Assessment of Models Trained with Synthetic Data

Training models with the help of synthetic data results in classifiers which exhibit an extraordinary high performance as can be seen in chapter Overview and Summary. This measured performance may be due to two reasons.

1. The data synthesis may have generated easy data to classify. More precisely stated, the synthetic data may be easier to classify than Swissbib's original data. When tested, the models do not experience any difficulty with the synthetic data. As their amount is considerably bigger than the amount of Swissbib's original data, the statistics improves.
2. The models trained with synthetic data have seen more pairs of duplicates during their training process. The higher experience of the models results in a better classification ability. This argument would be independent of the ratio of synthetic and pure Swissbib data.

Which of the two explanations is valid and an assessment of the quality of the model fitted with the synthetic data of this capstone project, shall be investigated in this appendix. To do so, the steps are as follows.

1. Synthetic data is generated for training and testing the models.
2. The models to be investigated are fitted with this data. This appendix concentrates on two models, only. The models of the Ensemble family will be chosen.
3. The data generation is redone, in this step with pure Swissbib data in the absence of any artificial synthetic data.
4. The models under consideration are fitted with this pure training data.
5. The performances for all estimators are assessed with the help of the testing data split off from the synthetic data and separately with the help of the testing data split off from the pure Swissbib data.

This process is reflected in the table of contents below.

Table of Contents

Generation of Synthetic Data

The wanted synthetic data can be produced easily calling the first notebooks of the capstone project, reusing code snippets of chapter Overview and Summary.

In [1]:
```
import os
import results_saving_funcs as rsf
import pandas as pd

def get_training_testing_data(nbs, run):
    path_results = './results'
    path_goldstandard = './daten_goldstandard'

    rsf.run_notebooks(nbs, runtime_param_dict, run, path_results)

    # Restore results so far
    df = pd.read_pickle(os.path.join(path_goldstandard, 'labelled_feature_matrix.pkl'), compression=None)
    display(df.head())

    print('Part of duplicates (1) and uniques (0) in units of [%]')
    print(df.duplicates.value_counts())
    print('Part of duplicates (1) and uniques (0) in units of [%]')
    print(round(df.duplicates.value_counts(normalize=True)*100, 2))

    return df
```

In [2]:
```
# Generate dictionary for parameter handover
runtime_param_dict = {
    'em' : 'full' #execution_mode
    , 'os' : 20 # oversampling
    , 'mr' : 0.2 # modification_ratio
    , 'dsn' : 1 # sampling_fraction_nreb
    , 'dsw' : 1 # sampling_fraction_reb
    , 'fa' : 1.0 # factor
    , 'me' : 'added_u' # mode_exactDate
    , 'sn' : True # strip_number_digits
}

# Let's have a look at the predefined parameters
print('Parameters for run : \n', runtime_param_dict)
```

```
Parameters for run :
 {'em': 'full', 'os': 20, 'mr': 0.2, 'dsn': 1, 'dsw': 1, 'fa': 1.0, 'me': 'added_u', 'sn': True}
```

In [3]:
```
# Determine all relevant notebooks, omit Overview Summary and Appendixes
notebook = ! ls [2-5]_* | grep .ipynb

df_labelled_feature_matrix = get_training_testing_data(notebook, 'c0')
```

```
Executing notebook 2_GoldstandardDataPreparation.ipynb
Executing notebook 3_DataSynthesizing.ipynb
Executing notebook 4_FeatureMatrixGeneration.ipynb
Executing notebook 5_FeatureDiscussionDummyBaseline.ipynb
```

	coordinate_E_delta	coordinate_N_delta	corporate_full_delta	doi_delta	edition_delta	exactDate_delta	format_prefix_delta	format_postfix_delta	isbn_delta
0	-1.0	-1.0	-1.0	-1.0	-1.0	0.75	1.0	1.0	1.0
1	-1.0	-1.0	-1.0	-1.0	-1.0	0.75	1.0	1.0	1.0
2	-1.0	-1.0	-1.0	-1.0	-1.0	0.75	1.0	1.0	1.0
3	-1.0	-1.0	-1.0	-1.0	-1.0	0.75	1.0	1.0	1.0
4	-1.0	-1.0	-1.0	-1.0	-1.0	0.75	1.0	1.0	1.0

5 rows × 21 columns

```
Part of duplicates (1) and uniques (0) in units of [%]
0    257955
1     67158
Name: duplicates, dtype: int64
Part of duplicates (1) and uniques (0) in units of [%]
0    79.34
1    20.66
Name: duplicates, dtype: float64
```

Models' Training with Synthetic Data

The wanted data has been produced, above. Now, the two models for assessment can be calculated. Only one estimator will be calculated for each model. The model parameters remain constant for the models due to reproduceability.

Decision Tree Classifier Training

The first relevant model is the Decision Tree Classifier fitted with cross-validation. The process has been copied out of chapter Decision Tree Model and will not be commented.

In [4]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
import classifier_fitting_funcs as cff

def fit_dtcv(df):
    # Training and testing data matrices
    X_tr, _, X_te, y_tr, _, y_te, idx_tr, _, idx_te = cff.split_feature_target(df, 'train_test')

    print(X_tr.shape, y_tr.shape, X_te.shape, y_te.shape)
    print('The test data set holds {:d} records of uniques and {:d} records of duplicates.'.format(
        len(y_te[y_te==0]), len(y_te[y_te==1])))

    # Find best parameters of Decision Tree
    parameters_dtcv = {
        'max_depth' : [20], # Number of features
        'criterion' : ['gini'], # Criterion for best estimator out of tuning in chapter 0
        'class_weight' : ['balanced'] # Class Weight for best estimator out of tuning in chapter 0
    }
    # Grid of values
    grid = cff.generate_parameter_grid(parameters_dtcv)

    # Create cross-validation object with DecisionTreeClassifer
    grid_cv = GridSearchCV(DecisionTreeClassifier(random_state=0),
                           param_grid = parameters_dtcv, cv=5
                           , verbose=1
                          )
    # Fit estimator
    grid_cv.fit(X_tr, y_tr)
    dtcv_best_synt = grid_cv.best_estimator_
    print(dtcv_best_synt)

    # Predict with best estimator
    return dtcv_best_synt, X_te, y_te
```

In [5]:
```
dtcv_synt, X_te_dtcv_synt, y_te_dtcv_synt = fit_dtcv(df_labelled_feature_matrix)
```

```
(260090, 20) (260090,) (65023, 20) (65023,)
The test data set holds 51591 records of uniques and 13432 records of duplicates.
The grid parameters are ...
max_depth [20]
criterion ['gini']
class_weight ['balanced']
 => Number of combinations : 1
Fitting 5 folds for each of 1 candidates, totalling 5 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.0s finished

DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=20, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

Random Forests Classifier Training

The second relevant model for comparison is the Random Forests Classifier, the best classifier in general, cp. Overview and Summary. The process, has been copied out of chapter Decision Tree Model.

In [6]:
```
from sklearn.ensemble import RandomForestClassifier

def fit_rf(df):
    # Training and testing data matrices
    X_tr, X_val, X_te, y_tr, y_val, y_te, idx_tr, id_val, idx_te = cff.split_feature_target(df, 'train_validation_test
')

    print(X_tr.shape, y_tr.shape, X_val.shape, y_val.shape, X_te.shape, y_te.shape)
    print('The test data set holds {:d} records of uniques and {:d} records of duplicates.'.format(
        len(y_te[y_te==0]), len(y_te[y_te==1])))

    parameters_rf = {
        'n_estimators' : 100,
        'max_depth' : 19,
        'class_weight' : None
    }

    # Create a decision tree
    rf_best = RandomForestClassifier(n_estimators=parameters_rf['n_estimators'],
                                     max_depth=parameters_rf['max_depth'],
                                     class_weight=parameters_rf['class_weight'],
                                     random_state=0
                                    )

    # Fit estimator
    rf_best.fit(X_tr, y_tr)
    print(rf_best)

    # Predict with best estimator
    return rf_best, X_te, y_te
```

In [7]:
```
rf_synt, X_te_rf_synt, y_te_rf_synt = fit_rf(df_labelled_feature_matrix)
```

```
(208072, 20) (208072,) (52018, 20) (52018,) (65023, 20) (65023,)
The test data set holds 51591 records of uniques and 13432 records of duplicates.
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=19, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

Generation of Pure Swissbib Data

The pure Swissbib data can be generated calling the first notebooks of the capstone project, the same way as above with different runtime parameters, though.

In [8]:
```
# Generate dictionary for parameter handover
runtime_param_dict = {
    'em' : 'full' #execution_mode
    , 'os' : 0 # oversampling
    , 'mr' : 0.2 # modification_ratio
    , 'dsn' : 1 # sampling_fraction_nreb
    , 'dsw' : 1 # sampling_fraction_reb
    , 'fa' : 1.0 # factor
    , 'me' : 'added_u' # mode_exactDate
    , 'sn' : True # strip_number_digits
}

# Let's have a look at the predefined parameters
print('Parameters for run : \n', runtime_param_dict)
```

```
Parameters for run :
 {'em': 'full', 'os': 0, 'mr': 0.2, 'dsn': 1, 'dsw': 1, 'fa': 1.0, 'me': 'added_u', 'sn': True}
```

In [9]:
```
df_labelled_feature_matrix = get_training_testing_data(notebook, 'c1')
```

```
Executing notebook 2_GoldstandardDataPreparation.ipynb
Executing notebook 3_DataSynthesizing.ipynb
Executing notebook 4_FeatureMatrixGeneration.ipynb
Executing notebook 5_FeatureDiscussionDummyBaseline.ipynb
```

	coordinate_E_delta	coordinate_N_delta	corporate_full_delta	doi_delta	edition_delta	exactDate_delta	format_prefix_delta	format_postfix_delta	isbn_delta
0	-1.0	-1.0	-1.0	-1.0	-1.0	0.75	1.0	1.0	1.0
1	-1.0	-1.0	-1.0	-1.0	-1.0	0.75	1.0	1.0	1.0
2	-1.0	-1.0	-1.0	-1.0	-1.0	0.75	1.0	1.0	1.0
3	-1.0	-1.0	-1.0	-1.0	-1.0	0.75	1.0	1.0	1.0
4	-1.0	-1.0	-1.0	-1.0	-1.0	0.75	1.0	1.0	1.0

5 rows × 21 columns

```
Part of duplicates (1) and uniques (0) in units of [%]
0    257955
1      1473
Name: duplicates, dtype: int64
Part of duplicates (1) and uniques (0) in units of [%]
0    99.43
1     0.57
Name: duplicates, dtype: float64
```

Models' Training with Pure Swissbib Data

With the help of the functions of the last section, the models' predictions are super short in code.

In [10]:
```
dtcv_pure, X_te_dtcv_pure, y_te_dtcv_pure = fit_dtcv(df_labelled_feature_matrix)
rf_pure, X_te_rf_pure, y_te_rf_pure = fit_rf(df_labelled_feature_matrix)
```

```
(207542, 20) (207542,) (51886, 20) (51886,)
The test data set holds 51591 records of uniques and 295 records of duplicates.
The grid parameters are ...
max_depth [20]
criterion ['gini']
class_weight ['balanced']
 => Number of combinations : 1
Fitting 5 folds for each of 1 candidates, totalling 5 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    2.9s finished

DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=20, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
(166033, 20) (166033,) (41509, 20) (41509,) (51886, 20) (51886,)
The test data set holds 51591 records of uniques and 295 records of duplicates.
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=19, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

Performance Measurements

Now, the comparison of the performance can be done with the help of the confusion matrix and its characteristic numbers.

In [11]:
```
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score

def print_score_values(y_te, y_pred):
    print('Area under the curve {:.3f}% - accuracy {:.3f}% - precision {:.3f}% - recall {:.3f}%'.format(
        100*roc_auc_score(y_te, y_pred),
                  100*accuracy_score(y_te, y_pred),
                  100*precision_score(y_te, y_pred),
                  100*recall_score(y_te, y_pred)
                 ))
    return None
```

Decision Tree Classifier Performance

With the following predictions, several combinations can be compared.

In [12]:
```
y_pred_dtcv_synt_synt = dtcv_synt.predict(X_te_dtcv_synt)
y_pred_dtcv_synt_pure = dtcv_synt.predict(X_te_dtcv_pure)
y_pred_dtcv_pure_synt = dtcv_pure.predict(X_te_dtcv_synt)
y_pred_dtcv_pure_pure = dtcv_pure.predict(X_te_dtcv_pure)
```

Pure train data, pure test data

In [14]:
```
from sklearn.metrics import confusion_matrix

print_score_values(y_te_dtcv_pure, y_pred_dtcv_pure_pure)
confusion_matrix(y_te_dtcv_pure, y_pred_dtcv_pure_pure)
```

```
Area under the curve 98.288% - accuracy 99.946% - precision 94.059% - recall 96.610%
```

Out[14]:
```
array([[51573,    18],
       [   10,   285]])
```

This is the original situation and the scoring is the experienced scoring of chapter Overview and Summary.

Synthetic train data, synthetic test data

In [15]:
```
from sklearn.metrics import confusion_matrix

print_score_values(y_te_dtcv_synt, y_pred_dtcv_synt_synt)
confusion_matrix(y_te_dtcv_synt, y_pred_dtcv_synt_synt)
```

```
Area under the curve 99.856% - accuracy 99.898% - precision 99.725% - recall 99.784%
```

Out[15]:
```
array([[51554,    37],
       [   29, 13403]])
```

In chapter Overview and Summary, this combination has been shown as well. Compared with the first constellation as a baseline, this case can be described as in the following list. Some of the statements below may be equivalent.

- The amount of records of pairs of duplicates with a total of 13,432 in the data set for testing, is by two orders of magnitude bigger than the amount of duplicates in the first case with pure test data.
- The accuracy is lower for this combination here, compared to the combination above.
- The score values for precision and recall are significantly higher. They reach a scoring value greater than 99.5% each.
- The ratio of false predictions is lower in this second combination compared to the first one.
- The total of false predictions for this constellation is higher for the case here, than the total of false predictions for the constallation of the situation above.
- The area under the curve is higher here compared to above.
- The number of correctly predicted records of pairs of uniques is lower in this combination compared to the same number above.

Synthetic train data, pure test data

In [16]:
```
print_score_values(y_te_dtcv_pure, y_pred_dtcv_synt_pure)
confusion_matrix(y_te_dtcv_pure, y_pred_dtcv_synt_pure)
```

```
Area under the curve 99.795% - accuracy 99.927% - precision 88.822% - recall 99.661%
```

Out[16]:
```
array([[51554,    37],
       [    1,   294]])
```

This is a new constellation and its result is interesting.

- The amount of records of pairs of duplicates is with a total of 295 exactly the same as in the first constellation. Only two records of duplicates have been wrongly predicted. The model's power for identifying duplicates seems to have increased for models trained with synthetic data.
- The accuracy is at about the same level as for the first case. This observation, together with the first item is equivalent to the observation that more records of uniques have been classified wrongly by this model compared to the first model.
- The score value for recall is significantly better for this case, compared to the first one. It is at about the same level as for the second case, above. The high amount of wrongly predicted uniques is expressed in a significantly lower precision score.
- The ratio of false predictions with a total of 30 is at about the same level as for the first model with a total of 28.
- The area under the curve is higher for this case than for the first combination. It is at about the same value as for the second situation above.
- The number of correctly predicted records of pairs of uniques is exactly the same in this combination as in the case of the model trained with synthetic data and tested with synthetic data.

Summary Training a model with the synthetic data, generated in chapter Data Synthesizing leads to models that can classify duplicates better at the cost of a decreased ability of identifying uniques.

Pure train data, synthetic test data

In [17]:
```
print_score_values(y_te_dtcv_synt, y_pred_dtcv_pure_synt)
confusion_matrix(y_te_dtcv_synt, y_pred_dtcv_pure_synt)
```

```
Area under the curve 85.565% - accuracy 94.016% - precision 99.812% - recall 71.166%
```

Out[17]:
```
array([[51573,    18],
       [ 3873,  9559]])
```

This combination is a countercheck to the second combination above.

- The precision score is the highest for all of the four compared combinations. The root cause of this high precision is due to the high total amount of duplicates that have been identified correctly by the model.
- The total of 18 wrongly predicted uniques is exactly the same in this combination as in the case of the model trained with synthetic data and tested with synthetic data.
- With a total of 3,833, more than one fourth of the records of duplicates has been wrongly classified as uniques. The recall is below 75%.
- The area under the curve is lowest for this constellation.

Summary For models trained with pure Swissbib data, the synthetic data of this capstone project is to a ratio of more than one fourth wrongly identified as uniques. The root cause for this bad performance must be the way of data synthesis. The implementation of chapter Data Synthesizing seems to produce data that resembles more pairs of distinct records for a model trained exclusively with Swissbib's data reality. Swissbib's data reality seems to be badly copied by the implemented synthesis logic.

On the other hand, let's keep in mind the result of the third constellation confirms that models trained with synthetic data. The artificial data seem to generate a better ability for identifying duplicates even for Swissbib's exclusive data reality. So the bad data do have a positive effect on the models when used for training, improving them on identifying duplicates.

Random Forests Classifier Performance

The same comparisons can be done for the Random Forests Classifier. This classifier is the winner of all classifiers compared in chapter Overview and Summary. Therefore, the results below are of big interest.

In [18]:
```
y_pred_rf_synt_synt = rf_synt.predict(X_te_rf_synt)
y_pred_rf_synt_pure = rf_synt.predict(X_te_rf_pure)
y_pred_rf_pure_synt = rf_pure.predict(X_te_rf_synt)
y_pred_rf_pure_pure = rf_pure.predict(X_te_rf_pure)
```

Pure train data, pure test data

In [19]:
```
print_score_values(y_te_rf_pure, y_pred_rf_pure_pure)
confusion_matrix(y_te_rf_pure, y_pred_rf_pure_pure)
```

```
Area under the curve 97.613% - accuracy 99.944% - precision 94.932% - recall 95.254%
```

Out[19]:
```
array([[51576,    15],
       [   14,   281]])
```

The starting point is qualitatively the same for this classifier as for the Decision Tree Classifier.

Synthetic train data, synthetic test data

In [20]:
```
print_score_values(y_te_rf_synt, y_pred_rf_synt_synt)
confusion_matrix(y_te_rf_synt, y_pred_rf_synt_synt)
```

```
Area under the curve 99.886% - accuracy 99.920% - precision 99.784% - recall 99.829%
```

Out[20]:
```
array([[51562,    29],
       [   23, 13409]])
```

For this constellation, it becomes visible that the Random Forests Classifier shows generally a better performance than the Decision Tree Classifier. The area under the curve is unbeatable, all scores show maximum values.

Synthetic train data, pure test data

In [21]:
```
print_score_values(y_te_rf_pure, y_pred_rf_synt_pure)
confusion_matrix(y_te_rf_pure, y_pred_rf_synt_pure)
```

```
Area under the curve 99.124% - accuracy 99.934% - precision 90.909% - recall 98.305%
```

Out[21]:
```
array([[51562,    29],
       [    5,   290]])
```

The statements for the Decision Tree Classifier can be repeated on a quality level for this classifier. The synthetic data improves the estimator's abilities of classifying correctly records of duplicates.

Pure train data, synthetic test data

In [22]:
```
print_score_values(y_te_rf_synt, y_pred_rf_pure_synt)
confusion_matrix(y_te_rf_synt, y_pred_rf_pure_synt)
```

```
Area under the curve 93.188% - accuracy 97.169% - precision 99.871% - recall 86.406%
```

Out[22]:
```
array([[51576,    15],
       [ 1826, 11606]])
```

Interesting about the recall of this constellation is its height which is more than 10 percent points bigger than the recall of the corresponding constellation for the Decision Tree Classifier. The Random Forests Classifier is the best classifier, even when tested with synthetic data.

Summary and Assessment

The results above show that the synthetic data of chapter Data Synthesizing may improve the models slightly when used for training. The synthetic data influence the ability of the models positively in identifying duplicates of Swissbib's pure data reality.

On the other hand, the performance measurement show that the implemented data synthesis does not produce data that reflect Swissbib's data reality very precisely. Therefore, only a part of the scoring measures are influenced positively when training the models with synthetic data of chapter Data Synthesizing.

In [ ]: