[redacted]

Alessio Miolla

June 19, 2021

1 Problem description

Q3: Modelling Assessment:

For this task use the Bank Marketing dataset available on the following address: http://archive.ics.uci.edu/ml/datasets/Bank+Marketing

Build 2 binary classification models using any 2 of the following methods (in R or Python): 1. Logistic Regression 2. Random Forest 3. GBM 4. Xgboost 5. Neural Network

Try and minimalize overfitting. Compare the performance of both models using ROC graphs, AUCs, confusion matrices. Provide also the full source code and description of any variable transformations or balancing performed.

For the current task I used the bank-additional.csv with 10% of the examples (4119). Logistic Regression and Random Forrest were used as ML models.

2 Import

[22]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪StratifiedKFold, RepeatedStratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
import category_encoders as ce
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report
import scikitplot as skplt
from sklearn import datasets, metrics, model_selection, svm
from sklearn.metrics import auc, roc_curve, roc_auc_score,␣
 ↪precision_recall_curve, precision_score, f1_score, recall_score,␣
 ↪confusion_matrix, accuracy_score
```

3 Load

[4]:
```
path_data = 'C:/Users/alessio/Desktop/[redacted]/'
df = pd.read_csv(path_data + 'bank-additional.csv', sep=';')
```

[5]:
```
df.head()
```

[5]:
```
   age          job  marital          education default  housing     loan  \
0   30  blue-collar  married           basic.9y      no      yes       no
1   39     services   single        high.school      no       no       no
2   25     services  married        high.school      no      yes       no
3   38     services  married           basic.9y      no  unknown  unknown
4   47       admin.  married  university.degree      no      yes       no

     contact month day_of_week  ...  campaign  pdays  previous     poutcome  \
0   cellular   may         fri  ...         2    999         0  nonexistent
1  telephone   may         fri  ...         4    999         0  nonexistent
2  telephone   jun         wed  ...         1    999         0  nonexistent
3  telephone   jun         fri  ...         3    999         0  nonexistent
4   cellular   nov         mon  ...         1    999         0  nonexistent

  emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y
0         -1.8          92.893          -46.2      1.313       5099.1  no
1          1.1          93.994          -36.4      4.855       5191.0  no
2          1.4          94.465          -41.8      4.962       5228.1  no
3          1.4          94.465          -41.8      4.959       5228.1  no
4         -0.1          93.200          -42.0      4.191       5195.8  no

[5 rows x 21 columns]
```

4 Exploratory analysis

[20]:
```
df.shape #check shape
```

[20]:
```
(4119, 21)
```

[24]:
```
df['y'].value_counts() #check proportion of the target classes
```

[24]:
```
no     3668
yes     451
Name: y, dtype: int64
```

[39]:
```
#check value counts for each feature
for x in df.columns:
    print(df[x].value_counts())
```

[71]:
```
#check dtype for each feature
for x in df.columns:
    print(df[x].dtype)
```

5 Conversion of Categorical variables

Binary transformation The following variables can be converted into dichotomic variables to optimize the learning of the models and optimize the value of the variables.

[6]:
```
#convert categorical in numerical values of the target variable
df['y'] = df['y'].map({'no':0, 'yes':1}).astype('uint8')
```

[7]:
```
#convert binary categorical variables into numerical
df['contact'] = df['contact'].map({'cellular': 1, 'telephone': 0}).
 ↪astype('uint8')
df['loan'] = df['loan'].map({'yes': 1, 'unknown': 0, 'no' : 0}).astype('uint8')
df['housing'] = df['housing'].map({'yes': 1, 'unknown': 0, 'no' : 0}).
 ↪astype('uint8')
df['default'] = df['default'].map({'no': 1, 'unknown': 0, 'yes': 0}).
 ↪astype('uint8')
```

[9]:
```
#convert previous col into binary col (0 if not contacted, 1 if contacted)
df['previous'] = df['previous'].apply(lambda x: 1 if x > 0 else 0).
 ↪astype('uint8')
```

[10]:
```
#convert poutcome col into binary (1 if successed, 0 if not)
df['poutcome'] = df['poutcome'].map({'nonexistent':0, 'failure':0, 'success':1}).
 ↪astype('uint8')
```

[8]:
```
#convert 999 pdays in 0
df['pdays'] = df['pdays'].replace(999, 0)
```

OHE encoding One Hot encoding was used to convert the following categorical variables into dummy variables.

[11]:
```
#convert job, week, month into one hot encoding

# function to One Hot Encoding
def encode(data, col):
    return pd.concat([data, pd.get_dummies(col, prefix=col.name)], axis=1)

df = encode(df, df.job)
df = encode(df, df.month)
df = encode(df, df.day_of_week)

df.drop(['job', 'month', 'day_of_week'], axis=1, inplace=True)
```

```
df.drop_duplicates(inplace=True)
```

Ordinal encoding The education level was converted into ordinal encoding to rank the different level of educations

[12]:
```
#convert categorical into ordinal encoding
df['education'] = df['education'].map({'illiterate':0, 'unknown':0, 'basic.4y':
↪1, 'basic.6y':1, 'professional.course':2, 'basic.9y':2,
                                          'high.school':3, 'university.degree':4 }).
↪astype('uint8')
```

Target encoding There is no ordinal relationship between the marital metrics (e.g., single or divorced). In addition, all the metrics are homogenous within the sample. A target encoding was used to optimize the encoding of the variable.

[13]:
```
# target variable for testing
y = df.y
# convert marital variable into target encoding --> final training_set
target_encode = ce.target_encoder.TargetEncoder(cols=('marital')).fit(df, y)
training_set = target_encode.transform(df)
# drop target variable to split features from target variable
training_set.drop('y', axis=1, inplace=True)
```

```
C:\Users\alessio\anaconda3\envs\wargaming\lib\site-
packages\category_encoders\utils.py:21: FutureWarning: is_categorical is
deprecated and will be removed in a future version.  Use is_categorical_dtype
instead
  elif pd.api.types.is_categorical(cols):
```

[14]:
```
training_set.head()
```

[14]:
```
   age   marital  education  default  housing  loan  contact  duration  \
0   30  0.100438          2        1        1     0        1       487
1   39  0.134432          3        1        0     0        0       346
2   25  0.100438          3        1        1     0        0       227
3   38  0.100438          2        1        0     0        0        17
4   47  0.100438          4        1        1     0        1        58

   campaign  pdays  ...  month_mar  month_may  month_nov  month_oct  \
0         2      0  ...          0          1          0          0
1         4      0  ...          0          1          0          0
2         1      0  ...          0          0          0          0
3         3      0  ...          0          0          0          0
4         1      0  ...          0          0          1          0

   month_sep  day_of_week_fri  day_of_week_mon  day_of_week_thu  \
```

```
0          0                1                0                0
1          0                1                0                0
2          0                0                0                0
3          0                1                0                0
4          0                0                1                0

   day_of_week_tue  day_of_week_wed
0                0                0
1                0                0
2                0                1
3                0                0
4                0                0

[5 rows x 44 columns]
```

6 ML models

The machine learning models used in the current are Logistic Regression and Random Forrest, respectively. F1 score was adopted as the main criteria since the unbalanced between the two target variables. Indeed, the highly skewed class distribution (89% of class0 versus 11% of class1) can lead the classifier to get a low misclassification rate simply by choosing the majority class (i.e., class 0). Consequently, I decided to get the classifiers with high F1 scores in both classes instead of other criteria such as accuracy, precision, and recall of the majority class0.

6.1 Basic approach

Logistic Regression

[150]:
```
#split training and test set
X_train, X_test, y_train, y_test = train_test_split(training_set, y, test_size=0.
 ↪2, random_state=42)
Counter(y_test)
```

[150]:
```
Counter({1: 92, 0: 732})
```

[151]:
```
sc = StandardScaler()
X_train_transformed = sc.fit_transform(X_train)
X_test_transformed = sc.transform(X_test)
```

[153]:
```
#LR
model = LogisticRegression()

#Tuning and finding the best hyperparams
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [0.3, 0.6, 0.7]
```

```
#grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
grid_search_logistic = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,␣
↪cv=5, scoring='f1',error_score=0)

#training
clf = grid_search_logistic.fit(X_train_transformed, y_train)
y_pred_train = clf.predict(X_train_transformed)
roc=roc_auc_score(y_train, y_pred_train)
print('roc', roc)
print('prediction_matrix', '\n', confusion_matrix(y_train, y_pred_train))
print(classification_report(y_train, y_pred_train))
```

```
roc 0.7324928085128991
prediction_matrix
 [[2870   66]
 [ 184  175]]
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      2936
           1       0.73      0.49      0.58       359

    accuracy                           0.92      3295
   macro avg       0.83      0.73      0.77      3295
weighted avg       0.92      0.92      0.92      3295
```

[89]:

```
#test
y_test_pred = clf.predict(X_test_transformed)

#performance
roc=roc_auc_score(y_test, y_test_pred)
print('roc', roc)
print('prediction_matrix', '\n', confusion_matrix(y_test, y_test_pred))
print(classification_report(y_test, y_test_pred))
```

```
roc 0.6772095509622238
prediction_matrix
 [[705  27]
 [ 56  36]]
              precision    recall  f1-score   support

           0       0.93      0.96      0.94       732
           1       0.57      0.39      0.46        92

    accuracy                           0.90       824
   macro avg       0.75      0.68      0.70       824
```

```
weighted avg       0.89      0.90      0.89       824
```

The test results show slight overfitting of the model with a global F1 score of 0.7 and AUC of 0.68. The minority class negatively affects the overall performance of the model as highlighted in its precision and recall.

Random Forrest

[156]:
```
#split training and test set
X_train, X_test, y_train, y_test = train_test_split(training_set, y, test_size=0.
 ↪2, random_state=42)
Counter(y_test)
```

[156]:
```
Counter({1: 92, 0: 732})
```

[207]:
```
#RF
random_forest_clf = RandomForestClassifier()

#Tuning and finding the best hyperparams

param_grid = {
    'max_leaf_nodes' : [10, 20, 30],
    'max_depth': [10, 20, 30],
}

#grid search
grid_search_rf = GridSearchCV(estimator = random_forest_clf, param_grid =␣
 ↪param_grid, cv=5, scoring='f1', verbose=0, n_jobs=-1)

#training
clf_rf = grid_search_rf.fit(X_train, y_train)
y_pred_train = clf_rf.predict(X_train)
roc=roc_auc_score(y_train, y_pred_train)
print('prediction_matrix', '\n', confusion_matrix(y_train, y_pred_train))
print('roc', roc)
print(classification_report(y_train, y_pred_train))
```

```
prediction_matrix
 [[2931    5]
 [ 249  110]]
roc 0.6523518439807822
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      2936
           1       0.96      0.31      0.46       359

    accuracy                           0.92      3295
   macro avg       0.94      0.65      0.71      3295
```

```
weighted avg       0.93      0.92      0.90      3295
```

[208]:
```
#test
y_pred = clf_rf.predict(X_test)

#performance
roc=roc_auc_score(y_test, y_pred)
print('roc', roc)
print('prediction_matrix', '\n', confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
roc 0.5930446661914944
prediction_matrix
 [[725   7]
 [ 74  18]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       732
           1       0.72      0.20      0.31        92

    accuracy                           0.90       824
   macro avg       0.81      0.59      0.63       824
weighted avg       0.89      0.90      0.88       824
```

RF model includes different limitations. First of all, there is slight overfitting of the training set that affects the test performance. Moreover, the model is severely biased by the majority class, as showed either by the low number of predictions of class1 and the higher precision/recall of the majority class. More restrictions should be adopted in the model to prevent overfitting, with the risk of decreasing the performance of the model or, conversely, causing underfitting of the model.

6.2 Oversampling

The imbalance between the two target classes led to having poor performance in the minority class. In the current section, the oversampling of the minority class was used to addressing this problem. The Synthetic Minority Oversampling Technique (SMOTE) is a type of data augmentation that duplicates examples in the minority class, although these examples don't add any new information to the model. Doing so, I tested if the previous low predictions on class1 are not due to the relatively little amount of information given in the training set.

Logistic Regression

[239]:
```
# summarize class distribution
counter = Counter(y)
print(counter)
#split_training and test set
X_train, X_test, y_train, y_test = train_test_split(training_set, y, test_size=0.
 ↪2, random_state=42)
```

```
#compromise oversampling
over = SMOTE(sampling_strategy=0.5)
X_train, y_train = over.fit_resample(X_train, y_train)
# summarize the new class distribution
counter = Counter(y_train)
print(counter)
```

```
Counter({0: 3668, 1: 451})
Counter({0: 2936, 1: 1468})
```

[240]:

```
sc = StandardScaler()
X_train_transformed = sc.fit_transform(X_train)
X_test_transformed = sc.transform(X_test)
```

[241]:

```
#LR
model = LogisticRegression()

#Tuning and finding the best hyperparams
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]

#grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
grid_search_logistic = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,␣
 ↪cv=5, scoring='f1',error_score=0)

#training
clf = grid_search_logistic.fit(X_train_transformed, y_train)
y_pred_train = clf.predict(X_train_transformed)
roc=roc_auc_score(y_train, y_pred_train)
print('roc', roc)
print('prediction_matrix', '\n', confusion_matrix(y_train, y_pred_train))
print(classification_report(y_train, y_pred_train))
```

```
roc 0.9189373297002724
prediction_matrix
 [[2846   90]
 [ 193 1275]]
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      2936
           1       0.93      0.87      0.90      1468

    accuracy                           0.94      4404
   macro avg       0.94      0.92      0.93      4404
weighted avg       0.94      0.94      0.94      4404
```

[242]:
```python
#test
y_test_pred = clf.predict(X_test_transformed)

#performance
roc=roc_auc_score(y_test, y_test_pred)
print('roc', roc)
print('prediction_matrix', '\n', confusion_matrix(y_test, y_test_pred))
print(classification_report(y_test, y_test_pred))
```

```
roc 0.6996020432406747
prediction_matrix
 [[698  34]
 [ 51  41]]
              precision    recall  f1-score   support

           0       0.93      0.95      0.94       732
           1       0.55      0.45      0.49        92

    accuracy                           0.90       824
   macro avg       0.74      0.70      0.72       824
weighted avg       0.89      0.90      0.89       824
```

The oversampling clearly increases the overfitting of the model. The F1 score and the AUC dropped more than 20% in testing. Additionally, the performance obtained by oversampling the minority class doesn't show significant benefits in testing performance than the first approach without oversampling.

Random Forrest

[246]:
```python
# summarize class distribution
counter = Counter(y)
print(counter)
#split_training and test set
X_train, X_test, y_train, y_test = train_test_split(training_set, y, test_size=0.
 ↪2, random_state=42)
#compromise oversampling
over = SMOTE(sampling_strategy=0.5)
X_train, y_train = over.fit_resample(X_train, y_train)
# summarize the new class distribution
counter = Counter(y_train)
print(counter)
```

```
Counter({0: 3668, 1: 451})
Counter({0: 2936, 1: 1468})
```

[247]:
```
#RF
random_forest_clf = RandomForestClassifier()

#Tuning and finding the best hyperparams
param_grid = {
    'max_leaf_nodes' : [10, 20, 30],
    'max_depth': [10, 20, 30],
}

#grid search
grid_search_rf = GridSearchCV(estimator = random_forest_clf, param_grid =␣
 ↪param_grid, cv=5, scoring='f1', verbose=0, n_jobs=-1)

#training
clf_rf = grid_search_rf.fit(X_train, y_train)
y_pred_train = clf_rf.predict(X_train)
roc=roc_auc_score(y_train, y_pred_train)
print('roc', roc)
print(classification_report(y_train, y_pred_train))
```

```
roc 0.9056539509536784
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      2936
           1       0.90      0.86      0.88      1468

    accuracy                           0.92      4404
   macro avg       0.91      0.91      0.91      4404
weighted avg       0.92      0.92      0.92      4404
```

[248]:
```
#test
y_test_pred = clf_rf.predict(X_test)

#performance
roc=roc_auc_score(y_test, y_test_pred)
print('roc', roc)
print('prediction_matrix', '\n', confusion_matrix(y_test, y_test_pred))
print(classification_report(y_test, y_test_pred))
```

```
roc 0.7389819434545023
prediction_matrix
 [[692  40]
 [ 43  49]]
              precision    recall  f1-score   support

           0       0.94      0.95      0.94       732
```

```
           1       0.55      0.53      0.54        92

    accuracy                           0.90       824
   macro avg       0.75      0.74      0.74       824
weighted avg       0.90      0.90      0.90       824
```

The RF with oversampling was revealed the best model so far. Despite the overfitting of training, the precision and recall of the minority class increase compared to the previous approach, bringing a global F1 score and AUC of 0.74. In the next section, the threshold model will be adjusted to improve the performance of the minority class.

7 Performance

In the current section the best model obtained in the previous approaches was used, namely the RF with oversampling.

[255]:

```
# summarize class distribution
counter = Counter(y)
print(counter)
#split_training and test set
X_train, X_test, y_train, y_test = train_test_split(training_set, y, test_size=0.
 ↪2, random_state=42)
#compromise oversampling
over = SMOTE()
X_train, y_train = over.fit_resample(X_train, y_train)
# summarize the new class distribution
counter = Counter(y_train)
print(counter)
```

```
Counter({0: 3668, 1: 451})
Counter({0: 2936, 1: 2936})
```

[261]:

```
#RF
random_forest_clf = RandomForestClassifier()

#Tuning and finding the best hyperparams
param_grid = {
    'max_leaf_nodes' : [10, 20, 30],
    'max_depth': [10, 20, 30],
}

#grid search
grid_search_rf = GridSearchCV(estimator = random_forest_clf, param_grid =␣
 ↪param_grid, cv=5, scoring='f1', verbose=0, n_jobs=-1)

#training
```

```
clf = grid_search_rf.fit(X_train, y_train)
y_train_probs = clf.predict_proba(X_train)
y_training_pred = clf.predict(X_train)
```

[262]:
```
#test
y_test_pred = clf.predict(X_test)
y_test_probs = clf.predict_proba(X_test)

#set threshold evaluation
threshold = np.arange(0.35, 0.70, 0.05)
for x in threshold:
    y_pred = (clf.predict_proba(X_test)[:,1] >= x).astype('int')
    print('test', "\n", x, "\n", classification_report(y_test, y_pred))
```

[272]:
```
#performance
y_pred = (clf.predict_proba(X_test)[:,1] >= 0.5).astype('int')
```

[273]:
```
roc=roc_auc_score(y_test, y_pred)
print('roc', roc)
print('prediction_matrix', '\n', confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
roc 0.7810049893086244
prediction_matrix
 [[666  66]
 [ 32  60]]
              precision    recall  f1-score   support

           0       0.95      0.91      0.93       732
           1       0.48      0.65      0.55        92

    accuracy                           0.88       824
   macro avg       0.72      0.78      0.74       824
weighted avg       0.90      0.88      0.89       824
```

8 Discussion

The final model was revealed to be the best model applied in the analysis, as demonstrated by the gold standard adopted for unbalanced set, such as AUC and F1 score. The AUC provides an aggregate measure of performance across all possible classification thresholds. The value of 0.78 suggests a good compromise between the True and False positive rates, with a global accuracy of 88%. In addition, the F1 score of 0.74 indicates low false positives and low false negatives. However, in the deployment phase of the model, it is still possible to adjust the threshold of the model according to the use case and business needs. In other words, if we are interested in maximizing the level of precision of the minority class, we would increase the level of the threshold of the model. In a real-world scenario, for example, if we want to be totally sure about the client that will subscribed a term deposit, we will adopt a higher level of precision of the minority class, sacrificing its recall and global F1 score. To conclude, the final implementation of the model will be driven by the business and company needs.

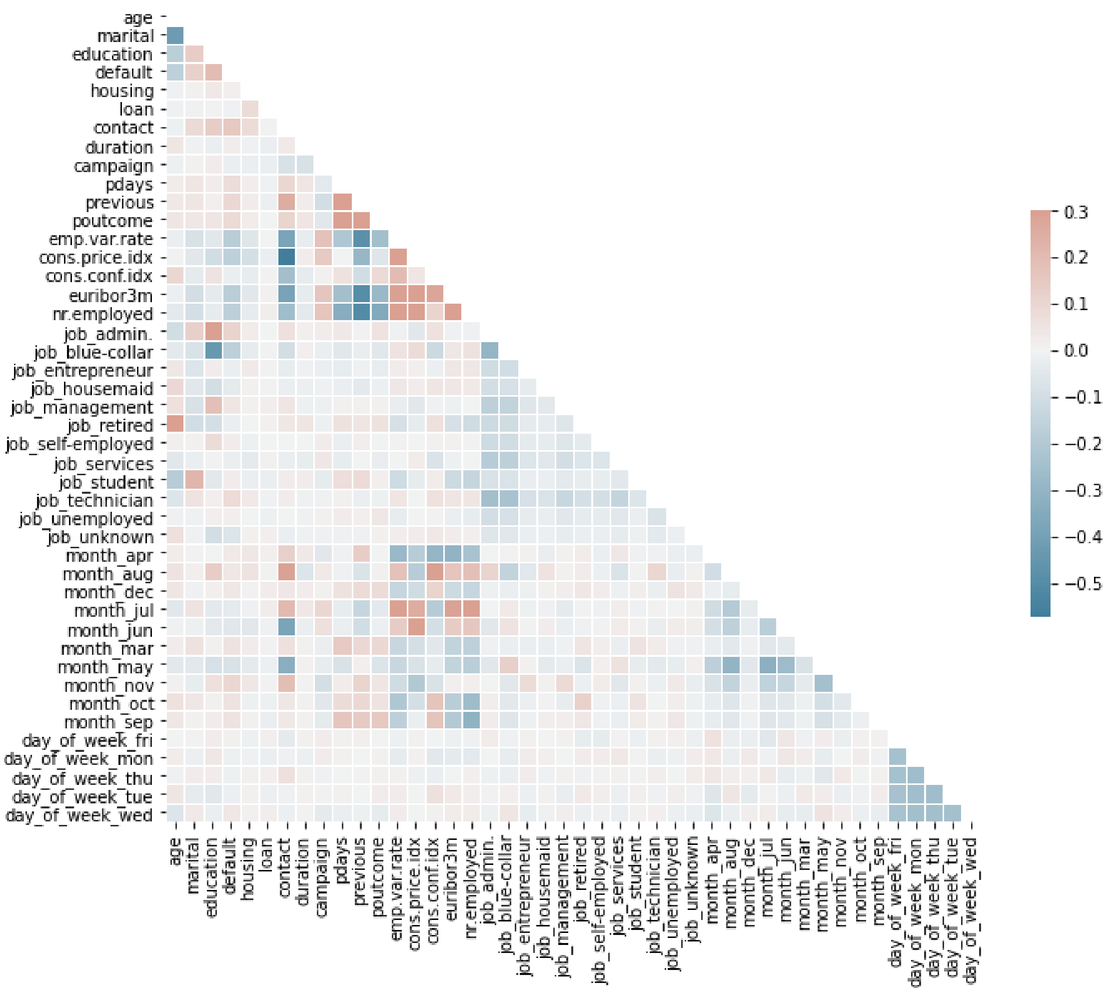
9 Save Model

[ ]:
```
import pickle
filename = 'finalized_model.sav'
pickle.dump(clf, open(filename, 'wb'))
```

10 Appendix

[130]:
```
#correlation matrix of the features (to identify potential correlations among␣
↪them)
corr = training_set.corr()
# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))
# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)
# Draw the heatmap with the mask and correct aspect ratio
10
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

[130]:
```
<AxesSubplot:>
```

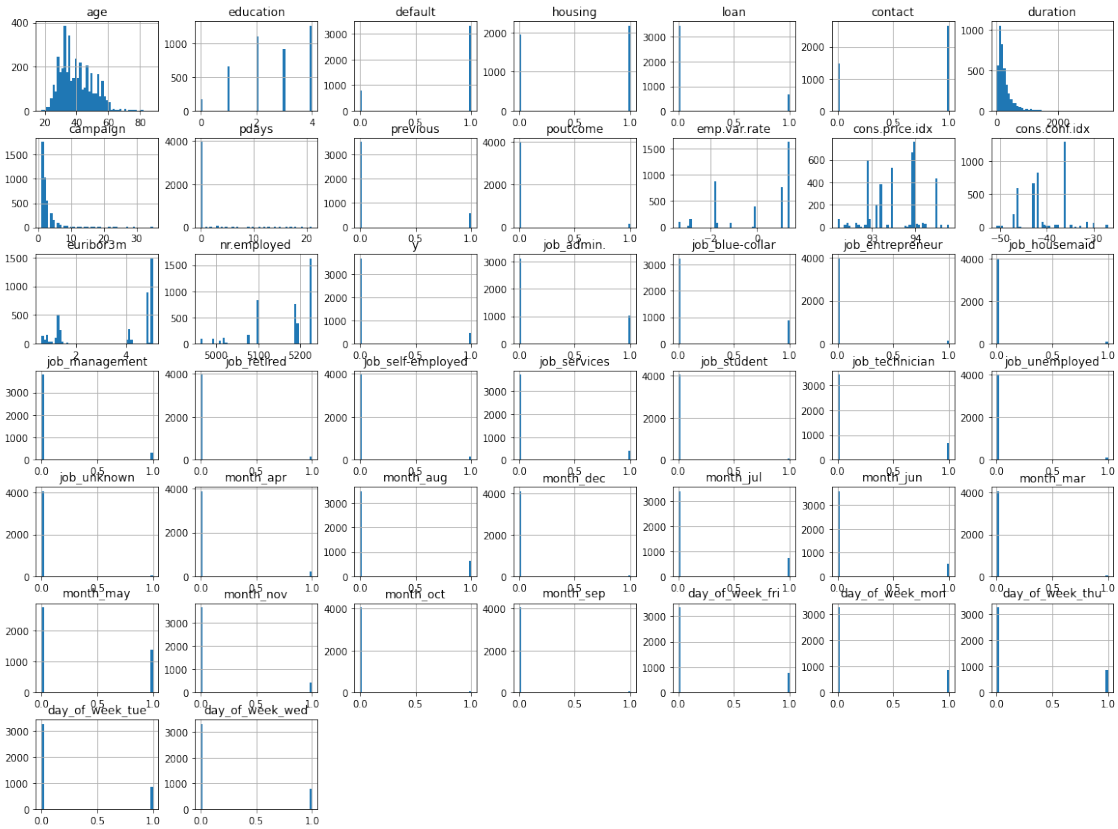


[146]:
```
#checking miss values
df.isnull().sum() #no miss values detected
```

[146]:
```
age               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
```

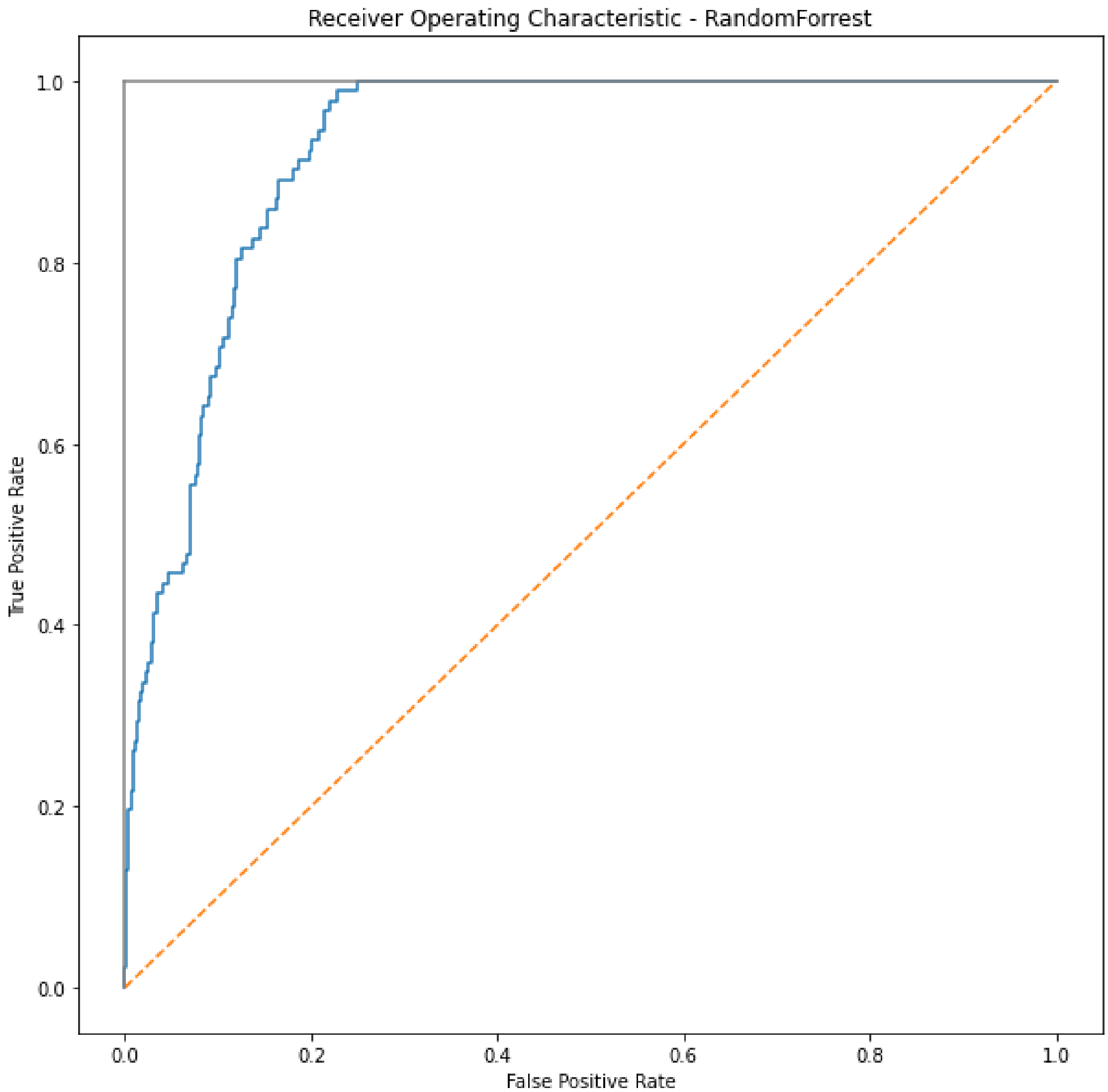
```
cons.price.idx        0
cons.conf.idx         0
euribor3m             0
nr.employed           0
y                     0
job_admin.            0
job_blue-collar       0
job_entrepreneur      0
job_housemaid         0
job_management        0
job_retired           0
job_self-employed     0
job_services          0
job_student           0
job_technician        0
job_unemployed        0
job_unknown           0
month_apr             0
month_aug             0
month_dec             0
month_jul             0
month_jun             0
month_mar             0
month_may             0
month_nov             0
month_oct             0
month_sep             0
day_of_week_fri       0
day_of_week_mon       0
day_of_week_thu       0
day_of_week_tue       0
day_of_week_wed       0
dtype: int64
```

[147]:
```
#checking distribution
df.hist(bins=50, figsize=(20, 15))
plt.show()
```



[286]:

```
#PLOT ROC
false_positive_rate1, true_positive_rate1, threshold1 = roc_curve(y_test,␣
    ↪y_test_probs[:,1])
plt.subplots(1, figsize=(10,10))
plt.title('Receiver Operating Characteristic - RandomForrest')
plt.plot(false_positive_rate1, true_positive_rate1)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".5"), plt.plot([1, 1] , c=".5")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```


Receiver Operating Characteristic - RandomForrest
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0