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     # Convert bool-values to int of 1 and 0
5     df['Field_info4'] = df['Field_info4'].apply(string_to_bool)
6     df['Property_info3'] = df['Property_info3'].apply(string_to_value)
7     # Convert special amount to int
```

```
8     df['Field_info3'] = df['Field_info3'].apply(format_amount)
9     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      # Fill up train and test frame to have the same column length
2      for key in list(set(train_df.keys()) - set(test_df.keys())):
3          test_df.loc[:, key] = pd.Series(np.zeros(len(test_df['
               Original_Quote_Date'])), index=test_df.index)
4      for key in list(set(test_df.keys()) - set(train_df.keys())):
```

```
5        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.

```
1    # Drop Personal_info5, it has lot of empty values
2    train_df.drop(columns=['Personal_info5'], inplace=True)
3    test_df.drop(columns=['Personal_info5'], inplace=True)
4    # Remove Rows with empty values
5    train_df.dropna(inplace=True)
6    # Fill empty values in test dataset, both are YN-Values, replace with
         previous value
7    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.

```
1  def remove_low_variance(df, keep_columns=None):
2      # Remove features with low variance
3      if keep_columns is None:
4          keep_columns = []
5      remove = []
6      for col in df:
7          if col not in keep_columns:
8              var = df.loc[:, col].var()
9              # If variance is really low remember for removal
```

```
10            if var < (.8 * (1 - .8)):
11                remove.append(col)
12                print('Remove ' + col + ' with variance of ' + str(var))
13
14    # 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 %.

```
train_target, train_data, test_data, train_df, test_df = preprocess.preprocess(
    "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.

```
model = RandomForestClassifier(n_estimators=98, criterion='entropy',
    n_jobs=-1)
model = smote_train_model(model, X_g_train, y_g_train)
y_predict = model.predict(X_g_test)
print(roc_auc_score(y_g_test, y_predict))
```

Listing 9: `RandomForestClassifier` used for classifing

3.3 Support Vector Machines

```
model = SVC(gamma='auto', kernel='rbf')
model = smote_train_model(model, X_g_train, y_g_train)
y_predict = model.predict(X_g_test)
print(roc_auc_score(y_g_test, y_predict))
```

Listing 10: `SVC` used for classifing

3.4 Multilayer Perceptron

```
model = MLPClassifier(solver='adam', alpha=0.0001, learning_rate_init
    =0.001,
                      hidden_layer_sizes=(17, 11), max_iter=1000,
                          warm_start=True)
model = smote_train_model(model, X_g_train, y_g_train)
y_predict = model.predict(X_g_test)
print(roc_auc_score(y_g_test, y_predict))
```

Listing 11: `MLPClassifier` used for classifing

4 The best classifier

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.

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