31250 Introduction to Data Analytics

Assignment #3

Data Mining in Action

Florian Lubitz
FEIT
13688799
florian.lubitz@student.uts.edu.au

Contents

1 The data mining problem

In the given scenario we have received a data set. This contains data points for various customers of an insurance company. The aim of data mining is to predict whether a customer will purchase an insurance.

The dataset contains multiple features, that have been explored in Assesment 2 of this subject. The list of features contains different type of features. Some of them are dichotomous, some are categorical and some are rational or ordinal features. This wide variety will require some preprocessing to build a good model. Most of the features contain characters as values, one of them contains a date in australian format.

The goal for this problem will be to categorize each datapoint. There are only two categories available: 1 for "will purchase insurance" and 0 for "will not purchase insurance".

2 Preprocessing and transformation

To provide a good starting point for the classifiers I preprocess the raw data. To solve the problem I use python with jupyter notebooks. The following explanations will be supported by short listings. The main sourcecode can be found in the appendix of this report.

2.1 Parsing the data

After importing the data from csv, I start by converting the date into a unix timestamp and all ordinal features into numbers. I also convert dichotomous features that are coded with "Y" and "N" into machine readable "0" and "1", respectively.

```
1 def parse_data(df):
2     # Convert Date
3     df['Original_Quote_Date'] = df['Original_Quote_Date'].apply(
          str_to_timestamp)
4     df['Property_info3'] = df['Property_info3'].apply(string_to_value)
5     # Convert special amount to int
6     df['Field_info3'] = df['Field_info3'].apply(format_amount)
```

```
7     return df
```

Listing 1: An excerpt of the parse function

2.2 Categorical data

After parsing all data I convert categorical features into many flags. This makes it easier for following classifiers to work with these features. To do this I use a function of `pandas` called `get_dummies`. After adding those flags I delete the original feature as it would contain redundant information. The shown function can handle multiple feature conversions at once.

```
1  def categorical_to_many(df, columns, keep_columns=None):
2      # Change Categorical
3      if keep_columns is None:
4          keep_columns = []
5      dummies = dict()
6      for col in columns:
7          dummies[col] = pd.get_dummies(df[col]).add_prefix(col + '_')
8      for dum in dummies:
9          # Keep generated columns as they might include lots of empty(same)
               values
10         keep_columns = keep_columns + list(dummies[dum].keys())
11         df.drop(columns=[dum], inplace=True)
12         df = pd.concat([df, dummies[dum]], axis=1)
13     return df, keep_columns
```

Listing 2: The function to convert one categorical feature into many dichotomous

After converting the features, the training set and and test set could contain different amount of features (flags). To solve this, I populate the sets with all missing flags.

```
1      for key in list(set(train_df.keys()) - set(test_df.keys())):
2          test_df.loc[:, key] = pd.Series(np.zeros(len(test_df['
               Original_Quote_Date'])), index=test_df.index)
3      for key in list(set(test_df.keys()) - set(train_df.keys())):
4          train_df.loc[:, key] = pd.Series(np.zeros(len(train_df['
               Original_Quote_Date'])), index=train_df.index)
```

Listing 3: Add missing flag features to both sets

2.3 Feature Selection

The next preprocessing step is feature selection. I remove different features depending on different reasons. The first removed feature is `Personal_info5` which contains lots of empty values and a low variance. After doing so I delete all rows with empty values inside the training set and fill all empty values inside the test set. I delete emtpy values inside the training set and fill them inside the test set because the empty values inside the training set are more numerous and appear in different features. Inside the test set they only appear in two dichotomous features and less often.

```
    train_df.drop(columns=['Personal_info5'], inplace=True)
    test_df.drop(columns=['Personal_info5'], inplace=True)

    # Remove Rows with empty values
    train_df.dropna(inplace=True)
    # Fill empty values in test dataset, both are YN-Values, replace with
       previous value
    test_df.fillna(method='ffill', inplace=True)
```

Listing 4: Removal of specifc features and emtpy values

Next I remove all features with a variance under 0.16. I remove those because they don't contain enough information to classify the data and only increase the dimensionality of the data set. The feature I remove in the training set, I also remove in the test set.

```
def remove_low_variance(df, keep_columns=None):
    # Remove features with low variance
    if keep_columns is None:
        keep_columns = []
    remove = []
    for col in df:
        if col not in keep_columns:
            var = df.loc[:, col].var()
            # If variance is really low remember for removal
            if var < (.8 * (1 - .8)):
                remove.append(col)
                print('Remove ' + col + ' with variance of ' + str(var))

    # Drop all features with low variance
```

```
15      return df.drop(columns=remove), remove
```

Listing 5: Removal of features with low variance

2.4 Outliers

To remove all outliers inside the training set, I calculate the z-score for most features and delete all rows on the edge of the normal distribution (z-score > 3). I don't alter the test set in this step

2.5 Scaling

To generate a scaled and normalized data set for the classifers to work with, I use the `StandardScaler` of scikit-learn. This will scale the features to unit variance and center them to have a mean of zero. The output of the Standard Scaler will be more gaussian than the input data and provide a better starting point for the classifiers to work with. Some of them will also require scaled data.

I tried out different scaling methods but the combination of these two got the best results for me. This makes sense as the resulting values will be distributed normaly and most classifiers work best with this kind of data.

3 Classification techniques

For the classification process I reviewed multiple classifiers. I ended up optimizing three of them.

After reading and preprocessing the data set, I am splitting the training set int a train and test part. The test part in this split takes up 30 %.

```
1 train_target, train_data, test_data, train_df, test_df = preprocess.
      preprocess(
2     "TrainingSet.csv", 'TestSet.csv', limit=None, remove_low_variance=True
          , remove_outliers=True)
```

```
X_g_train, X_g_test, y_g_train, y_g_test = train_test_split(train_data,
    train_target, test_size=0.30)
```

Listing 6: Preprocessing and spliting the dataset

For each classifier I use the SMOTE technique to oversample our dataset. SMOTE can be used on unbalanced datasets as ours (we less buyers than other). SMOTE is a method to oversample a data set to improve ROC performance NITESH V. CHAWLA.

```
def smote_train_model(classifier_model, x, y):
    # Use SMOTE to oversample the dataset for better training accuracy
    sm = SMOTE()
    x_train_oversampled, y_train_oversampled = sm.fit_sample(x, y)

    # Fit and predict
    classifier_model.fit(x_train_oversampled, y_train_oversampled)
    return classifier_model
```

Listing 7: Function with SMOTE to oversample the dataset

3.1 Parameter optimization

To improve the performance of all classifiers I use `GridSearchCV` to find the optimal parameters for each classifier. The parameters shown in the following sections are all found using this technique. `GridSearchCV` runs the classifier with every possible combination of parameters and can return the parameter with the highest accuracy.

```
model = RandomForestClassifier(n_jobs=-1)
params = {'n_estimators':range(0,200), 'criterion':('gini','entropy')}
gridSearch = GridSearchCV(model, params, cv=5, verbose=2, n_jobs=-1)
gridSearch.fit(X_g_train, y_g_train)
gridSearch.cv_results_['params'][gridSearch.best_index_]
```

Listing 8: Example use of `GridSearchCV` for the random forest classifier

3.2 Random Forest

The random forest is an ensemble classification method. It will build many decision trees and train them with a subset of the supplied data. After running those trees it will collect votes from all trees to classify a data point.

The selected parameters set the number of trees inside the forest to 185. This number gave optimal results without overfitting. Overfitting with too many trees can have an impact on the performance of the random forest. The selected criterion "entropy" uses information gain to build the decision trees.

```
1    print('Start Random Forest')
2    model = RandomForestClassifier(n_estimators=185, criterion='entropy',
        n_jobs=-1)
3    model = smote_train_model(model, X_g_train, y_g_train)
4    y_predict = model.predict(X_g_test)
```

Listing 9: `RandomForestClassifier` used for classifing

3.3 Support Vector Machines

A SVM tries to find a hyperplane in a n-dimensional room, where n is the number of features provided. The hyperplane should distinctly classify all data points. It will look for the hyperplane with the biggest distance between points of each class.

```
1
2 rf_df = decide_for_unsure(result_df, 'TestSet')
3 rf2_df = decide_for_unsure(t_df, 'TrainSet')
```

Listing 10: `SVC` used for classifing

3.4 Multilayer Perceptron

Neural Networks with multiple fully connected single neurons are called Multilayer Perceptron Networks (MLP networks). They are feedforward artifical neural networks and uses backpropagation to learn in each iteration of its training. The selected network for this data mining problem consists of 4 layers. The two hidden ones are specified inside the parameters of the constructor.

```
y_predict = model.predict(X_g_test)
print(roc_auc_score(y_g_test, y_predict))

result_predict['SVM'] = np.array(model.predict(test_data))
test_predict['SVM'] = np.array(model.predict(X_g_test))
```

Listing 11: `MLPClassifier` used for classifing

4 The best classifier

For the given dataset and problem the random forest achieves the best results for me. I therefore selected the random forest as the best classifier. The problem is a unbalanced data set and is not easily linearly solvable. Therefore the Vector machine has its problems with this type of data set. The MLP is not able to reach the performance of the random forest in my case.

4.1 Classifer type

As described above, the random forest belongs to the ensemble classifiers. It obtains its result from a large number of decision trees and counts the votes together to make the classification. This increases the accuracy in comparison to a single tree and also overfitting becomes a minor problem. It is also true that with increased variation the accuracy increases to a certain degree.

4.2 Performance

4.3 Solving the problem

Listings

Bibliography

Nitesh V. Chawla

Nitesh V. Chawla, Lawrence O. Hall W. Philip K. Kevin W. Bowyer B. Kevin W. Bowyer: SMOTE: Synthetic Minority Over-sampling Technique.

A Appendix

A.1 List of all Features

Quote_ID, Original_Quote_Date, QuoteConversion_Flag, Field_info1, Field_info2, Field_info3, Field_info4, Coverage_info1, Coverage_info2, Coverage_info3, Sales_info1, Sales_info2, Sales_info3, Sales_info4, Sales_info5, Personal_info1, Personal_info2, Personal_info3, Personal_info4, Personal_info5, Property_info1, Property_info2, Property_info3, Property_info4, Property_info5, Geographic_info1, Geographic_info2, Geographic_info3, Geographic_info4, Geographic_info5.

A.2 Source code

main.py

```
1  import preprocess
2  import pandas as pd
3  import numpy as np
4  from sklearn.model_selection import train_test_split
5  from sklearn.ensemble import RandomForestClassifier
6  from sklearn.neural_network import MLPClassifier
7  from sklearn.svm import SVC
8  from imblearn.over_sampling import SMOTE
9  from sklearn.metrics import roc_auc_score
10
11
12 def smote_train_model(classifier_model, x, y):
13     # Use SMOTE to oversample the dataset for better training accuracy
14     sm = SMOTE()
15     x_train_oversampled, y_train_oversampled = sm.fit_sample(x, y)
16
17     # Fit and predict
18     classifier_model.fit(x_train_oversampled, y_train_oversampled)
19     return classifier_model
20
21
22 def decide_for_unsure(df, name):
```

```
23      df['Sum'] = df.sum(axis=1)
24      final = []
25      width = len(df.keys()) - 1
26      print(df.keys())
27      count = []
28      # init counter
29      for i in range(0, width + 1):
30          count.append(0)
31      for row in df['Sum']:
32          if 0 < row < width:
33              # Do +1 as most of the data points tend to be 0 rather than 1
34              final.append(1 if row > ((width / 2) + 1) else 0)
35              count[row] += 1
36          else:
37              count[row] += 1
38              final.append(0 if row == 0 else 1)
39      df['Final'] = final
40      for i in range(0, width + 1):
41          print(str(count[i]) + ' times ' + str(i) + ' on ' + name)
42      print(str(sum(count[1:-1]) / len(df) * 100) + '% Unsure on ' + name)
43      return df
44
45
46  def save_to_file(df, prediction, suffix):
47      df['QuoteConversion_Flag'] = pd.Series(prediction, index=df.index)
48
49      toDrop = []
50      for col in df.columns:
51          if col not in ['Quote_ID', 'QuoteConversion_Flag']:
52              toDrop.append(col)
53      df.drop(columns=toDrop, inplace=True)
54      df.to_csv(f'./results/Kaggle_Submission{suffix}.csv', index=False)
55      print('Written to file')
56
57
58  # Get Preprocessed Data
```

```
59 train_target, train_data, test_data, train_df, test_df = preprocess.
       preprocess(
60     "TrainingSet.csv", 'TestSet.csv', limit=None, remove_low_variance=True
           , remove_outliers=True)
61 X_g_train, X_g_test, y_g_train, y_g_test = train_test_split(train_data,
       train_target, test_size=0.30)
62 print(f'TrainSet has {train_target.sum()} times 1')
63
64 # Init some variables for later use
65 result_predict = dict()
66 test_predict = dict()
67
68 # Random Forest
69 for i in range(0,20):
70     print('Start Random Forest')
71     model = RandomForestClassifier(n_estimators=185, criterion='entropy',
           n_jobs=-1)
72     model = smote_train_model(model, X_g_train, y_g_train)
73     y_predict = model.predict(X_g_test)
74     print(roc_auc_score(y_g_test, y_predict))
75
76     result_predict['RandomForest_' + str(i)] = np.array(model.predict(
           test_data))
77     test_predict['RandomForest_' + str(i)] = np.array(model.predict(
           X_g_test))
78
79 result_df = pd.DataFrame(result_predict)
80 t_df = pd.DataFrame(test_predict)
81
82 rf_df = decide_for_unsure(result_df, 'TestSet')
83 rf2_df = decide_for_unsure(t_df, 'TrainSet')
84
85 print(roc_auc_score(y_g_test, rf2_df['Final']))
86 save_to_file(test_df, rf_df['Final'], '_rf')
87
88 # SVM
89 print('SVM')
```

```
90  model = SVC(gamma='auto', kernel='rbf')
91  model = smote_train_model(model, X_g_train, y_g_train)
92  y_predict = model.predict(X_g_test)
93  print(roc_auc_score(y_g_test, y_predict))
94
95  result_predict['SVM'] = np.array(model.predict(test_data))
96  test_predict['SVM'] = np.array(model.predict(X_g_test))
97  save_to_file(test_df, result_predict['SVM'], '_svm')
98
99  # Neural Network
100 print('Start MLPClassifier')
101 model = MLPClassifier(solver='adam', alpha=0.0001, learning_rate_init
        =0.001,
102                       hidden_layer_sizes=(17, 11), max_iter=1000,
                              warm_start=True)
103 model = smote_train_model(model, X_g_train, y_g_train)
104 y_predict = model.predict(X_g_test)
105 print(roc_auc_score(y_g_test, y_predict))
106
107 result_predict['mlpNetwork'] = np.array(model.predict(test_data))
108 test_predict['mlpNetwork'] = np.array(model.predict(X_g_test))
109 save_to_file(test_df, result_predict['mlpNetwork'], '_mlp')
110
111 # Do stuff with unsure rows
112
113 result_df = pd.DataFrame(result_predict)
114 result_df = decide_for_unsure(result_df, 'TestSet')
115
116 print('\n Test:')
117 # Do it for test to
118 t_df = pd.DataFrame(test_predict)
119 t_df = decide_for_unsure(t_df, 'TrainingSet')
120 print(roc_auc_score(y_g_test, t_df['Final']))
121
122 # Save Result to file
123 save_to_file(test_df, result_df['Final'], '_all')
```

Listing 12: main.py

preprocess.py

```
1  import numpy as np
2  import pandas as pd
3  from sklearn.preprocessing import StandardScaler
4  from sklearn.preprocessing import normalize
5  import helpers
6
7  pd.set_option('display.width', 0)
8
9
10 def preprocess(train_file, test_file, limit=None, remove_low_variance=
       True, remove_outliers=True):
11     train_df = pd.read_csv(train_file)
12     test_df = pd.read_csv(test_file)
13
14     if limit is None:
15         limit = len(train_df)
16     if 0 < limit < len(train_df):
17         print('Limited Sample: ' + str(limit))
18         train_df = train_df.sample(n=limit)
19
20     train_df = helpers.parse_data(train_df)
21     test_df = helpers.parse_data(test_df)
22
23     keepColumns = ['QuoteConversion_Flag']
24
25     train_df, keepColumns = helpers.categorical_to_many(train_df, ['
           Geographic_info5'], keepColumns)
26     test_df,a = helpers.categorical_to_many(test_df, ['Geographic_info5'],
            keepColumns)
27
28     # Fill up train and test frame to have the same column length
29     for key in list(set(train_df.keys()) - set(test_df.keys())):
30         test_df.loc[:, key] = pd.Series(np.zeros(len(test_df['
               Original_Quote_Date'])), index=test_df.index)
31     for key in list(set(test_df.keys()) - set(train_df.keys())):
```

```
32          train_df.loc[:, key] = pd.Series(np.zeros(len(train_df['
                Original_Quote_Date'])), index=train_df.index)
33
34      # Feature Pre-Selection
35
36      # Drop Personal_info5, it has lot of empty values
37      train_df.drop(columns=['Personal_info5'], inplace=True)
38      test_df.drop(columns=['Personal_info5'], inplace=True)
39
40      # Remove Rows with empty values
41      train_df.dropna(inplace=True)
42      # Fill empty values in test dataset, both are YN-Values, replace with
            previous value
43      test_df.fillna(method='ffill', inplace=True)
44
45      if remove_low_variance:
46          train_df, removed_columns = helpers.remove_low_variance(train_df,
                keepColumns)
47          test_df.drop(columns=removed_columns, inplace=True)
48
49      print('DataFrame shape after feature selection:' + str(train_df.shape)
            )
50
51      # Detect and Remove outliers
52      if remove_outliers:
53          train_df = helpers.remove_outliers(train_df)
54
55      print('DataFrame shape after outlier removal:' + str(train_df.shape))
56
57      # Extract dependent variable from dataset
58      train_dv = np.array(train_df['QuoteConversion_Flag'])
59      train_data = np.array(train_df.drop(columns=['QuoteConversion_Flag', '
            Quote_ID']))
60      test_data = np.array(test_df.drop(columns=['QuoteConversion_Flag','
            Quote_ID']))
61
62
```

```
63      # Scale things
64      standard_scaler = StandardScaler()
65      standard_scaler.fit(train_data)
66      train_data = standard_scaler.transform(train_data)
67      test_data = standard_scaler.transform(test_data)
68
69      # Normalize it to be more gaussian
70      train_data = normalize(train_data, return_norm=False, axis=0)
71      test_data = normalize(test_data, return_norm=False, axis=0)
72
73      return train_dv, train_data, test_data, train_df, test_df
```

Listing 13: preprocess.py

helpers.py

```
1  from datetime import datetime
2  import pandas as pd
3  import numpy as np
4  from scipy import stats
5
6
7  def string_to_bool(string):
8      if string == 'Y':
9          return 1.0
10     if string == 'N':
11         return 0.0
12     # If its anything else return none
13     return None
14
15
16 def string_to_value(string):
17     if string == '':
18         return None
19     if len(string) == 1:
20         return ord(string) - 64
21     value = 0
22     for index, char in enumerate(string):
```

```
23          value += (ord(char) - 64) * pow(10, index)
24      return value
25
26
27  def format_amount(string):
28      return int(string.replace(',', ''))
29
30
31  def str_to_timestamp(date_string):
32      return int(datetime.strptime(date_string.zfill(10), '%d/%m/%Y').
            timestamp())
33
34
35  def minus_to_none(value):
36      if value == -1:
37          return None
38      return value
39
40
41  def parse_data(df):
42      # Convert Date
43      df['Original_Quote_Date'] = df['Original_Quote_Date'].apply(
            str_to_timestamp)
44      # Convert bool-values to int of 1 and 0
45      df['Field_info4'] = df['Field_info4'].apply(string_to_bool)
46      df['Personal_info1'] = df['Personal_info1'].apply(string_to_bool)
47      df['Property_info1'] = df['Property_info1'].apply(string_to_bool)
48      df['Geographic_info4'] = df['Geographic_info4'].apply(string_to_bool)
49      # Convert string to int values
50      df['Field_info1'] = df['Field_info1'].apply(string_to_value)
51      df['Coverage_info3'] = df['Coverage_info3'].apply(string_to_value)
52      df['Sales_info4'] = df['Sales_info4'].apply(string_to_value)
53      df['Personal_info3'] = df['Personal_info3'].apply(string_to_value)
54      df['Property_info3'] = df['Property_info3'].apply(string_to_value)
55      # Convert special amount to int
56      df['Field_info3'] = df['Field_info3'].apply(format_amount)
57      return df
```

```
58
59
60 def categorical_to_many(df, columns, keep_columns=None):
61     # Change Categorical
62     if keep_columns is None:
63         keep_columns = []
64     dummies = dict()
65     for col in columns:
66         dummies[col] = pd.get_dummies(df[col]).add_prefix(col + '_')
67     for dum in dummies:
68         # Keep generated columns as they might include lots of empty(same)
               values
69         keep_columns = keep_columns + list(dummies[dum].keys())
70         df.drop(columns=[dum], inplace=True)
71         df = pd.concat([df, dummies[dum]], axis=1)
72     return df, keep_columns
73
74
75 def remove_low_variance(df, keep_columns=None):
76     # Remove features with low variance
77     if keep_columns is None:
78         keep_columns = []
79     remove = []
80     for col in df:
81         if col not in keep_columns:
82             var = df.loc[:, col].var()
83             # If variance is really low remember for removal
84             if var < (.8 * (1 - .8)):
85                 remove.append(col)
86                 print('Remove ' + col + ' with variance of ' + str(var))
87
88     # Drop all features with low variance
89     return df.drop(columns=remove), remove
90
91
92 def remove_outliers(df):
93     # Calculate z-score and store in numpy-array
```

```
94      z = np.abs(stats.zscore(df))
95      # Remove axis where z-score is lower than 3 (p=0.0013 of being in
            normal distribution)
96      return df[(z < 3).all(axis=1)]
```

Listing 14: helpers.py