Santander Product Recommendation

Kaggle Competition

Content

1. Introduction
2. Data Exploration
3. Models
4. Results

1. Introduction

Task

In this competition, there are 1.5 years of customers behavior data from Santander bank to predict what new products customers will purchase. The data starts at 2015-01-28 and has monthly records of products a customer has, such as "credit card", "savings account", etc. The goal is to predict what additional products a customer will get in the last month, 2016-06-28, in addition to what they already have at 2016-05-28. These products are the columns named: ind_(xyz)_ult1, which are the columns #25 - #48 in the training data.

Evaluation metric

Submissions in this competition are evaluated according to the Mean Average Precision @ 7 (MAP@7). It means that it is needed to predict no more than 7 products for every customer. Note that order matters. But it depends. Order matters only if there is at least one incorrect prediction. In other words, if all predictions are correct, it doesn't matter in which order they are given. This makes sense to show up first the most relevant results.

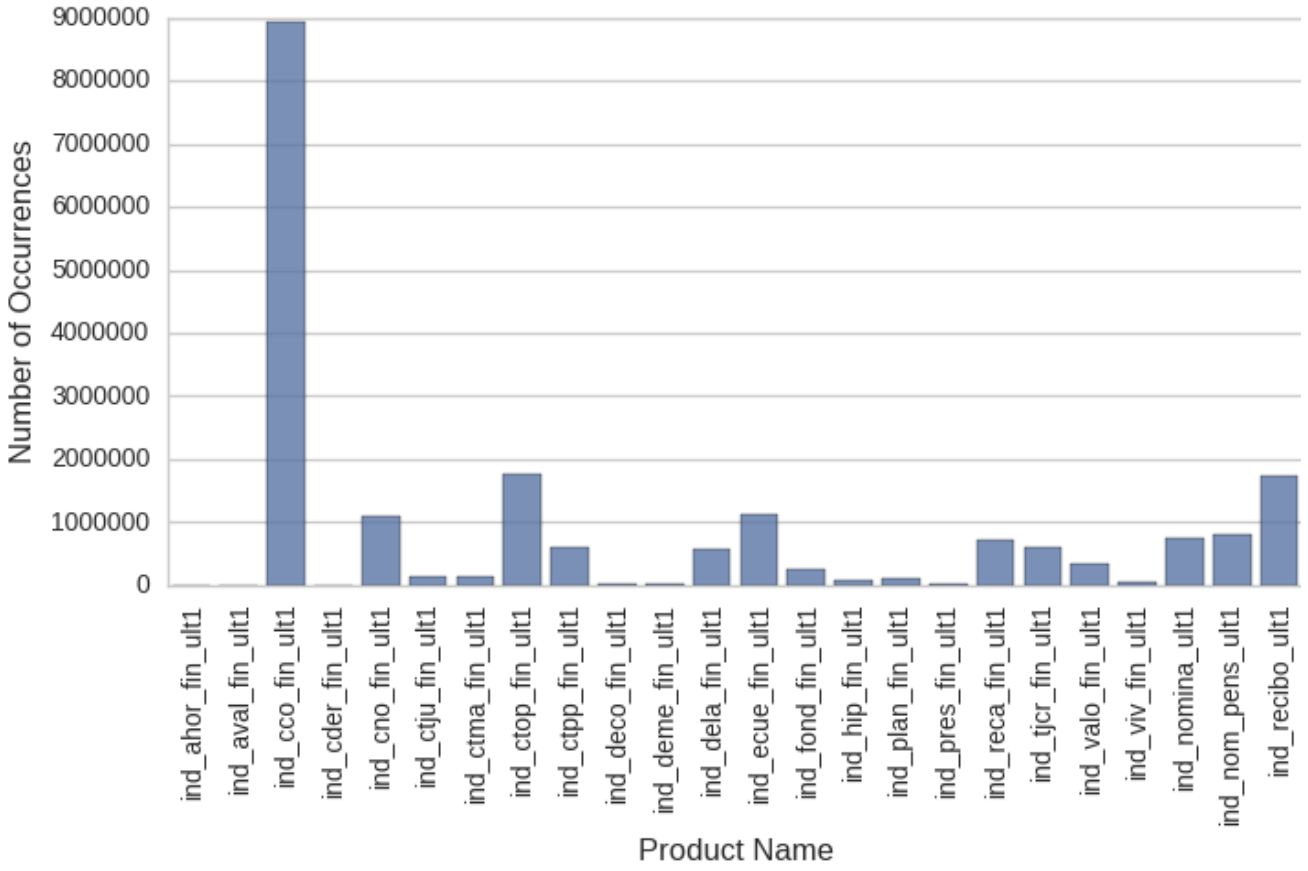
2. Data Exploration

Data Fields

Input space	
fecha_dato	tiprel_1mes
ncodpers	indresi
ind_empleado	indext
pais_residencia	conyuemp
sexo	canal_entrada
age	indfall
fecha_alta	tipodom
ind_nuevo	cod_prov
antiguedad	nomprov
Indrel	ind_actividad_cliente
ult_fec_cli_1t	renta
indrel_1mes	segmento

Target features	
ind_ahor_fin_ult1	ind_ecue_fin_ult1
ind_aval_fin_ult1	ind_fond_fin_ult1
ind_cco_fin_ult1	ind_hip_fin_ult1
ind_cder_fin_ult1	ind_plan_fin_ult1
ind_cno_fin_ult1	ind_pres_fin_ult1
ind_ctju_fin_ult1	ind_reca_fin_ult1
ind_ctma_fin_ult1	ind_tjcr_fin_ult1
ind_ctop_fin_ult1	ind_valo_fin_ult1
ind_ctpp_fin_ult1	ind_viv_fin_ult1
ind_deco_fin_ult1	ind_nomina_ult1
ind_deme_fin_ult1	ind_nom_pens_ult1
ind_dela_fin_ult1	ind_recibo_ult1

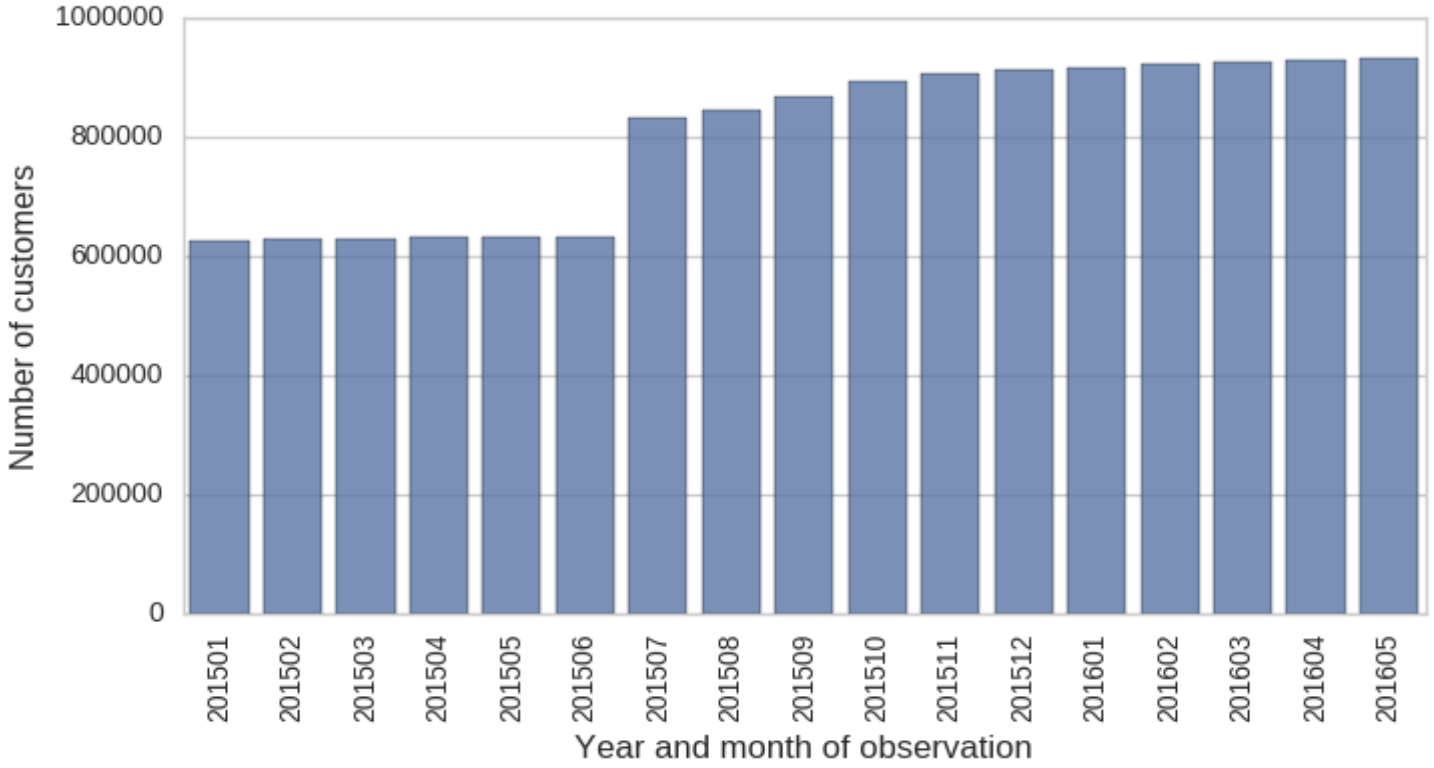
Histogram of the target features



It is easy to see that the product **ind_cco_fin_ult1 (Current Accounts)** is the most popular one, while products **ind_ahor_fin_ult1 (Saving Account)** and **ind_aval_fin_ult1 (Guarantees)** have very small numbers of purchases.

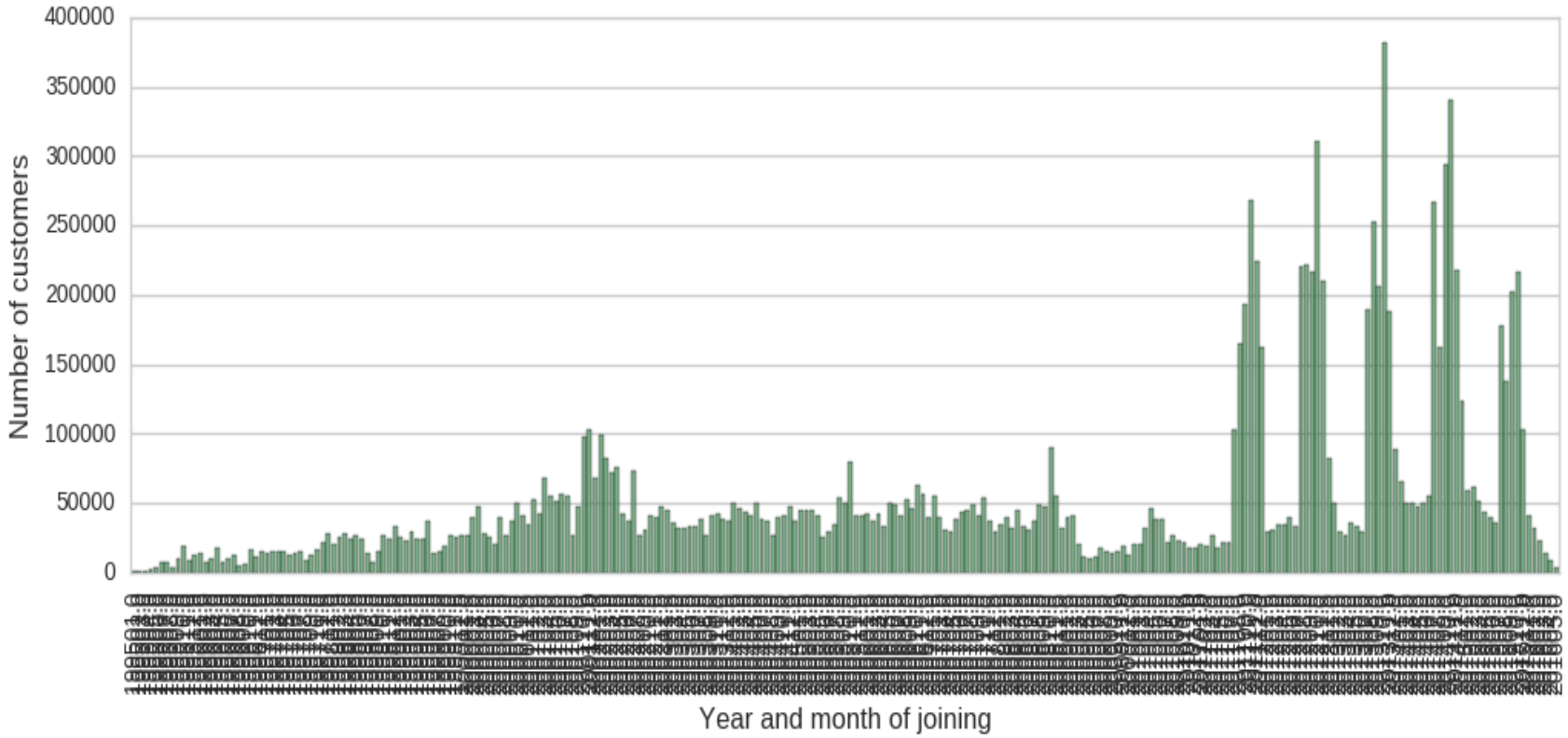
As it was discovered further, the good strategy is to not predict these two features at all.

Exploring Dates 1



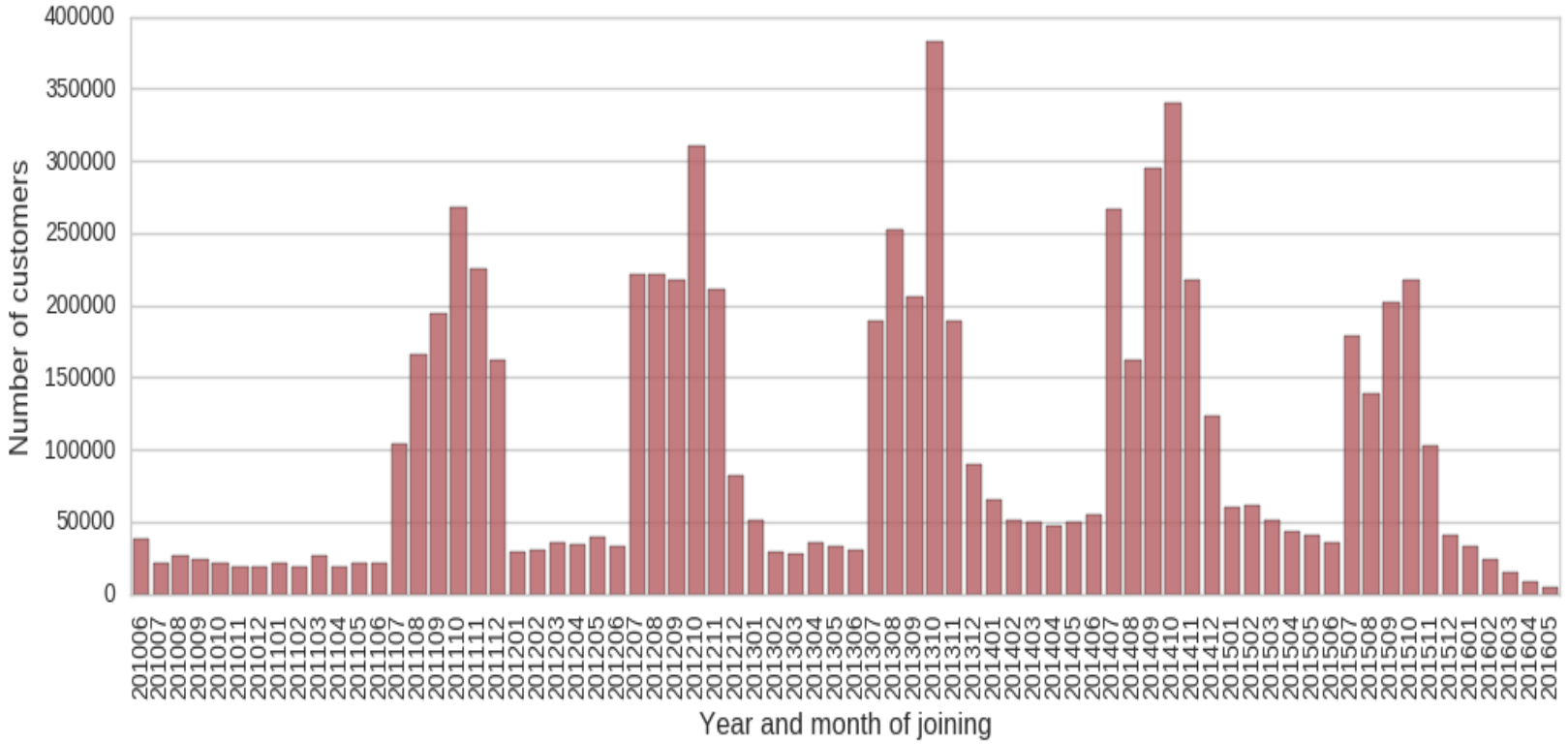
For the first six months of the given train data, the number of customers / observations remain almost same and then there is a sudden spike in the number of customers / observations during July 2015.

Exploring Dates 2



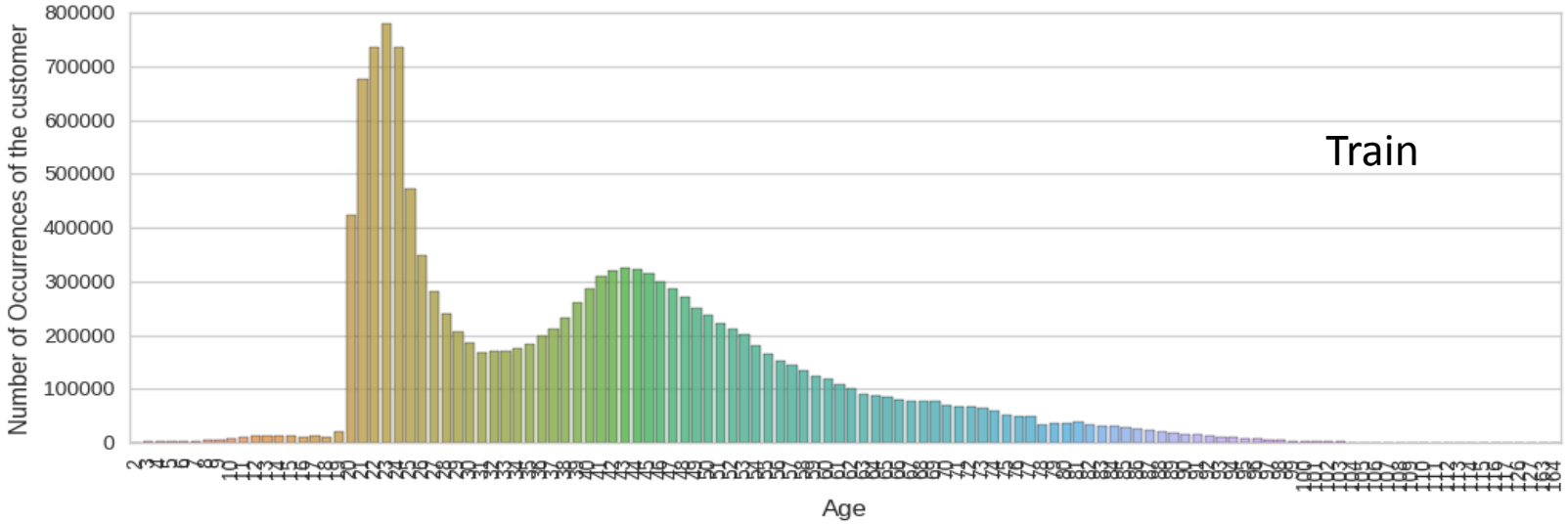
So the first holder date starts from January 1995. But as we can see, the number is high during the recent years.!
Also it seems there are some seasonal peaks in the data. Let us have a close look at them.!

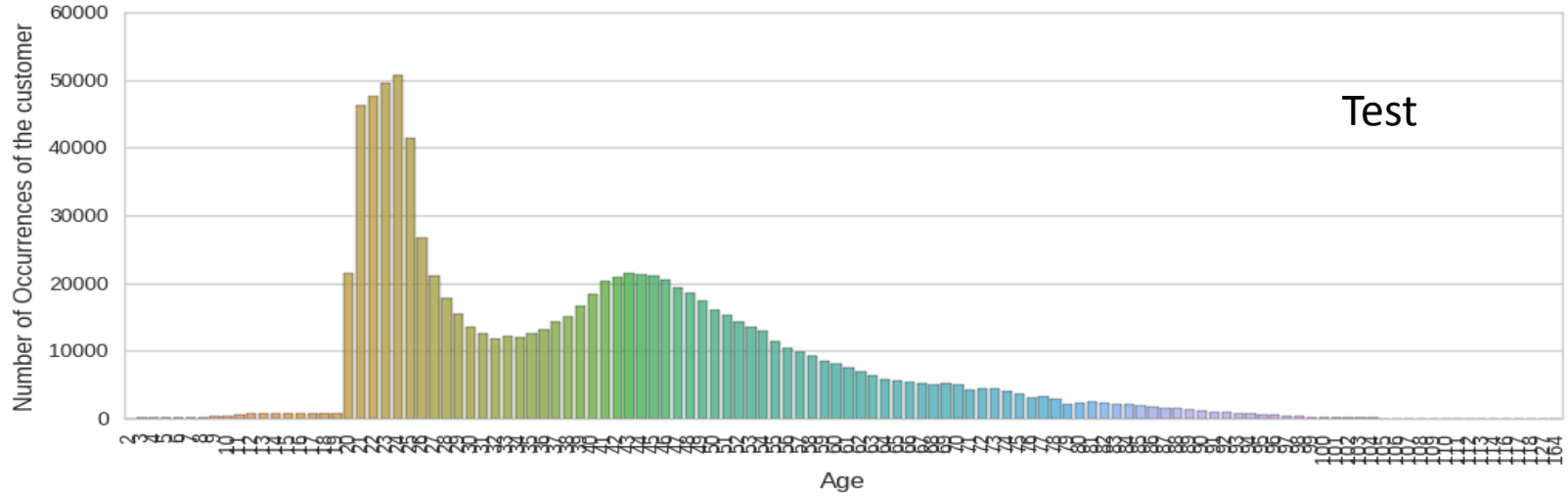
Exploring Dates 3



From 2011, the number of customers becoming the first holder of a contract in the second six months is much higher than the first six months in a calendar year and it is across all years after that. June is like an edge between the two seasons. It is a very specific month.

Age distribution

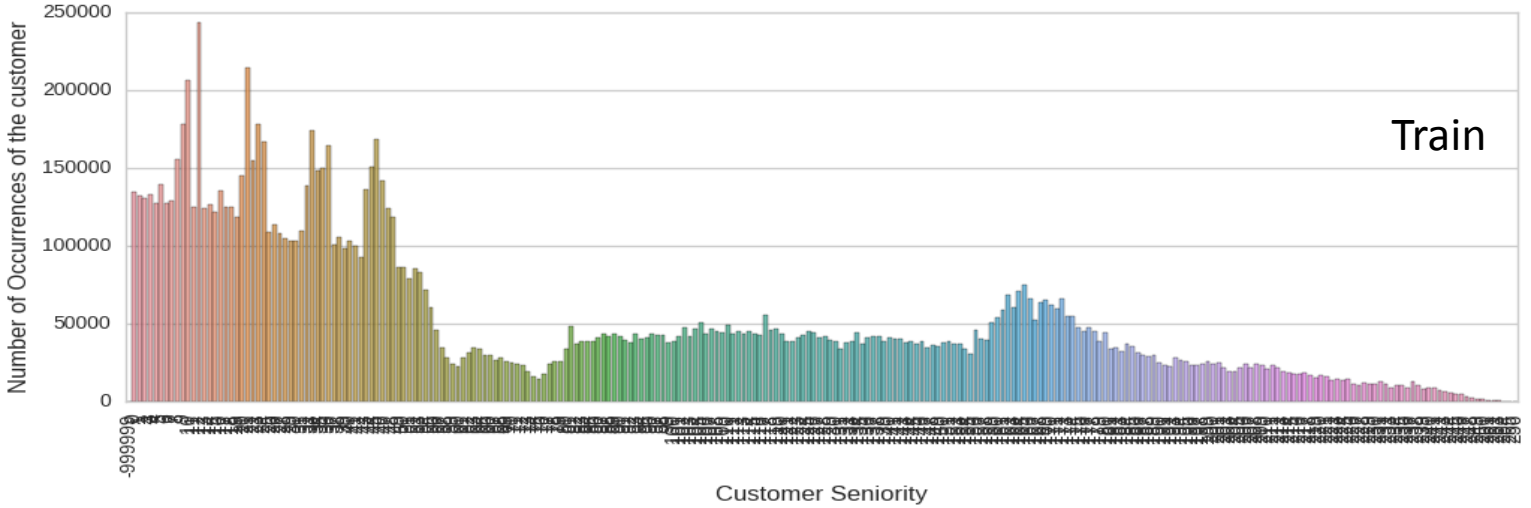


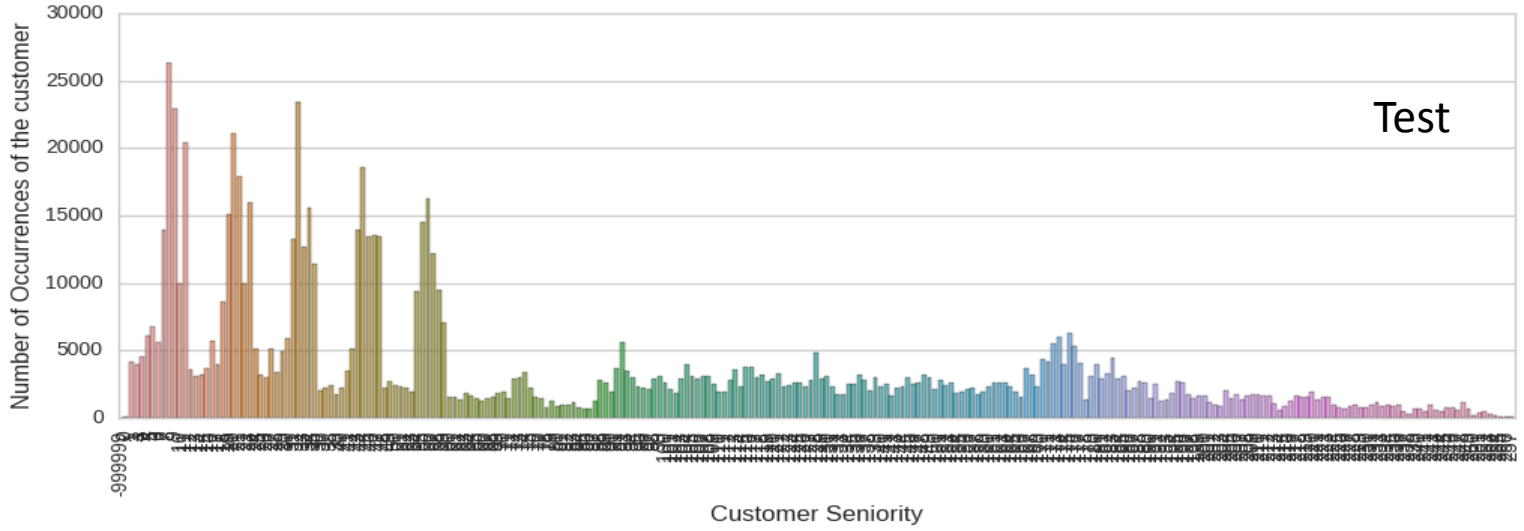


We could see that there is a very long tail at both the ends. So we can have min and max cap at some points respectively (I would use 20 and 86 from the graph).

Good to see that the distribution is similar between train and test.!

Antiguedad (Customer seniority in months)

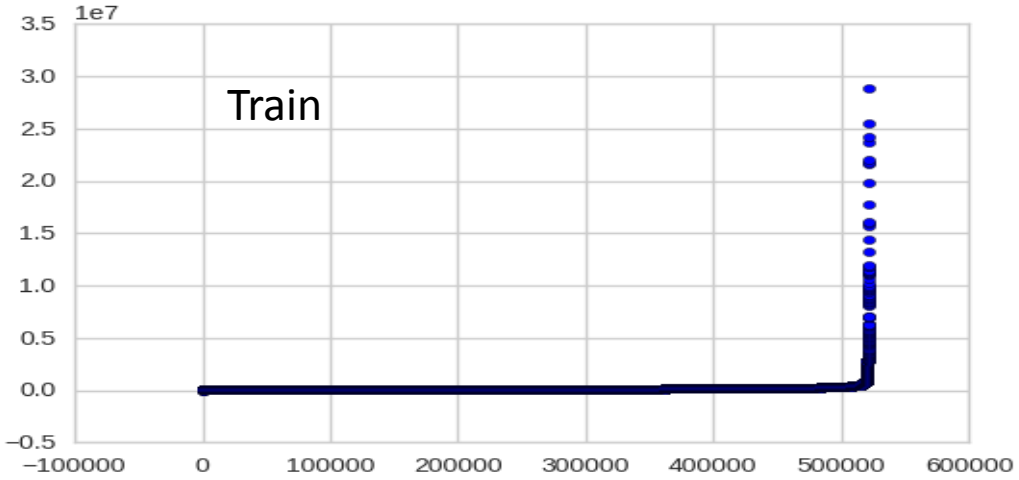


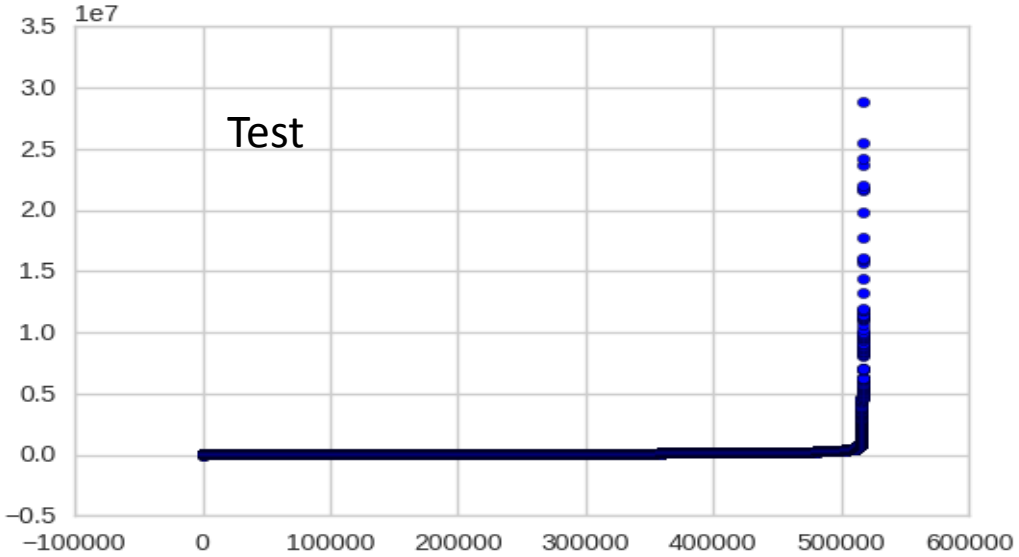


There are few peaks and troughs in the plot but there are no visible gaps or anything as such which is alarming (at least to me.!)

The distributions are pretty much the same, the picks are identical.

Renta (Gross income) distribution

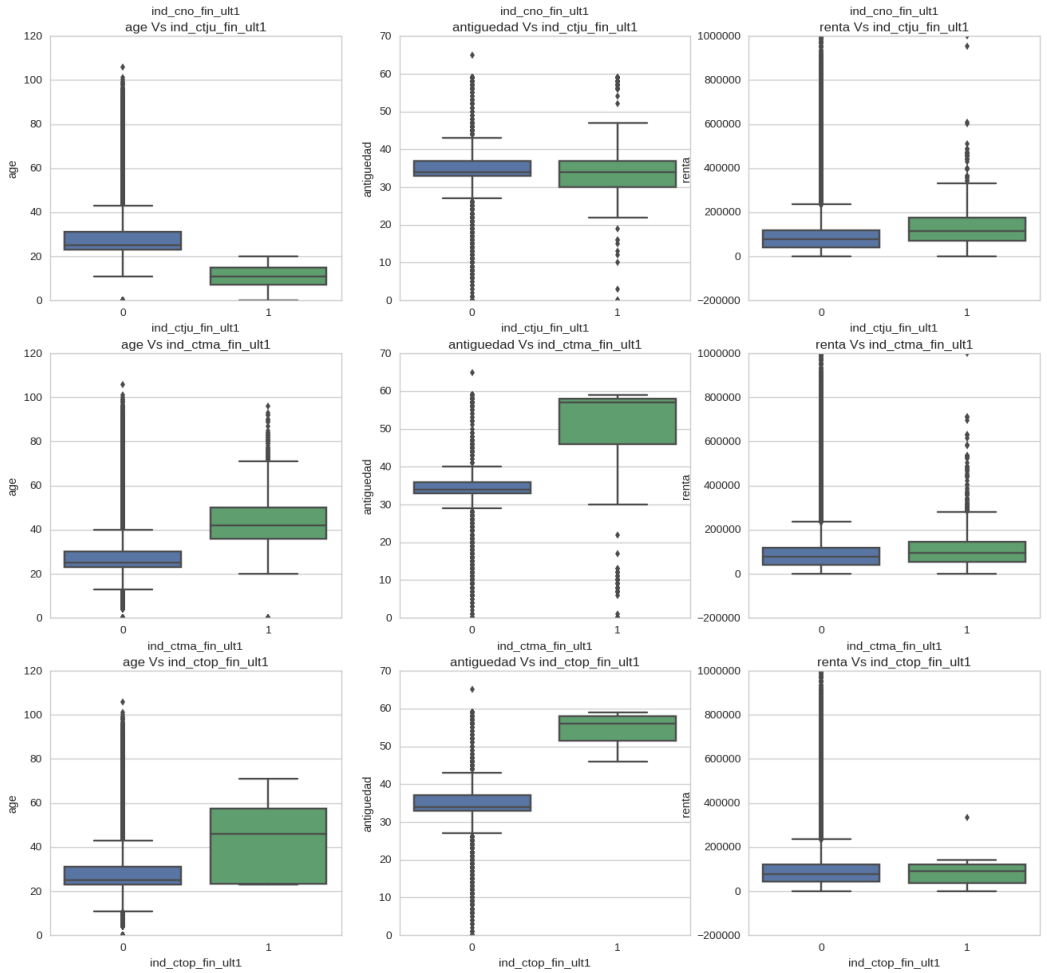




It seems the distribution of rent is highly skewed. There are few very high valued customers present in the data.

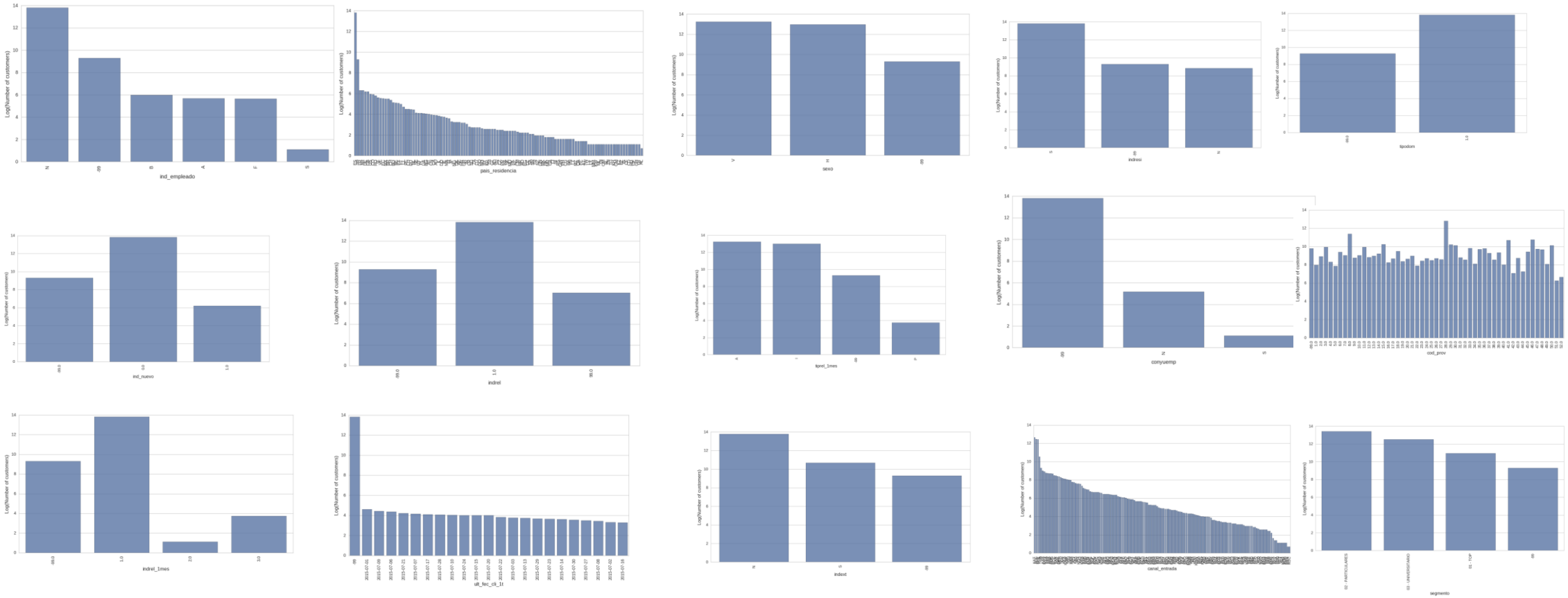
The distributions look similar though.

Numerical vs. Target



Seems all these numerical variables have some predictive power since they show some different behavior between 0's and 1's.

Categorical features



3. Models

K-Nearest Neighbors

```
print("Building model..")
clf = KNeighborsClassifier()
clf = clf.fit(train_X, train_y)
print("Predicting..")
preds = clf.predict_proba(test_X)
```

```
print("Getting the top products..")
target_cols = np.array(target_cols)
preds = np.argsort(preds, axis=1)
preds = np.fliplr(preds)[:,:7]
test_id = np.array(pd.read_csv("test_ver2.csv", usecols=['ncodpers'])['ncodpers'])
final_preds = [" ".join(list(target_cols[pred])) for pred in preds]
out_df = pd.DataFrame.from_items([('ncodpers', test_id), ('added_products', final_preds)])
out_df.to_csv('knn.csv', index=False)
```

XGBoost

```
def runXGB(train_X, train_y, seed_val=123):
    param = {}
    param['objective'] = 'multi:softprob'
    param['eta'] = 0.06
    param['max_depth'] = 6
    param['silent'] = 1
    param['num_class'] = 22
    param['eval_metric'] = "map@7"
    param['min_child_weight'] = 1
    param['subsample'] = 0.8
    param['colsample_bytree'] = 0.8
    param['seed'] = seed_val
    num_rounds = 60
    plst = list(param.items())
    xgtrain = xgb.DMatrix(train_X, label=train_y)
    model = xgb.train(plst, xgtrain, num_rounds)
    return model
```

Simple Neural Network

```
input_X = T.imatrix()
target_y = T.ivector()
```

```
input_layer = lasagne.layers.InputLayer(shape=(None, 41), input_var=input_X)
nn = lasagne.layers.DenseLayer(input_layer, num_units=38, nonlinearity=lasagne.nonlinearities.rectify)
nn = lasagne.layers.DenseLayer(nn, num_units=28, nonlinearity = lasagne.nonlinearities.rectify)
nn_output = lasagne.layers.DenseLayer(nn, num_units = 22, nonlinearity = lasagne.nonlinearities.softmax)
```

```
y_predicted = lasagne.layers.get_output(nn_output)
all_weights = lasagne.layers.get_all_params(nn_output, trainable=True)
```

```
loss = lasagne.objectives.categorical_crossentropy(y_predicted, target_y).mean()
accuracy = lasagne.objectives.categorical_accuracy(y_predicted, target_y).mean()
updates_sgd = lasagne.updates.adam(loss, all_weights, learning_rate=0.01)
```

```
train_fun = theano.function([input_X,target_y],[loss,accuracy], allow_input_downcast=True, updates=updates
_sgd)
```

```
y_predicted_det = lasagne.layers.get_output(nn_output, deterministic=True)
accuracy_det = lasagne.objectives.categorical_accuracy(y_predicted_det, target_y).mean()
accuracy_fun = theano.function([input_X,target_y], accuracy_det)
```

Neural Network with Embedding

```
inputs_layers = []
i = 0
for col in cat_cols:
    inputs_layers.append(lasagne.layers.InputLayer((None, ), input_var=inputs[i]))
    i = i + 1
for col in target_cols:
    inputs_layers.append(lasagne.layers.InputLayer((None, ), input_var=inputs[i]))
    i = i + 1
inputs_layers.append(lasagne.layers.InputLayer((None, 4), input_var=inputs[i]))
concat_layers = []
i = 0
for col in cat_cols:
    concat_layers.append(lasagne.layers.EmbeddingLayer(inputs_layers[i], input_size=max(train_X[:,i])+1,
                                                       output_size=32))
    i = i + 1
for col in target_cols:
    concat_layers.append(lasagne.layers.EmbeddingLayer(inputs_layers[i], input_size=max(train_X[:,4+i])+1,

                                                       output_size=32))
    i = i + 1
concat_layers.append(lasagne.layers.DenseLayer(inputs_layers[i],
                                               num_units=16, nonlinearity=lasagne.nonlinearities.rectify))
nn1 = lasagne.layers.concat(concat_layers)
nn2 = lasagne.layers.DenseLayer(nn1, 1024, nonlinearity=lasagne.nonlinearities.rectify)
nn3 = lasagne.layers.DenseLayer(nn2, 512, nonlinearity=lasagne.nonlinearities.rectify)
nn3 = lasagne.layers.DenseLayer(nn3, 256, nonlinearity=lasagne.nonlinearities.rectify)
nn3 = lasagne.layers.DenseLayer(nn3, 128, nonlinearity=lasagne.nonlinearities.rectify)
nn3 = lasagne.layers.DenseLayer(nn3, 64, nonlinearity=lasagne.nonlinearities.rectify)
n_output = lasagne.layers.DenseLayer(nn3, 22, nonlinearity=lasagne.nonlinearities.softmax)
```

4. Results

Public Score MAP@7

KNN	0.022102
XGBoost	0.028848
Simple NN	0.025951
Embedding NN	0.026012