Data to Model Relevance. Dropping Drifting Features.

Data to Model Relevance. Dropping Drifting Features. 1
How (dis)similar are train and test data? 1
Covariate Shift 2
Test relevance weights 4
Predicting ‘is_train’ for Test Set 7
Predicting ‘is_train’ for Validation Set 7
Drifting Features 9
Dropping drifting features to improve the model. 11
Sberbank Russian Housing Market 12
Home Credit Default Risk 15
Conclusion 18
References 18

How (dis)similar are train and test data?

There may be cases when even after selecting the best model (with cross-validation) that performs well on training data, this model fails to match up to the same performance for the test data.

The reason for this may be some inherent patterns in test data that the model is not capturing.

These new patterns result in a *dataset shift*.

Types of dataset shift:

- Shift in the independent variables (Covariate Shift) - training and test distributions are different but function mapping input variables to target variable remains the same.
- Shift in the target variable (Prior probability shift)
- Shift in the relationship between the independent and the target variable (Concept Shift)

Covariate Shift

Target variable has the same functional dependency on input variables in the train and test set, but variables have different distributions.

$$P_{train}(y|x) = P_{test}(y|x)$$

$$P_{train}(X) \neq P_{test}(X)$$

Main idea: The relevance of test data to a trained model can be calculated by checking the distribution of input variables in train and test data.

If there exists a covariate shift, then upon mixing train and test we'll be able to classify the origin of each data point (whether it is from test or train) with good accuracy.

Then a relevance weight of data from the test set is the probability of this data to be in the train set:

$$w_{train}(x_{test}) = (P_{train}(x_{test}).$$

To evaluate this idea, a simulated train and test set with a clear covariate shift were generated.

The train set is further split into two sets with the same distribution: validation and train sets.

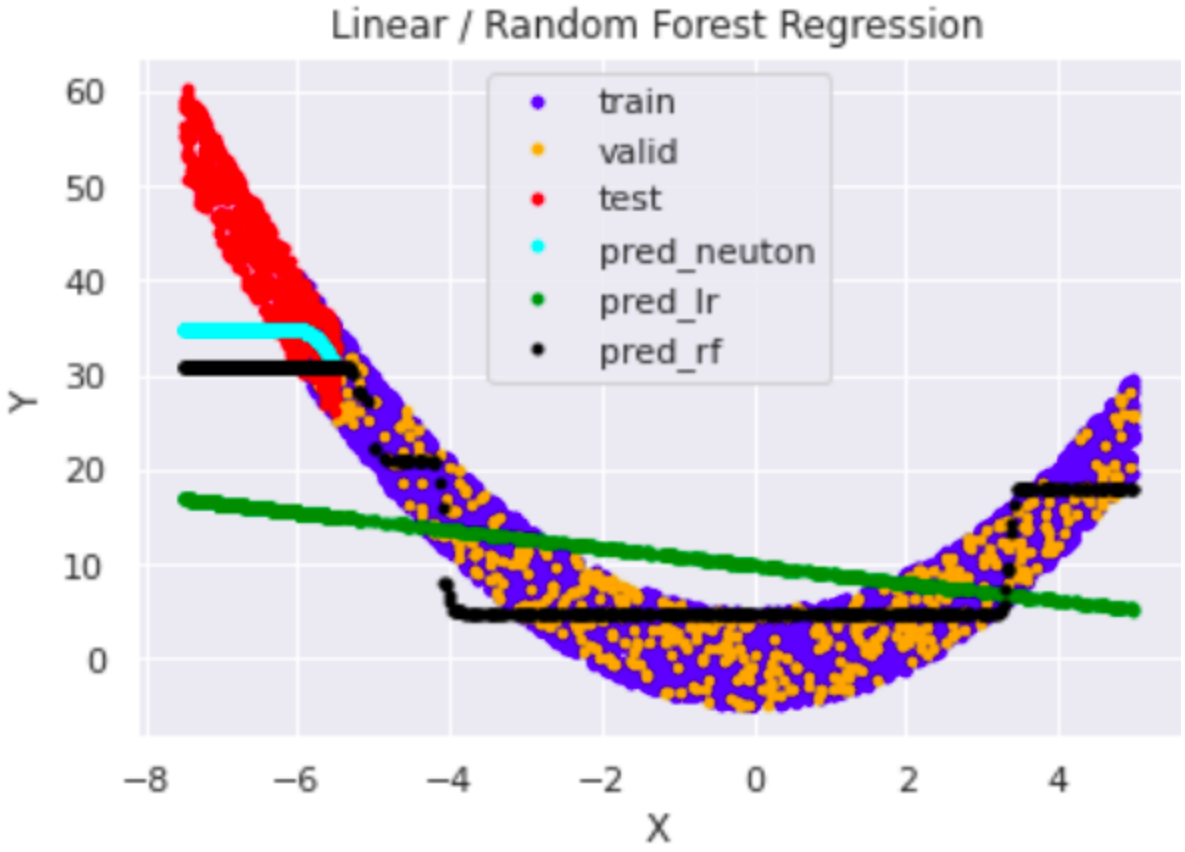
```
n_samples_X = 5000
n_samples_TEST = 500
x = 11*np.random.random(n_samples_X)- 6.0
y = x**2 + 10*np.random.random(n_samples_X) - 5
all_train_df = pd.DataFrame({'x':x, 'y':y})

x = 2*np.random.random(n_samples_TEST) - 7.5
y = x**2 + 10*np.random.random(n_samples_TEST) - 5
test_df = pd.DataFrame({'x':x, 'y':y})
```

```
train_df, valid_df = train_test_split(all_train_df,
test_size=0.1, random_state=0)
print("Train {} Test {} Valid {} ".format(len(train_df),
len(test_df), len(valid_df)))
```

After splitting data, we have 3 datasets:

Dataset	Number of elements
train	4500
validation	500
test	500

We train simple Linear Regression, Random Forest Regression, and Neuton AI models on the train set and predict $y$for validation and test sets. As expected, because of the covariate shift, all three models have very poor results when predicting test data:



Test relevance weights

The relevance of test to train data can be calculated as the probability of test instances in the train set. To find these probabilities or relevance weights, train and test data must be combined in one dataset with the additional target feature `is_train`.
Target `is_train = 1` for train data and `is_train = 0` for the test data. A binary classifier is then trained on this combined data to find test relevance weights. The complete process has the following steps:

- Drop target variable from the train
- Add categorical variable 'is_train' to train and test sets and combine them

- Use StratifiedKFold to train a classifier and predict probabilities of all samples to be from the train and test sets.
- Calculate test sample relevance weights as the probability of test sample to be in the train set.

Function calculating weights:

```
def test_weights(test, train, target):
    """Calculate test weights
    """
    # Adding a column to identify whether a row comes from
train or not
    test.loc[:,('is_train')] = 0
    train.loc[:,('is_train')] = 1

    # Combining test and train data
    df_combine = pd.concat([train, test], axis=0,
ignore_index=True)

    # Dropping 'target' column as it is not present in the
test
    df_combine = df_combine.drop(target, axis =1)

    # Labels
    y = df_combine['is_train'].values

    # Covariates or our independent variables
    x = df_combine.drop('is_train', axis=1).values
    tst, trn = test.values, train.values

    # Predict the labels for each row in the combined
dataset.
    m = RandomForestClassifier(n_jobs=-1, max_depth=5,
min_samples_leaf = 5)
    predictions = np.zeros((y.shape[0],2)) #creating an
```

```
empty prediction array

    # Stratified fold to ensure that percentage for each
class is preserved
    # and we cover the whole data once.
    # For each row the classifier will calculate the
probability of it belonging to train.
    skf = StratifiedKFold(n_splits=20, shuffle=True,
random_state=100)
    for fold, (train_idx, test_idx) in
enumerate(skf.split(x, y)):
        X_train, X_test = x[train_idx], x[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]

        m.fit(X_train, y_train)
        predictions[test_idx] = m.predict_proba(X_test)
#calculating the probability

    # High AUC (greater than 0.8) implies strong covariate
shift between train and test.
    auc =  AUC(y, predictions[:, 1])

    predictions_test = predictions[len(trn):]
    #Test relevance weights equal to the probability of
test sample to be in train set
    test_relevance_weights = predictions_test[:,1]

    return auc, test_relevance_weights, predictions
```

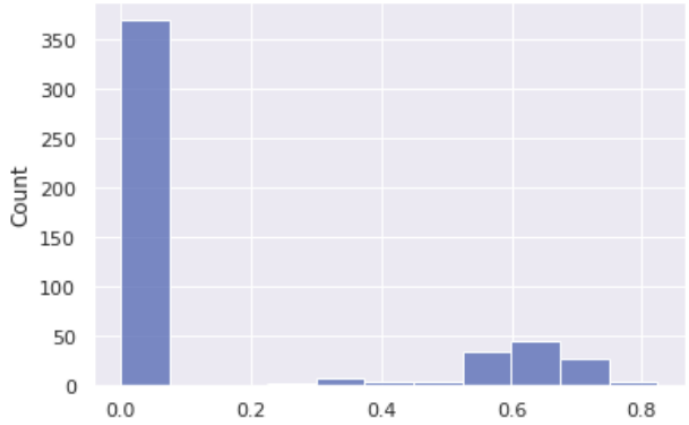
Classification with $AUC - ROC \gg 0.5$ indicates that train and test sets are almost ideally separated. From this follows that variables in train and test sets have different distributions.

Probabilities of test instances in the train set are their model relevance weights. **The smaller the weight the less importance the corresponding test instance has in the model.**

Predicting ‘is_train’ for Test Set

AUC-ROC = 0.9939

Test Relevance Weights

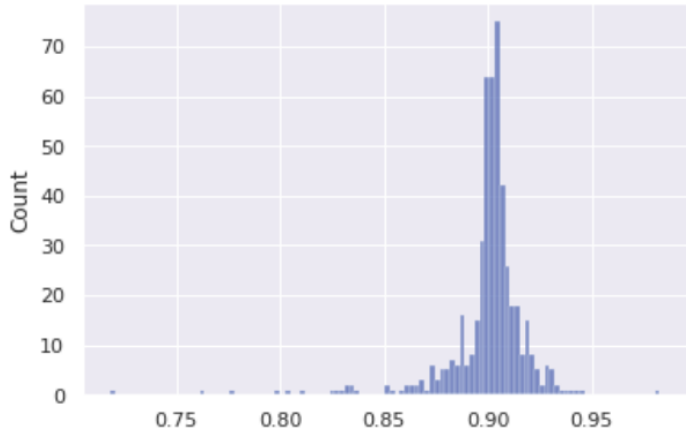


- AUC-ROC> 0.9 indicates that train and test sets are well separated.
- Train and test sets have different distributions, as expected for the test with covariate shift.
- Distribution of test to model relevance has **most** weighs less than 0.1

Predicting ‘is_train’ for Validation Set

AUC = 0.4941

Validation Relevance Weights



- AUC-ROC around 0.5 indicates that train and validation sets have a very similar distribution.
- Distribution of validation set to model relevance has most weights around 0.9.
- Because validation data was randomly chosen from the train set it has no covariate shift.

Drifting Features

Features that have values with a much higher probability found in a test set than in a train set are drifting features (covariate shift).

Main steps to find drifting features:

1. Preprocessing: Impute missing values and label encode categorical variables.
2. Create a random sample of training and test data
3. Check if covariate shift exists. AUC-ROC greater than some threshold (AUC-ROC >> 0.5) indicates that train and test sets are well separated. Train and test sets and have different distributions. Find test relevance weights.
4. Add a new feature ‘is_test’ which has a value depending on whether the observation comes from the training dataset or the test dataset.
5. Combine these random samples into a single dataset. Note that the shape of both the samples of training and test dataset should be nearly equal, otherwise it can be a case of an unbalanced dataset.
6. Train a model taking one feature at a time while having ‘is_test’ as the target variable on a part of the dataset (say ~75%).
7. Predict the rest part(~25%) of the dataset and calculate the value of AUC-ROC.
8. If the value of AUC-ROC for a particular feature is greater than the threshold (0.80), classify that feature as drifting.

Note: There may be a need of changing a threshold value depending on the dataset.

Code to find drifting features:

```
# take random sample from training and test data and combine
them
training = training.sample(7662, random_state=12)
testing = test.sample(7000, random_state=11)
combi = training.append(testing)
y = combi['is_test']
combi.drop('is_test',axis=1,inplace=True)
# find drifting features
model = RandomForestClassifier(n_estimators = 50, max_depth =
5,min_samples_leaf = 5)
drop_list = []
for i in combi.columns:
        score =
cross_val_score(model,pd.DataFrame(combi[i]),y,cv=2,scoring='
roc_auc')
        if (np.mean(score) > drift_threashold):
            drop_list.append(i)
```

Dropping drifting features to improve the model.

Drifting features are not important in the model and dropping them sometimes improves prediction.
Main steps to improve data with covariate shift.

1. Preprocessing (missing values and categorical variables).
2. Make sure data has a covariate shift.
3. Train and predict with all features.
4. Find N top most important features.
5. Detect drifting features and remember them in a list.
6. If there are any, remove important features from the drifting features list.
7. Train and predict without drifting features.
8. Compare scores.

Next, follow the examples dealing with drifting features in some Kaggle datasets.

Sberbank Russian Housing Market

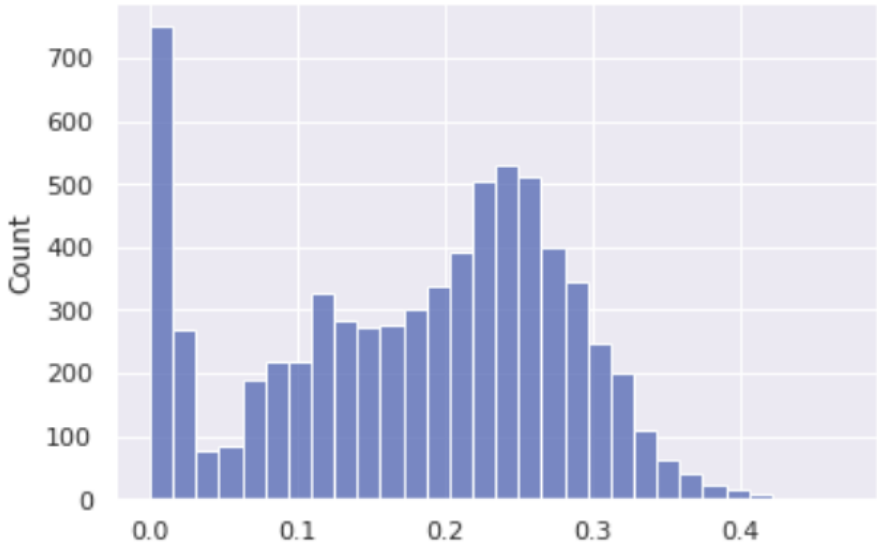
“The aim of this competition is to predict the ‘sale price’ of each property. The target variable is called price_doc in train.csv.”

Features	Train	Test	Train Sample	Test Sample
292	30741	7662	7662	7000

Predicting ‘is_test’ for Test Set

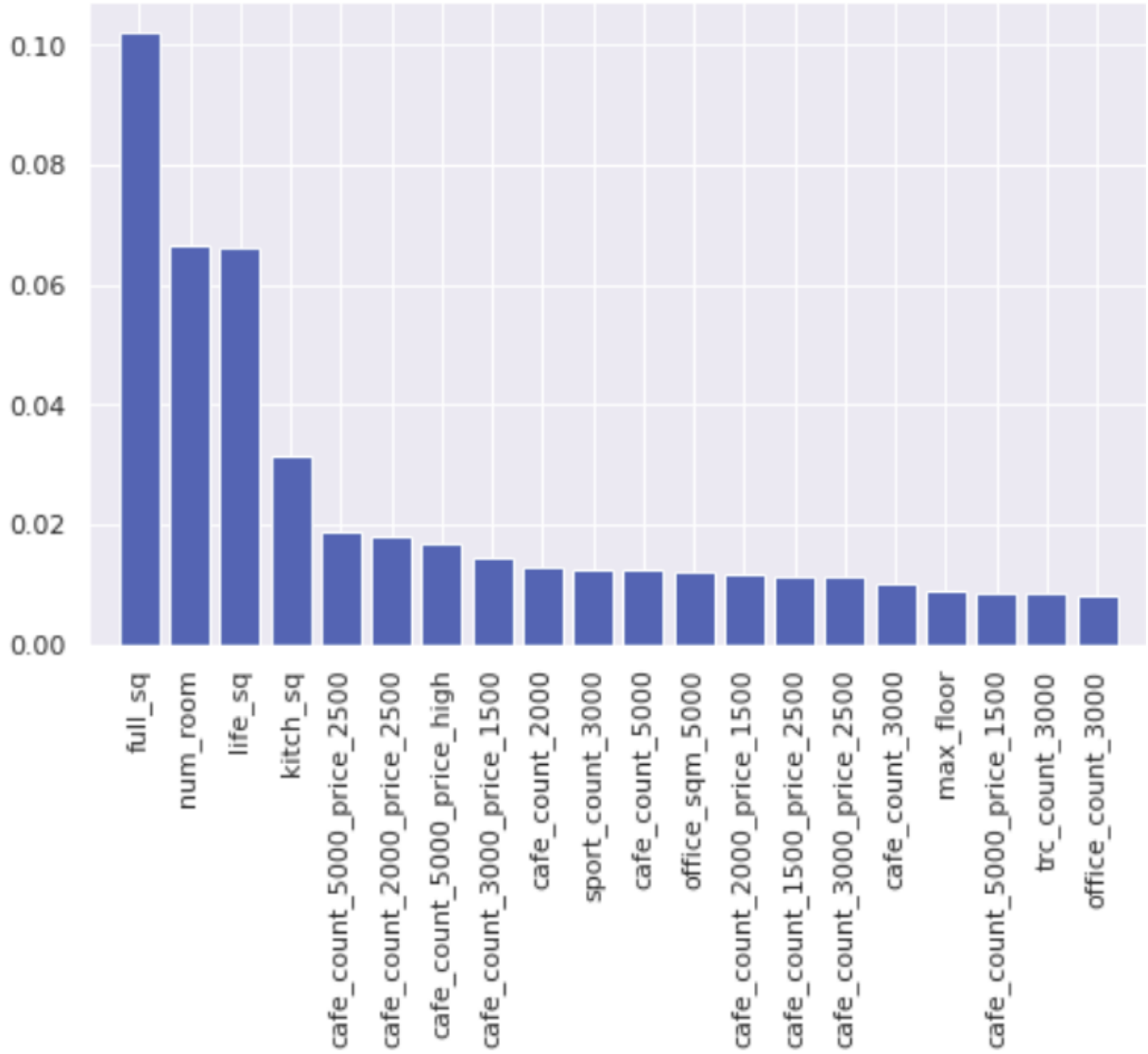
AUC-ROC = 0.9999

Test Relevance Weights:



- AUC-ROC> 0.9 indicates that train and test sets are well separated.
- Train and test sets have different distributions, as expected for the test with covariate shift.
- Distribution of test to model relevance has all weighs less than 0.5

Top 20 Most Important Features



Drifting Features: 9

```
id 1.0
timestamp 0.9595754558675467
life_sq 0.8489691799977626
build_year 0.8008218480814409
kitch_sq 0.8933212887347578
hospital_beds_raion 0.883880467613827
cafe_sum_500_min_price_avg 0.8439294663832643
cafe_sum_500_max_price_avg 0.8442571503150986
cafe_avg_price_500 0.8452648133646568
```

Important Drifting Features To Keep: 2

```
{'kitch_sq', 'life_sq'}
```

Prediction Scores

Kaggle Score With All Features	Kaggle Score Without Drifting Features	Number of Drifting Features	Number of Dropped Features
0.40646	0.39952	9	7

Home Credit Default Risk

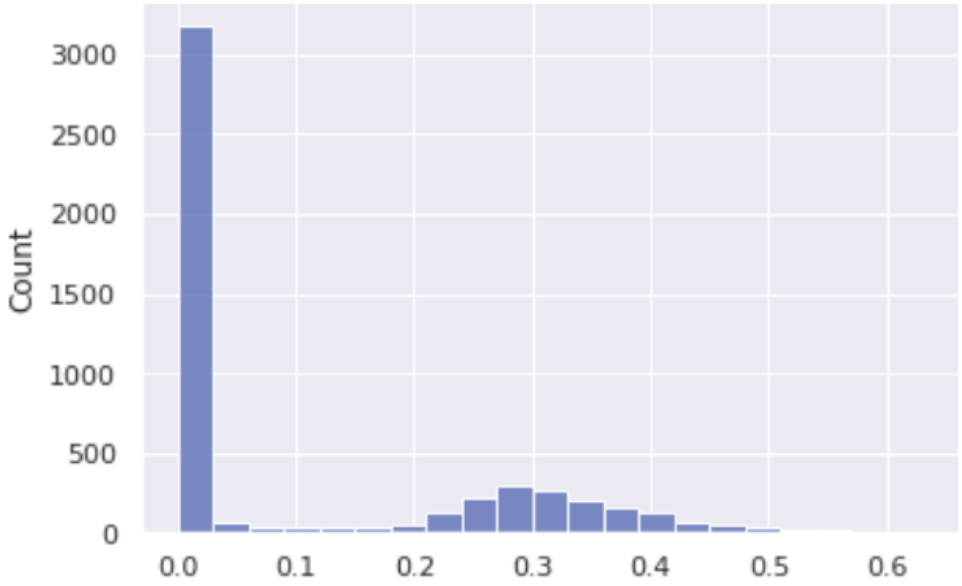
“Predict clients' repayment abilities using static data for all applications. One row represents one loan in our data sample.”

Features	Train	Test	Train Sample	Test Sample
122	307511	48744	5000	5000

Predicting ‘is_test’ for Test Data

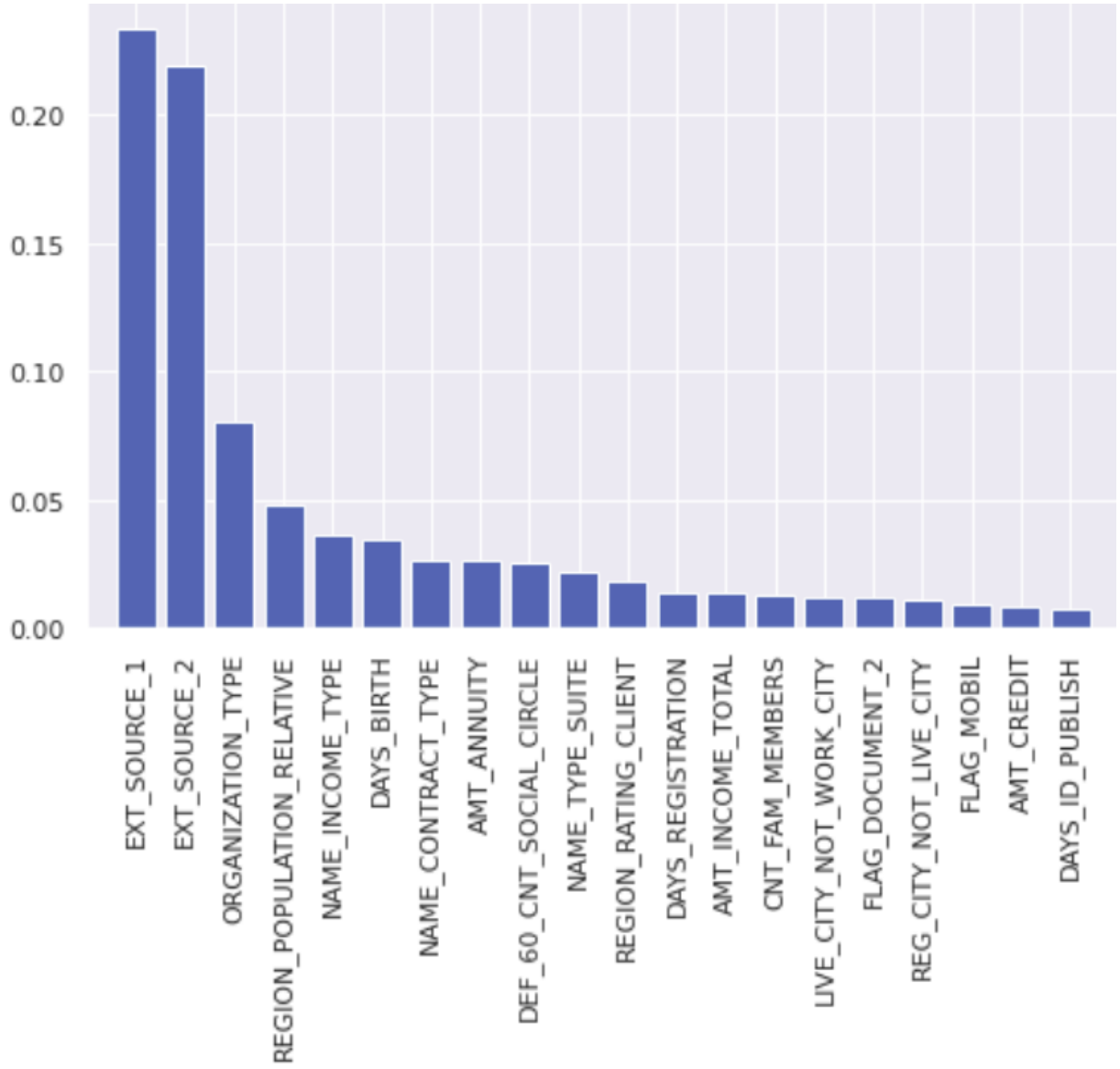
AUC-ROC = 0.9984

Test Relevance Weights:



- AUC-ROC> 0.9 indicates that train and test sets are well separated.
- Train and test sets have different distributions, as expected for the test with covariate shift.
- Distribution of test to model relevance has all weighs less than 0.5

Top 20 Most Important Features



Drifting Features: 46

```
NAME_INCOME_TYPE 0.9461938136256851
OWN_CAR_AGE 0.9472203080135735
EXT_SOURCE_1 0.8767449192676287
APARTMENTS_AVG 0.8735099190811799
BASEMENTAREA_AVG 0.9088818473356453
YEARS_BEGINEXPLUATATION_AVG 0.8693109035313421
YEARS_BUILD_AVG 0.9440639706156542
```

```
COMMONAREA_AVG 0.9522544095163514
ELEVATORS_AVG 0.8902589402244845
ENTRANCES_AVG 0.8698603497781259
FLOORSMAX_AVG 0.8707289965320505
FLOORSMIN_AVG 0.9466746093895664
...
```

Important Drifting Features To Keep: 2

```
{'NAME_INCOME_TYPE', 'EXT_SOURCE_1'}
```

Prediction Scores

Kaggle Score With All Features	Kaggle Score Without Drifting Features	Number of Drifting Features	Number of Dropped Features
0.71348	0.71217	46	44

Conclusion

Covariate shift is the case when data distribution in train and test sets is different but function mapping input variables to target variable remains the same.
To check data for covariate shift one can combine train and test together labeling data according to the origin (whether it is from test or train). Binary classification of this combined data resulting in a high F1 score indicates covariate shift.
Then relevance weight of data from the test to the train set is the probability of this data to be in the train set.
Features that have values with a much higher probability found in a test set than in a train set are drifting features. Drifting features are not important in the model and dropping them sometimes improves prediction.

References

1. Covariate shift
https://www.seldon.io/what-is-covariate-shift/

2. Sberbank Russian Housing Market
https://www.kaggle.com/c/sberbank-russian-housing-market/data

3. Home Credit Default Risk
https://www.kaggle.com/c/home-credit-default-risk/data