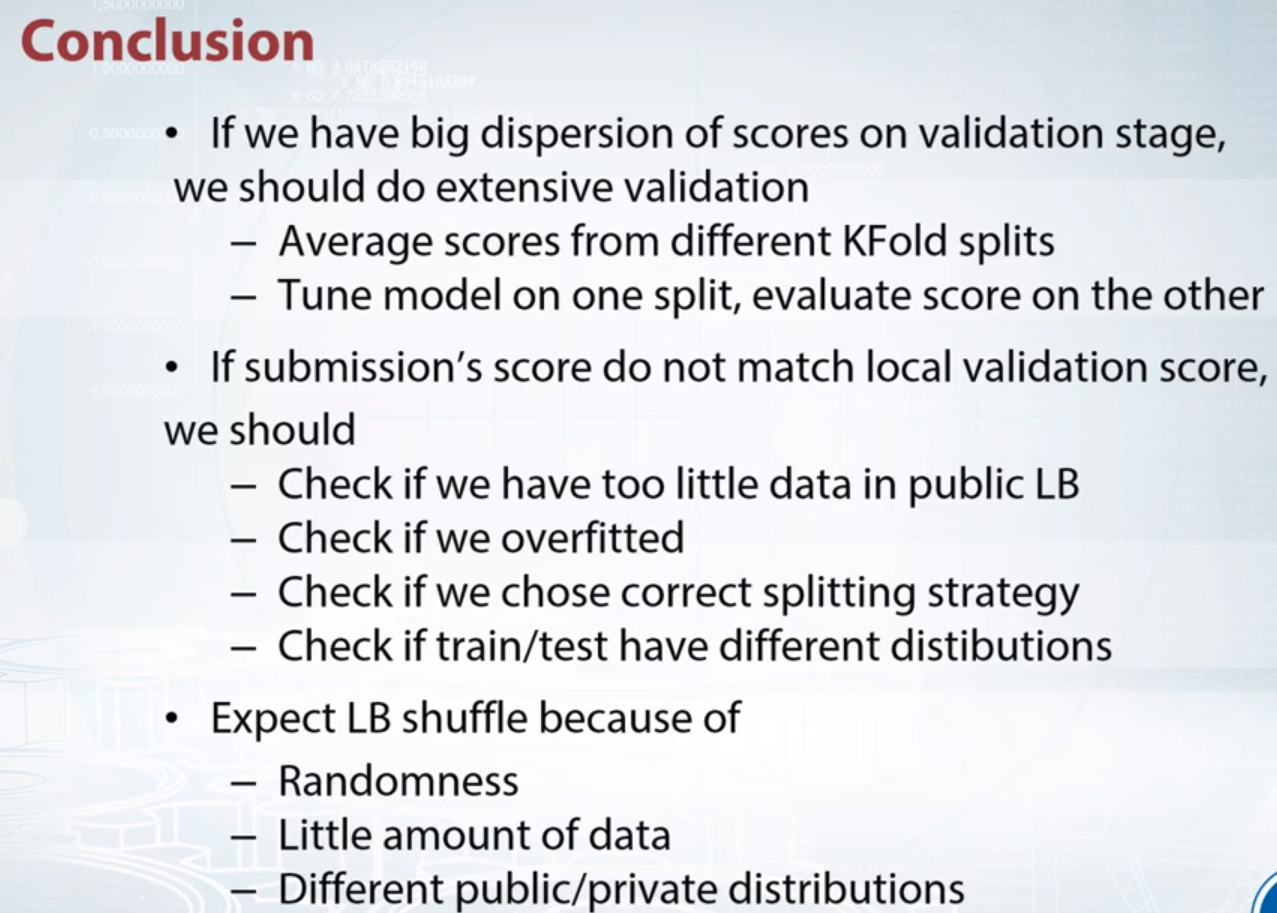
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**LB SHUFFLE**

leaderboard shuffle happens when participants position some public and private leaderboard drastically different. This is seen after the final results of the competition are out.

Expect LB shuffle because of:

1. Randomness in data
2. Too little data
3. Different distribution in public and private distributions.



Avg score from different k-folds here mean that. Do a complete k fold and take all the scores, then do another full k fold take all these scores and avg up all of them.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

We should keep on using KFold scheme as the data is homogeneous and KFold is the most computationally efficient scheme.

We can use a simple holdout validation scheme because the data is homogeneous.

**Correct! If scores on different folds are similar, we indeed can use holdout split. In fact, this is often the case.**

Leave-one-out because the data is not homogeneous.

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

LOO

KFold

**Correct. This is the most frequent way to deal with this kind of situations. Also, scores deviation in KFold will help you to select statistically significant change in scores while tuning a model.**

Holdout

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

True Correct. For an explanation check out the third video in the module about choosing a train/test split.

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

Most of the competitors have very similar scores

**Correct**

In this case randomness can shuffle scores on the private leaderboard

Different public/private data or target distributions

**Correct**

In this case competitors can receive quite unexpected scores on private LB.

Little amount of training or/and testing data

**Correct**

In this case randomness can shuffle scores on the private leaderboard

# Question 1

## Select true statements

Correct answers:

* We use validation to estimate the quality of our model. Correct! This is the main purpose of validation.
* The logic behind validation split should mimic the logic behind train-test split. Correct! This is the main rule of making a reliable validation.
* Underfitting refers to not capturing enough patterns in the data. Correct! Because a model cant utilize all existing patterns, it has lower quality than it could have.

Incorrect answers:

* The model, that performs best on the validation set is guaranteed to be the best on the test set. Incorrect. Target in the test set can have different distribution and our score estimation can fail.
* Performance increase on a fixed cross-validation split guaranties performance increase on any cross-validation split. Incorrect. You can overfit to the specific CV-split. You should change your split from time to time to reduce the chance of overfitting.

# Question 2

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

Correct answers:

* Generally, this approach is based on the assumption that the test data may have a different target distribution compared to the train data. If that would be the true, the submission which was chosen based on Public LB, will perform better. If, otherwise, the above distributions will be similar, the submission which was chosen based on validation scores, will perform better.

Incorrect answers:

* Generally, this approach is based on the assumption that people rarely tend to overfit to the Public LB. Almost always you have a lot of data in the test set and it is quite hard to overfit. Indeed, this render validation useless.
* Generally, this approach is based on the assumption that validation is rarely valid in competitions. Almost always it is hard to trust your validation and thus you should account for both cases if the validation will succeed and if the validation will fail.

# Question 3

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

### Campaign\_id, Date, {some features}, Number\_of\_new\_customers

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

## Identify train/test split in a competition.

Correct answer:

* Combined split. For each campaign train and test are divided by a date, and this date can be different for different campaigns. Thus, split is made by id and by time.

Incorrect answers:

* Random split
* Time-based split
* Id-based split

# Question 4

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

Correct answers:

* Different scores/optimal parameters between folds. Correct. This can be identified during validation.
* Public leaderboard score will be unreliable because of too little data. Correct. Usually you can estimate variance of Public LB score using validation. You need to train a model and see how its score varies on different folds with the same size as Public LB.
* Train and test data are from different distributions. Correct! Often enough we can find out this during EDA. To refresh your memory about this problem, review the last video in the Validation module.

Incorrect answers:

* Train and test target distribution are from different distributions. Incorrect! To do this, we would need to have test target values, which is not possible in a competition.

**ADDITIONAL LINKS**

[**https://scikit-learn.org/stable/modules/cross\_validation.html**](https://scikit-learn.org/stable/modules/cross_validation.html)

**Above link shows all types of cross validation implemented in sklearn**

I vow to come out with some principles systematically select final models. Here are the lessons learnt:

* Always do cross-validation to get a reliable metric. If you don’t, the validation score you get on a single validation set is unlikely to reflect the model performance in general. Then, you will likely see a model improvement in that single validation set, but actually performs worse in general. *Keep in mind the CV score can be optimistic, but your model is still overfitting.*
* Trust your CV score, and not LB score. The leaderboard score is scored only on a small percentage of the full test set. In some cases, it’s only a few hundred test cases. Your cross-validation score will be much more reliable in general.
  + If your CV score is not stable (perhaps due to ensembling methods), you can run your CV with more folds and multiple times to take average.
  + If a single CV run is very slow, use a subset of the data to run the CV. This will help your CV loop to run faster. Of course, the subset should not be too small or else the CV score will not be representative.
* For the final 2 models, pick very different models. Picking two very similar solutions means that your solutions either fail together or win together, effectively meaning that you only pick one model. You should reduce your risk by picking two confident but very different models. *You should not depend on the leaderboard score at all.*
  + Try to group your solutions by methodologies. Then, pick the best CV score model from each group. Then compare these best candidates of each group, pick two.
    - Example: I have different groups 1) Bagging of SVMs 2) RandomForest 3) Neural Networks 4) LinearModels. Then, each group should produce one single best model, then you pick 2 out of these.
  + Pick a robust methodology. Here is the tricky part which depends on experience, even if you have done cross validation, you can still get burned: Sketchy methods of improving the CV score like making cubic features, cubic root features, boosting like crazy, magical numbers(without understanding it), etc, will likely be a bad model to pick even if the CV score is good. Unfortunately, you will probably have to make this mistake once to know what this means.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Full data leakages section left**

We'll define leakage in a very general sense as

an unexpected information in the data that

allows us to make unrealistically good predictions.

Data leaks are very, very bad.

They are completely unusable in real world.

They usually provide way too much signal and thus make competitions lose its main point,

and quickly turn them into a leak hunt race.

Time series data leak:

Future picking:

When you enter a time serious competition at first,

check train, public, and private splits.

If even one of them is not on time,

then you found a data leak.

In such case, unrealistic features like prices next week will be the most important.

But even when split by time,

data still contains information about future.

We still can access the rows from the test set.

Meta information in dataset. Like in images meta information we have things like camera, date and other details as well

Leakage in ids as well. They may carry information and so make sure you include it in during training

To see some improvements if they are there.

Row based indexing. Sometimes adding row index and number helped as the organizers shuffled the

data well and may seem to increase the models prediction.

Another type of leakage could be found in IDs.

IDs are unique identifiers of every row usually used for convenience.

It makes no sense to include them into the model.

It is assumed that they are automatically generated.

In reality, that's not always true.

ID may be a hash of something,

probably not intended for disclosure.

It may contain traces of information connected to target variable.

It was a case in Caterpillar competition.

A link ID as a feature slightly improve the result.

So I advise you to pay close attention to IDs and

always check whether they are useful or not.

Next is row order.

In trivial case, data may be shuffled by target variable.

Sometimes simply adding row number or relative number,

suddenly improves this course.

Like, in Telstra Network Disruptions competition.

It's also possible to find something way more interesting

like in TalkingData Mobile User Demographics competition.

There was some kind of row duplication,

rows next to each other usually have the same label.

This is it with a regular type of leaks.

To sum things up, in this video,

we embrace the concept of data leak and cover data leaks from future picking,

meta data, IDs, and row order.

Leader board probing:

Types of leaderboard probing:

It's leaderboard probing.

**There are actually two types of leaderboard probing.**

1. The first one is simply extracting all ground truth from public part of

The leaderboard.

It's usually pretty harmless, only a little more of straining data.

It is also a relatively easy to do and

I have a submission change on the small set of rows so that you can

Unambiguously calculate ground truth for those rows from leaderboard score.

I suggest checking out the link to Alek Trott's post in additional materials.

He thoroughly explains how to do it very efficiently

with minimum amount of submissions.

Our main focus will be on another type of leaderboard probing.

Remember the purpose of public, private split.

It's supposed to protect private part of test set from information extraction.

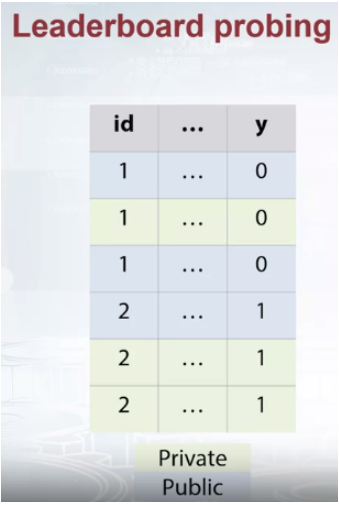
It turns out that it's still vulnerable.

Sometimes, it's possible to submit predictions in such a way

that will give out information about private data.

It's all about consistent categories.

Imagine, a chunk of data with the same target for every row.



Like in the example, rows with the same IDs have the same target.

Organizers split it into public and private parts.

But we still know that that particular chunk has the same label for every role.

After setting all the predictions close to 0 in our submission for

that particular chunk of data, we can expect two outcomes.

The first one is when score improved, it means that ground truth in public is 0.

And it also means that ground truth in private is 0 as well.

Remember, our chunk has the same labels.

The second outcome is when the score became worse.

Similarly, it means that ground truth in both public and private is 1.

Some competitions indeed have that kind of categories.

Categories that with high certainty have the same label.

You could have encountered those type of categories in Red Hat and

West Nile competitions.

It was a key for winning.

With a lot of submissions, one can explore a good part of private test set.

It's probably the most annoying type of data leak.

It's mostly technical and even if it's released close to the competition

deadline, you simply won't have enough submissions to fully exploit it.

Another example, target label could simply have the same distribution for public and

private parts of data.

It was the case in Quora Question Pairs competition.

In that competition there was a binary classification

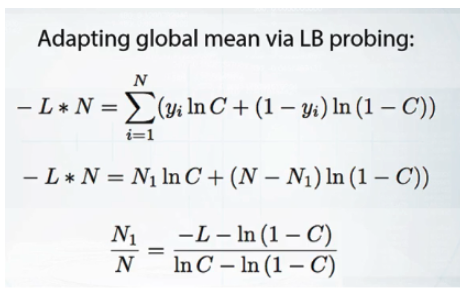
task being evaluated by log loss metric.

What's important target were able had different distributions in train and

test, but allegedly the same and private and public parts of test data.

And because of that, we could benefit a lot via leaderboard probing.

Treating the whole test set as a consistent category.



Take a look at the formula on the slide.

This logarithmic loss for submission with constant predictions C big.

Where N big is the real number of rows,

N1 big is the number of rows with target one.

And L big is the leader board score given by that constant prediction.

From this equation, we can calculate N1 divided by N or

in other words, the true ratio of once in the test set.

That knowledge was very beneficial.

We could use it rebalance training data points

to have the same distribution of target variable as in the test set.

This little trick gave a huge boost in leaderboard score.

As you can see, leaderboard probing is a very serious problem

that could occur under a lot of different circumstances.

Now, finally, I like to briefly walk through the most peculiar and

interesting competitions with data leakage.

And first,

let's take a look at Truly Native competition from different point of view.

In this competition, participants were asked to predict whether the content

in an HTML file is sponsored or not.

As was already discussed in previous video,

there was a data leak in archive dates.

We can assume that sponsored and

non-sponsored HTML files were gotten during different periods of time.

So do we really get rid of data leak after erasing archive dates?

The answer is no.

Texts in HTML files may be connected to dates in a lot of ways.

From explicit timestamps to much more subtle things, like news contents.

As you've probably already realized,

the real problem was not metadata leak, but rather data collection.

Even without metainformation,

machine learning algorithms will focus on actually useless features.

The features that only act as proxies for the date.

The next example is Expedia Hotel Recommendations,

and that competitions, participants worked with logs of customer behavior.

These include what customers searched for, how they interacted with search results,

and clicks or books, and whether or not the search result was a travel package.

Expedia was interested in predicting which hotel group a user is going to book.

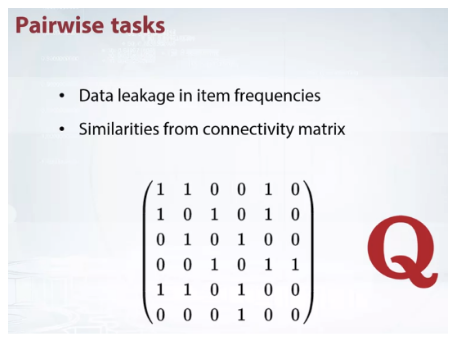
Within the logs of customer behavior, there was a very tricky feature.

At distance from users seeking their hotel.

Turned out, that this feature is actually a huge data leak.

Using this distance, it was possible to reverse engineer two coordinates, and

simply map ground truth from train set to the test set.



The last example is going to cover pairwise tasks.

Where one needs to predict whether the given pair of

items are duplicates or not, like in Quora question pairs competition.

There is one thing common to all the competitions with pairwise tasks.

Participants are not asked to evaluate all possible pairs.

There is always some nonrandom subsampling, and

this subsampling is the cause of data leakage.

Usually, organizers sample mostly hard-to-distinguish pairs.

Because of that, of course, imbalance in item frequencies.

It results in more frequent items having the higher

possibility of being duplicates.

But that's not all.

We can create a connectivity matrix N times N,

where N is the total number of items.

If item I and item J appeared in a pair then we place 1 in I,

J and J, I positions.

Now, we can treat the rows in connectivity matrix as vector representations for

every item.

This means that we can compute similarities between those vectors.

This tricks works for a very simple reason.

When two items have similar sets of neighbors

they have a high possibility of being duplicates.

Question 1

Suppose that you have a credit scoring task, where you have to create a ML model that approximates expert evaluation of an individual's creditworthiness. Which of the following can potentially be a data leakage? Select all that apply.

**Correct answers:**

* An ID of a data point (row) in the train set correlates with target variable. Data was not shuffled, this information can not be used in real-world scenario.
* First half of the data points in the train set has a score of 0, while the second half has scores > 0. Same as above, data was not shuffled, this information can not be used in real-world scenario..

**Incorrect answers:**

* Among the features you have a company\_id, an identifier of a company where this person works. It turns out that this feature is very important and adding it to the model significantly improves your score. This is a perfectly fine categorical feature, don't mix it up with and ID of a data point.

Question 2

What is the most foolproof way to set up a time series competition?

**Correct answers:**

* Split train, public and private parts of data by time. Remove all features except IDs (e.g. timestamp) from test set so that participants will generate all the features based on past and join them themselves. Correct! Only complete removal of all features from test set can guarantee that there is no data leakage.

**Incorrect answers:**

* Make a time based split for train/test and a random split for public/private. Vulnerable to leaderboard probing.
* Split train, public and private parts of data by time. Remove time variable from test set, keep the features. Participants can try to reverse engineer time order and exploit future peeking.

Question 3

Suppose that you have a binary classification task being evaluated by logloss metric. You know that there are 10000 rows in public chunk of test set and that constant 0.3 prediction gives the public score of 1.01. Mean of target variable in train is 0.44. What is the mean of target variable in public part of test data (up to 4 decimal places)?

**Correct answers:**

* ~0.771 Use logloss formula.

Question 4

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

Correct answer is 3. Check image name from inspect element and then see its URL to find that out.

# **NOTEBOOK ASSIGNMENT**

# **Introduction**

In this programming assignment we will illustrate a very severe data leakage, that can often be found in competitions, where the pairs of object should be scored, e.g. predict 11 if two objects belong to the same class and 00 otherwise.

The data in this assignment is taken from a real competition, and the funniest thing is that *we will not use training set at all* and achieve almost 100% accuracy score! We will just exploit the leakage.

Now go through the notebook and complete the assignment.

**import** numpy **as** np

**import** pandas **as** pd

**import** scipy.sparse

# **Load the data**

Let's load the test data. Note, that we don't have any training data here, just test data. Moreover, *we will not even use any features* of test objects. All we need to solve this task is the file with the indices for the pairs, that we need to compare.

Let's load the data with test indices.

test = pd.read\_csv('../readonly/data\_leakages\_data/test\_pairs.csv')

test.head(10)

Out[85]:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **pairId** | **FirstId** | **SecondId** |
| **0** | 0 | 1427 | 8053 |
| **1** | 1 | 17044 | 7681 |
| **2** | 2 | 19237 | 20966 |
| **3** | 3 | 8005 | 20765 |
| **4** | 4 | 16837 | 599 |
| **5** | 5 | 3657 | 12504 |
| **6** | 6 | 2836 | 7582 |
| **7** | 7 | 6136 | 6111 |
| **8** | 8 | 23295 | 9817 |
| **9** | 9 | 6621 | 7672 |

For example, we can think that there is a test dataset of images, and each image is assigned a unique Id from 00 to N−1N−1 (N -- is the number of images). In the dataframe from above FirstId and SecondId point to these Id's and define pairs, that we should compare: e.g. do both images in the pair belong to the same class or not. So, for example for the first row: if images with Id=1427 and Id=8053 belong to the same class, we should predict 11, and 00 otherwise.

But in our case we don't really care about the images, and how exactly we compare the images (as long as comparator is binary).

**We suggest you to try to solve the puzzle yourself first.** You need to submit a .csv file with columns pairId and Prediction to the grader. The number of submissions allowed is made pretty huge to let you explore the data without worries. The returned score should be very close to 11.

**If you do not want to think much** -- scroll down and follow the instructions below.

test[['FirstId', 'SecondId']].values.ravel()

Out[86]:

array([ 1427, 8053, 17044, ..., 5588, 1767, 11874])

# **EDA and leakage intuition**

As we already know, the key to discover data leakages is careful EDA. So let's start our work with some basic data exploration and build an intuition about the leakage.

First, check, how many different ids are there: concatenate FirstId and SecondId and print the number of unique elements. Also print minimum and maximum value for that vector.

*# YOUR CODE GOES HERE*

print("first id unique " **+** str(test.FirstId.nunique()))

print("second id unique " **+** str(test.SecondId.nunique()))

array = pd.unique(test[['FirstId', 'SecondId']].values.ravel())

print("combined unique in array " **+** str(len(array)))

print("min " **+** str(min(array)) **+** " max " **+** str(max(array)))

first id unique 26325

second id unique 26310

combined unique in array 26325

min 0 max 26324

and then print how many pairs we need to classify (it is basically the number of rows in the test set)

test.shape[0]

Out[88]:

368550

Now print, how many distinct pairs it would be possible to create out of all "images" in the dataset?

*# array is the number of distinct values found in firstId and secondId*

len(array)**\***(len(array)**-**1)**/**2

Out[89]:

346489650.0

So the number of pairs we are given to classify is very very small compared to the total number of pairs.

To exploit the leak we need to **assume (or prove)**, that the total number of positive pairs is small, compared to the total number of pairs. For example: think about an image dataset with 10001000 classes, NN images per class. Then if the task was to tell whether a pair of images belongs to the same class or not, we would have 1000N(N−1)21000N(N−1)2 positive pairs, while total number of pairs was 1000N(1000N−1)21000N(1000N−1)2.

Another example: in [Quora competitition](https://www.kaggle.com/c/quora-question-pairs) the task was to classify whether a pair of qustions are duplicates of each other or not. Of course, total number of question pairs is very huge, while number of duplicates (positive pairs) is much much smaller.

Finally, let's get a fraction of pairs of class 1. We just need to submit a constant prediction "all ones" and check the returned accuracy. Create a dataframe with columns pairId and Prediction, fill it and export it to .csv file. Then submit to grader and examine grader's output.

*# YOUR CODE GOES HERE*

explore = pd.DataFrame(test.pairId)

explore['Prediction'] = int(1)

explore.to\_csv('explore.csv', index=**False**)

explore.head()

Out[90]:

|  |  |  |
| --- | --- | --- |
|  | **pairId** | **Prediction** |
| **0** | 0 | 1 |
| **1** | 1 | 1 |
| **2** | 2 | 1 |
| **3** | 3 | 1 |
| **4** | 4 | 1 |

So, we assumed the total number of pairs is much higher than the number of positive pairs, but it is not the case for the test set. It means that the test set is constructed not by sampling random pairs, but with a specific sampling algorithm. Pairs of class 1 are oversampled.

Now think, how we can exploit this fact? What is the leak here? If you get it now, you may try to get to the final answer yourself, othewise you can follow the instructions below.

# **Building a magic feature**

In this section we will build a magic feature, that will solve the problem almost perfectly. The instructions will lead you to the correct solution, but please, try to explain the purpose of the steps we do to yourself -- it is very important.

## **Incidence matrix**

First, we need to build an [incidence matrix](https://en.wikipedia.org/wiki/Incidence_matrix). You can think of pairs (FirstId, SecondId) as of edges in an undirected graph.

The incidence matrix is a matrix of size (maxId + 1, maxId + 1), where each row (column) i corresponds i-th Id. In this matrix we put the value 1 to the position [i, j], if and only if a pair (i, j) or (j, i) is present in a given set of pais (FirstId, SecondId). All the other elements in the incidence matrix are zeros.

**Important!** The incidence matrices are typically very very sparse (small number of non-zero values). At the same time incidence matrices are usually huge in terms of total number of elements, and it is **impossible to store them in memory in dense format**. But due to their sparsity incidence matrices **can be easily represented as sparse matrices**. If you are not familiar with sparse matrices, please see [wiki](https://en.wikipedia.org/wiki/Sparse_matrix) and [scipy.sparse reference](https://docs.scipy.org/doc/scipy/reference/sparse.html). Please, use any of scipy.sparseconstructors to build incidence matrix.

For example, you can use this constructor: scipy.sparse.coo\_matrix((data, (i, j))). We highly recommend to learn to use different scipy.sparseconstuctors, and matrices types, but if you feel you don't want to use them, you can always build this matrix with a simple for loop. You will need first to create a matrix using scipy.sparse.coo\_matrix((M, N), [dtype]) with an appropriate shape (M, N) and then iterate through (FirstId, SecondId) pairs and fill corresponding elements in matrix with ones.

**Note**, that the matrix should be symmetric and consist only of zeros and ones. It is a way to check yourself.

mat = np.zeros((max(array)**+**1, max(array)**+**1))

**for** i **in** zip(test.FirstId.values, test.SecondId.values):

**if** mat[i] **!**= 1:

mat[i] = 1

## **Now build the magic feature**

Why did we build the incidence matrix? We can think of the rows in this matix as of representations for the objects. i-th row is a representation for an object with Id = i. Then, to measure similarity between two objects we can measure similarity between their representations. And we will see, that such representations are very good.

Now select the rows from the incidence matrix, that correspond to test.FirstId's, and test.SecondId's.

*# Note, scipy goes crazy if a matrix is indexed with pandas' series.*

*# So do not forget to convert `pd.series` to `np.array`*

*# These lines should normally run very quickly*

first\_row = test.FirstId.tolist()

second\_row = test.SecondId.tolist()

len(test.FirstId)

Out[92]:

368550

Our magic feature will be the *dot product* between representations of a pair of objects. Dot product can be regarded as similarity measure -- for our non-negative representations the dot product is close to 0 when the representations are different, and is huge, when representations are similar.

Now compute dot product between corresponding rows in rows\_FirstId and rows\_SecondId matrices.

*# Note, that in order to do pointwise multiplication in scipy.sparse you need to use function `multiply`*

*# regular `\*` corresponds to matrix-matrix multiplication*

f = []

​

**for** first, second **in** zip(first\_row, second\_row):

f.append(np.dot(mat[first], mat[second].T))

​

f = np.array(f)

*# Sanity check*

**assert** f.shape == (368550, )

That is it! **We've built our magic feature.**

# **From magic feature to binary predictions**

But how do we convert this feature into binary predictions? We do not have a train set to learn a model, but we have a piece of information about test set: the baseline accuracy score that you got, when submitting constant. And we also have a very strong considerations about the data generative process, so probably we will be fine even without a training set.

We may try to choose a thresold, and set the predictions to 1, if the feature value f is higer than the threshold, and 0 otherwise. What threshold would you choose?

How do we find a right threshold? Let's first examine this feature: print frequencies (or counts) of each value in the feature f.

*# For example use `np.unique` function, check for flags*

val, bins = np.histogram(f)

print(f.shape)

print(val)

print(bins)

(368550,)

[184185 0 0 47397 40377 33357 26337 19326 12298 5273]

[ 0. 2.1 4.2 6.3 8.4 10.5 12.6 14.7 16.8 18.9 21. ]

Do you see how this feature clusters the pairs? Maybe you can guess a good threshold by looking at the values?

In fact, in other situations it can be not that obvious, but in general to pick a threshold you only need to remember the score of your baseline submission and use this information. Do you understand why and how?

Choose a threshold below:

f[f **>** 6.3] = 1

pred = f

pred[0:10]

Out[95]:

array([ 1., 0., 1., 1., 0., 1., 0., 0., 0., 0.])

# **Finally, let's create a submission**

submission = test.loc[:,['pairId']]

submission['Prediction'] = pred.astype(int)

​

submission.to\_csv('submission.csv', index=**False**)

Now submit it to the grader! It is not possible to submit directly from this notebook, as we need to submit a csv file, not a single number (limitation of Coursera platform).

To download submission.csv file that you've just produced [click here](https://mrswzenwnufdumipqzjcjz.coursera-apps.org/notebooks/Programming%20assignment%2C%20week%202%3A%20Data%20leakages/submission.csv) (if the link opens in browser, right-click on it and shoose "Save link as"). Then go to [assignment page](https://www.coursera.org/learn/competitive-data-science/programming/KsASv/data-leakages/submission) and submit your .csv file in 'My submission' tab.

If you did everything right, the score should be very high.

**Finally:** try to explain to yourself, why the whole thing worked out. In fact, there is no magic in this feature, and the idea to use rows in the incidence matrix can be intuitively justified.

# **Bonus**

Interestingly, it is not the only leak in this dataset. There is another totally different way to get almost 100% accuracy. Try to find it!