REI602M Machine Learning - Homework 4

Due: *Sunday* 14.2.2021 ¶

Objectives: Parameter tuning, Ensemble tree classifiers, Feature importance, Stacking, Pre-processing, performance metrics, scikit-learn and pandas.

Name: (your name here), **email:** (your email here), **collaborators:** (if any)

Please provide your solutions by filling in the appropriate cells in this notebook, creating new cells as needed. Hand in your solution on Gradescope, taking care to locate the appropriate page numbers in the PDF document. Make sure that you are familiar with the course rules on collaboration (encouraged) and copying (very, very, bad).

1. [Pre-processing and parameter tuning in an SVM classifier, 20 points]

The Statlog data set is an old benchmark in machine learning. It contains data from satellite images and the aim is to predict land type (red soil, cotton crop etc). There are 36 integer valued features and 6 classes. The file `sat.trn` contains the 4435 training examples and `sat.tst` contains 2000 test examples. Your task is to obtain an RBF-SVM classifier which obtains high classification accuracy on this data set.

a) Evaluate the accuracy of a RBF-SVM obtained with default values for C and γ.

b) Scale the training set prior to training an RBF-SVM (scale the test data accordingly) and repeat the task from a).

c) Use cross-validation on the (scaled) training set to identify optimal values of C and γ. You should start with a coarse grid and then do another run with a finer grid. Logarithmically spaced grid values are often used. Evaluate the accuracy of the resulting classifier by re-training using the best parameter values and report the error on the test set (why is the cross-validation error not a good estimate of the true classifier error here?)

d) Using randomized search for hyper-parameter values has been shown to be more efficient than traditional grid search. Instead of evaluating parameter values at regular intervals, the values are sampled from a distribution over possible parameter values. This enables considerable time savings (exhaustive grid search is expensive), or the identification of better parameter values for a given computational budget. Repeat the parameter search using `RandomizedSearchCV`. You can either fix the budget (n_iter parameter) so that the cost is comparable to the exhaustive grid search in c), or set the budget to e.g. 10% of the cost in iii). Report the results of the best classifier.

Summarize briefly your findings from a) to d).

Comments:

1) Description of the data set: https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite (https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite))

2) For scaling the data, see: `preprocessing.StandardScaler` in scikit.

3) The parameters C and γ in SVMs are called *hyper-parameters* since they are considered to be fixed during optimization of the model parameters (θ-values). The procedure of selecting hyper-parameter values is referred to as *model selection* and is typically performed by training the model for several different values of hyper-parameters, and evaluating the resulting model on a (cross-)validation set and finally selecting the values that give the best results.

4) An exhaustive search of parameter combinations on a grid is called is referred to as a *grid-search*. The `GridSearchCV` class in scikit-learn makes this easy to do. For details see the scikit documentation, sections 3.1 (Cross-validation: evaluating estimator performance), 3.2 (Tuning the hyper-parameters of an estimator) and the "Parameter estimation using grid search with cross-validation" example.

5) The paper *Random Search for Hyper-Parameter Optimization* by Bergstra and Bengio describes why randomized search of hyperparamter values is an efficient strategy. https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf (https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf)

In [193]:
```
#Imports
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

#Lesa gögn

data = pd.read_csv("sat.trn", sep=" ",header=None)

X=data[data.columns[:36]]
y=data[data.columns[-1]]



#A-liður
clf = SVC(kernel='rbf', gamma = 'scale')
clf.fit(X,y)
accuracy = clf.score(X,y)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9034949267192784
```

In [194]:
```
#B liður
scaler = StandardScaler()
Xscaled = scaler.fit_transform(X)
clfScaled =SVC(gamma='auto',kernel='rbf')
clfScaled.fit(Xscaled,y)
accuracyScaled = clfScaled.score(Xscaled,y)
print(f'Accuracy for the scaled dataset : {accuracyScaled}')
```

```
Accuracy for the scaled dataset : 0.9095828635851184
```

In [195]:
```
#C Liður

parameters = {'C':[1, 50], 'gamma' : [1,50]}
clf = GridSearchCV(SVC(), parameters, cv=5)

best_model = clf.fit(X,y)
print(best_model.best_estimator_.get_params()['C'])
print(best_model.best_estimator_.get_params()['gamma'])
```

```
1
1
```

```
In [198]: #D liður
          #Setja inn classifier fyrir bestu gildin

          C = np.linspace(1,100)
          gamma = np.linspace(1,100)

          parameters = dict(C=C, gamma = gamma)
          RandomSearch = RandomizedSearchCV(SVC(), parameters,n_iter=100, random_state =
          42)
          RandomSearch.fit(X,y)
          print(RandomSearch.best_params_)
```

```
{'gamma': 95.95918367346938, 'C': 57.57142857142857}
```

Sjáum að það er lítil breyting á accuracy ef við scölum breyturnar

RandomSearch modelið gefur okkur nákvæmari niðurstöður fyrir C og gamma breytuna þ.a við notum þær breytur til að þjálfa modelið okkar og athugum svo niðustöðunar.

```
In [199]: clf = SVC(kernel='rbf', gamma = 96, C = 58)
          clf.fit(X,y)
          accuracy = clf.score(X,y)
          print(f'Accuracy: {accuracy}')
```

```
Accuracy: 1.0
```

Accuracy hjá okkur er nuna komið í 100%

2. [Random Forests and feature selection, 20 points]

Quantitative Structure - Activity Relationship (QSAR) models relate the activity of chemical compounds to their structural properties. The activity can represent e.g. the potency of a drug or its toxicity. The structural properties may contain basic information such as i) the fraction of carbon atoms in the compound, ii) the spatial arrangement of atoms in the compound and iii) quantities computed from quantum mechanical simulations (*ab-initio* calculations).

The QSAR biodegradation Data Set `biodeg.csv` contains 41 molecular descriptors for two groups of compounds, those that are readily biodegradable (RB) and those that are not (NRB). The data set has 356 examples in the RB class and 699 in the NRB class, i.e. it is somewhat unbalanced.

a) Using `sklearn.ensemble.RandomForestClassifier` , obtain a classifier for predicting whether a given compound is readily biodegradable or not. Use a random 80/20 train/test set split for evaluting the performance of your classifier. Report the confusion matrix for the test set and calculate the accuracy of the classifier together with *sensitivity* and *specificity* (see below).

b) List the names of the 10 *most useful* features for the classification task (see below). Retrain a Random forests classifier using only the 10 most useful features and report sensitivity, specificity and accuracy. How does the performance compare to the classifier trained on the full feature set in a)?

Comments:

1) The file `biodeg_desc.txt` containes the feature names. A description of the data set can be found here: https://archive.ics.uci.edu/ml/datasets/QSAR+biodegradation (https://archive.ics.uci.edu/ml/datasets/QSAR+biodegradation)

2) Use `sklearn.model_selection.train_test_split` to create a train/test split.

3) A correctly predicted RB example is said to be a *true positive* and a correctly predicted NRB examples is said to be a *true negative*. When an NRB example is incorrectly predicted as RB it is a *false positive*. False negatives are defined analogously.

The *sensitivity* of a binary classifier is defined as TP/(TP+FN) and the *specificity* is defined as TN/(TN+FP) where TP is the number of true positives etc. These values are conveniently obtained from a confusion matrix, e.g. via `sklearn.metrics.confusion_matrix`

4) The feature importance measure provided by the Random Forests implementation in scikit has significant drawbacks. Therefore you will be using a so-called permutation importance measure (see problem 1 for a description). This is implemented in `sklearn.inspection.permutation_importance` .

5) Sidenote: Repeatedly retraining a classifier with smaller and smaller number of top features until the validation error (or out-of-bag error) starts to increase can easily lead to overfitting.

In [190]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

np.set_printoptions(suppress = True, precision=5)

data = np.genfromtxt('biodeg.csv',delimiter=';')
features = []
with open("biodeg_desc.txt") as f:
    for line in f:
        stak = line.split(':')[0]
        features.append(stak)


X = data[:,0:-1]
y = data[:,-1]


X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=155, random_
state=42)
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_siz
e=0.2)

#A liður
clf = RandomForestClassifier(n_estimators = 10, random_state =42)
clf.fit(X_train,y_train)
y_predTrain = clf.predict(X_train)
cmtrain = confusion_matrix(y_predTrain, y_train)
print(f'Confusion matrix for the training set:\n {cmtrain}')
print()

TN = cmtrain[0][0]
FN = cmtrain[1][0]
TP = cmtrain[1][1]
FP = cmtrain[0][1]

sensT = TP/(TP + FN)
print(f'Sensitivity for the training set: {sensT}')

spect = TN / (TN + FP)
print(f'Specificity for the training set: {spect}')

accuracyt = (TN+TP)/(TN+TP+FN+FP)
print(f'Accuracy for the training set: {accuracyt}')
print()

y_predTest = clf.predict(X_test)
cmtest = confusion_matrix(y_predTest, y_test)
print(f'Confusion matrix for the test set:\n {cmtest}')
print()

TN = cmtest[0][0]
FN = cmtest[1][0]
TP = cmtest[1][1]
FP = cmtest[0][1]
```

```
sensTest = TP/(TP + FN)
print(f'Sensitivity for the test set: {sensTest}')

spectest = TN / (TN + FP)
print(f'Specificity for the test set: {spectest}')

accuracytest = (TN+TP)/(TN+TP+FN+FP)
print(f'Accuracy for the test set: {accuracytest}')
print()
```

```
Confusion matrix for the training set:
 [[472   4]
 [  0 244]]

Sensitivity for the training set: 1.0
Specificity for the training set: 0.9915966386554622
Accuracy for the training set: 0.9944444444444445

Confusion matrix for the test set:
 [[115  16]
 [ 13  36]]

Sensitivity for the test set: 0.7346938775510204
Specificity for the test set: 0.8778625954198473
Accuracy for the test set: 0.8388888888888889
```

In [191]:
```
#B liður
from sklearn.inspection import permutation_importance

imp = permutation_importance(clf, X_val, y_val)
result = imp.importances_mean

top10 = result.argsort()[-10:]
print('Top 10 features')
for i in top10:
    print(features[i])
```

```
Top 10 features
34) F02[C-N]
15) SM6_L
10) nO
12) SdssC
9) nCp
27) SpMax_A
1) SpMax_L
11) F03[C-N]
39) SM6_B(m)
36) SpMax_B(m)
```

In [192]:
```
X_train10 = X_train[:, top10]
X_test10 = X_test[:,top10]

clf = RandomForestClassifier(n_estimators = 10, random_state =42)
clf.fit(X_train10,y_train)

y_predTrain = clf.predict(X_train10)
cmtrain = confusion_matrix(y_predTrain, y_train)
print(f'Confusion matrix for the training set:\n {cmtrain}')
print()

TN = cmtrain[0][0]
FN = cmtrain[1][0]
TP = cmtrain[1][1]
FP = cmtrain[0][1]

sensT = TP/(TP + FN)
print(f'Sensitivity for the training set: {sensT}')

spect = TN / (TN + FP)
print(f'Specificity for the training set: {spect}')

accuracyt = (TN+TP)/(TN+TP+FN+FP)
print(f'Accuracy for the training set: {accuracyt}')
print()

y_predTest = clf.predict(X_test10)
cmtest = confusion_matrix(y_predTest, y_test)
print(f'Confusion matrix for the test set:\n {cmtest}')
print()

TN = cmtest[0][0]
FN = cmtest[1][0]
TP = cmtest[1][1]
FP = cmtest[0][1]

sensTest = TP/(TP + FN)
print(f'Sensitivity for the test set: {sensTest}')

spectest = TN / (TN + FP)
print(f'Specificity for the test set: {spectest}')

accuracytest = (TN+TP)/(TN+TP+FN+FP)
print(f'Accuracy for the test set: {accuracytest}')
print()
```

```
Confusion matrix for the training set:
 [[471   4]
 [  1 244]]

Sensitivity for the training set: 0.9959183673469387
Specificity for the training set: 0.991578947368421
Accuracy for the training set: 0.9930555555555556

Confusion matrix for the test set:
 [[114  17]
 [ 14  35]]

Sensitivity for the test set: 0.7142857142857143
Specificity for the test set: 0.8702290076335878
Accuracy for the test set: 0.8277777777777777
```

3. [Stacked regression models, 30 points]

In this problem you will construct a *stacked* two-stage regression model for a subset of the Million Song Database (MSD). The data set contains audio features for approximately 500K songs. Each song is represented by 90 features describing its "timbre" that are derived from the sampled recordings. The task is to predict the release year of a song.

A two-stage stacking model has several regression models in stage 1, all trained on the same data set. Predictions from stage 1 models form a new (derived) data set which is used as input to a single regression model in stage 2. This model "blends" predictions from the stage 1 models to create a final prediction, hopefully more accurate than the individual stage 1 predictions.

Your stacked regression model will employ Lasso, ExtraTrees, Random Forests and Gradient boosted trees in stage 1 and a linear regression model in stage 2. Training and testing are performed as follows:

Training: Train each model on the training set, using default parameters to begin with, but increase the number of trees for Extra Trees and Random Forests. Construct a training data set for the stage 2 model by sending the *validation* set (not the original training set) through each of the stage 1 models, resulting in an `n_val` by 4 matrix X_2 of prediction values. Train a linear regression model for stage 2 on (X_2, y_{val}).

Testing: Send the test data though all the models in stage 1 to obtain an `n_test` by 4 matrix. The stage 2 linear regression model is used to predict the data in this matrix to obtain the final predictions.

Start by creating a histogram of the number of songs per year in the data set to obtain insight into how realistic this prediction task is.

a) [20 points] Report the mean-squared error of the individual stage 1 models on the test set and the corresponding R^2 coefficient. Construct the stacked regression model described above, report its mean-squared error and R^2 coefficient on the test set.

b) [10 points] Answer the following questions:

i) Are the individual models doing a good job on the prediction task? (Consider the root-mean squared error).

ii) Are the individual classifiers failing in some obvious way (failure modes)?

iii) Is the stacking procedure worth the extra effort in your opinion? Why or why not?

iv) Why is it not a good idea to use the original training set to construct the X_2 data set for the stage 2 regression model?

Comments:

1) Download the subset of the Million Song Databse from here (210 MB): http://archive.ics.uci.edu/ml/datasets/YearPredictionMSD# (http://archive.ics.uci.edu/ml/datasets/YearPredictionMSD#) (mirror: https://notendur.hi.is/steinng/kennsla/2021/ml/data/YearPredictionMSD.zip (https://notendur.hi.is/steinng/kennsla/2021/ml/data/YearPredictionMSD.zip))

2) Use the train, validation and test partitions of the data defined in `load_msd.py`

```
 import load_msd as lmsd
X_train, y_train, X_val, y_val, X_test, y_test = lmsd.get_data(ntrain=10000)
```

3) For Extra Trees and Random Forests you can set `n_jobs=-1` to use multiple cores/processors for training

In [9]:
```
#Stage1
#Lasso, ExtraTrees, Random Forests and Gradient boosted trees
import load_msd as lmsd
X_train, y_train, X_val, y_val, X_test, y_test = lmsd.get_data(ntrain=1000)
```

In [10]:
```
#Lasso
from sklearn.metrics import mean_squared_error
from sklearn import linear_model

clfLasso = linear_model.Lasso(alpha = 0.1)
clfLasso.fit(X_train,y_train)

y_pred = clfLasso.predict(X_test)
MSELasso = mean_squared_error(y_test,y_pred)
print(MSELasso)
RLasso = clfLasso.score(X_test, y_test)
print(RLasso)

print(y_pred)
```

```
101.45056146704584
0.1352414043173511
[2000.12069266 2001.40940871 2001.98053083 ... 1996.06972017 1990.83448327
 1991.76367077]
```

In [11]:
```
#ExtraTrees
from sklearn.ensemble import ExtraTreesRegressor
clfET = ExtraTreesRegressor(n_estimators=1000)
clfET.fit(X_train, y_train)
y_pred = clfET.predict(X_test)
MSEET = mean_squared_error(y_test,y_pred)
RET = clfET.score(X_test, y_test)

print(MSEET)
print(RET)
```

```
97.66483979184319
0.1675106723445864
```

In [17]:
```
#RandomForest
from sklearn.ensemble import RandomForestRegressor
clfRF = RandomForestRegressor(n_estimators = 1000)
clfRF.fit(X_train, y_train)
y_pred = clfRF.predict(X_test)

MSErf = mean_squared_error(y_test,y_pred)
Rrf = clfRF.score(X_test, y_test)

print(MSErf)
print(Rrf)
```

```
101.14853665191275
0.1378158460081509
```

In [20]:
```
#Gradient boosted trees
from sklearn.ensemble import GradientBoostingClassifier

clfGBS = GradientBoostingClassifier(n_estimators=200)
clfGBS.fit(X_train, y_train)
y_pred = clfGBS.predict(X_test)

MSEgbs = mean_squared_error(y_test,y_pred)
Rgbs = clfGBS.score(X_test, y_test)

print(MSEgbs)
print(Rgbs)
```

```
221.90354094214354
0.057824849826114445
```

Modelin gefa okkur mismunadi niðurstöður með MSE og R gildið. Tökum svo spágildin úr þeim modelum og þjálfum þau með LinearRegression

In [21]:
```
#part 2
y_predLasso = clfLasso.predict(X_val)
y_predExtraTrees =clfET.predict(X_val)
y_predRandomForest = clfRF.predict(X_val)
y_predgbs = clfGBS.predict(X_val)

X2 = np.array([ y_predLasso, y_predExtraTrees, y_predRandomForest, y_predgbs])
```

In [22]:
```python
#Linear regression model
from sklearn.linear_model import LinearRegression

clf = LinearRegression()
clf.fit(X2.T,y_val)
y_pred = clf.predict(X2.T)

print(y_pred.shape)
print(y_test.shape)
print(y_val.shape)
print(y_val.shape)
mse = mean_squared_error(y_val,y_pred)
print(mse)
```

```
(20000,)
(31630,)
(20000,)
(20000,)
92.38050184347401
```

b liður

i) Are the individual models doing a good job on the prediction task? (Consider the root-mean squared error).

Modelin eru öll með frekar há MSE en Gradient Boosted Trees modelið gefur okkur mun hærra MSE en hin modelin en gefur okkur samt besta R gildið.

ii) Are the individual classifiers failing in some obvious way (failure modes)?

Sakvæmt MSE gildunum þeirra eru þetta ekki marktækar niðurstöður.

iii) Is the stacking procedure worth the extra effort in your opinion? Why or why not?

Það er þess virði að athuga "stacking procedure", vegna þess að við erum að taka besta úr báðum heimum, t.d GBT gefur okkur gott R gildi á meðan Extra trees gefur okkur lægsta MSE gildið. Loka niðurstöðunar gaf okkur einnig besta gildi úr MSE fyrir öll modelin.

iv) Why is it not a good idea to use the original training set to construct the $X2$ data set for the stage 2 regression model?

Vegna þess að þá erum við bara að "fitta" gögnin við linear regression model, gætum þá alveg eins slept stage1 og fittað bara gögnin strax við linear regression ef það væri meiningin okkar.

4. [Preprocessing, performance metrics, 30 points]

In this problem you will construct a predictive model for Telemarketing. The data comes from a telemarketing campaign in Portugal where the goal was to get clients to subscribe to long-term savings deposits. The predictive model is to be used to identify customers that are likely to subscribe, based on personal information, economic indicators, whether the client has been contacted before etc. This should make the campaign more effective and reduce marketing costs.

The data is in file `bank-additional-full.csv` with a brief description in `bank-additional-names.txt` . The data contains a mixture of continuous and categorical features, with categorical data in text format. Some preprocessing is therefore needed before you can use it with scikit-learn algorithms.

The data is time ordered which means that randomly splitting it into training and test sets amounts to peeking into the future and will provide too optimistic estimates of model performance. This is therefore not a suitable evaluation strategy. In situations like this, one uses historical data to train a model and predicts data from the current period. To simulate this scenario you will use the most recent (last) 4000 samples for testing. You then use the remaining samples for training (or a subset thereof).

Train a Random Forests or an Extra Trees classifier and evaluate it on the test set using an appropriate performance metric (see below). The selection of metric should take into account whether the classes are balanced or not, as well as the goal of the prediction task. Do you think your model would be useful in practice? Why or why not?

Comments:

1) This is an open ended problem. There is no single correct answer.

2) The data set is described in some detail here: https://archive.ics.uci.edu/ml/datasets/bank+marketing (https://archive.ics.uci.edu/ml/datasets/bank+marketing) and a previous attempt at predictive modeling in: https://www.sciencedirect.com/science/article/pii/S016792361400061X (https://www.sciencedirect.com/science/article/pii/S016792361400061X)

3) To convert the data into a matrix format suitable for scikit-learn, it is probably easiest to use the Python Data Analysis Library (pandas) package. You load the data using

```
import pandas as pd
bank_df=pd.read_csv('bank-additional-full.csv',sep=';')
```

You can iterate over the features using e.g.

```
for col in bank_df.columns:
    if bank_df[col].dtype == object:
        print("Categorical: ",col)
    else:
        print("Numerical: ", col)
```

The output variable (`y`) is 1 if a customer subscribes and 0 otherwise.

Start by using only the numerical data. Then add the categorical data gradually. More data does not always help.

The simplest conversion of categories to integers is called label encoding. In pandas:

```
bank_df['y']=bank_df['y'].astype('category')
bank_df['y']=bank_df['y'].cat.codes
y=bank_df['y'].values
bank_df=bank_df.drop('y',axis=1) # Remove output variable
```

or using scikit-learn instead:

```
from sklearn.preprocessing import LabelBinarizer
lb = LabelBinarizer()
y = lb.fit_transform(bank_df['y'])[:,0]
```

Label encoding of a feature assumes that the feature values have a natural ordering (are ordinal). This has some drawbacks as detailed in the lecture notes and one-hot-encoding is generally preferred. This is most conveniently done by using `pd.get_dummies` with `drop_first=True` . For this particular data set, direct application of one-hot-encoding does not necessarily improve performance.

4) In addition to `sklearn.metrics.confusion_matrix` which can be used to derive sensitivity, specificity and accuracy, the `sklearn.metrics.classification_report` class provides performance metrics called precision recall and F-score (see Wikipedia for details).

In [39]:
```
import pandas as pd

np.set_printoptions(suppress = True, precision=3)
pd.set_option('precision',1)


bank_df=pd.read_csv('bank-additional-full.csv',sep=';')

#Breyta í klasanum okkar í numerical breytu
from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
bank_df['y'] = lb.fit_transform(bank_df['y'])[:,0]

#Droppa öllum categorical breytum
for col in bank_df.columns:
    if bank_df[col].dtype == object:
        print("Categorical: ",col)
        bank_df = bank_df.drop(col, axis=1)
    else:
        print("Numerical: ", col)
```

```
Numerical:  age
Categorical:  job
Categorical:  marital
Categorical:  education
Categorical:  default
Categorical:  housing
Categorical:  loan
Categorical:  contact
Categorical:  month
Categorical:  day_of_week
Numerical:  duration
Numerical:  campaign
Numerical:  pdays
Numerical:  previous
Categorical:  poutcome
Numerical:  emp.var.rate
Numerical:  cons.price.idx
Numerical:  cons.conf.idx
Numerical:  euribor3m
Numerical:  nr.employed
Numerical:  y
```

In [40]:
```
print(bank_df.shape)
bank_dftest = bank_df.tail(4000)

bank_dftrain = bank_df.head(bank_df.shape[0]-bank_dftest.shape[0])

print(bank_dftest.shape)

print(bank_dftest.describe())
```

```
(41188, 11)
(4000, 11)
          age  duration  campaign   pdays  previous  emp.var.rate  \
count  4000.0    4000.0    4000.0  4000.0    4000.0        4000.0   
mean     43.4     288.9       1.8   713.8       0.8          -2.3   
std      16.5     252.4       1.3   449.3       1.0           0.9   
min      17.0       1.0       1.0     0.0       0.0          -3.4   
25%      30.0     134.0       1.0    10.0       0.0          -3.4   
50%      38.0     220.5       1.0   999.0       1.0          -1.8   
75%      55.0     359.2       2.0   999.0       1.0          -1.7   
max      98.0    3785.0      16.0   999.0       7.0          -1.1   

       cons.price.idx  cons.conf.idx  euribor3m  nr.employed       y  
count          4000.0         4000.0     4000.0       4000.0  4000.0  
mean             93.3          -35.1        0.8       5013.0     0.5  
std               0.9            6.2        0.1         34.5     0.5  
min              92.2          -50.8        0.6       4963.6     0.0  
25%              92.4          -39.8        0.7       4991.6     0.0  
50%              93.4          -34.6        0.8       5008.7     0.0  
75%              94.1          -30.1        0.9       5017.5     1.0  
max              94.8          -26.9        1.1       5076.2     1.0  
```

In [43]:
```
from sklearn.metrics import confusion_matrix

X_train1 = bank_dftrain.drop('y', axis=1)
y_train1 = bank_dftrain['y']

X_test1 = bank_dftest.drop('y', axis = 1)
y_test1 = bank_dftest['y']

clf = RandomForestClassifier(random_state=42, n_estimators=100)
clf = clf.fit(X_train1,y_train1)

y_predTrain = clf.predict(X_train1)
cm = confusion_matrix(y_train1, y_predTrain)

print(f'Confusion matrix for the training set:\n {cm}')
print()

y_predTest = clf.predict(X_test1)
cmtest = confusion_matrix(y_test1, y_predTest)

print(f'Confusion matrix for the Test set :\n {cmtest}')
print()

#Declaring true negative, false negative, true positive and false positive for
the training set
TN = cm[0][0]
FN = cm[1][0]
TP = cm[1][1]
FP = cm[0][1]

sens = TP/(TP + FN)
print(f'Sensitivity for the training set : {sens}')

spec = TN / (TN + FP)
print(f'Specification for the training set : {spec}')

accuracy = (TN+TP)/(TN+TP+FN+FP)
print(f'Total accuracy for the training set: {accuracy}')
print()

#Test set
TNt = cmtest[0][0]
FNt = cmtest[1][0]
TPt = cmtest[1][1]
FPt = cmtest[0][1]

senst = TPt/(TPt + FNt)
print(f'Sensitivity for the test set : {senst}')

spect = TNt / (TNt + FPt)
print(f'Specification for the test set : {spect}')

accuracyt = (TNt+TPt)/(TNt+TPt+FNt+FPt)
print(f'Total accuracy for the test set: {accuracyt}')
```

```
Confusion matrix for the training set:
 [[34402     1]
 [    6  2779]]

Confusion matrix for the Test set :
 [[1992  153]
 [1341  514]]

Sensitivity for the training set : 0.9978456014362657
Specification for the training set : 0.9999709327674912
Total accuracy for the training set: 0.999811767236743

Sensitivity for the test set : 0.277088948787062
Specification for the test set : 0.9286713286713286
Total accuracy for the test set: 0.6265
```

In [45]:
```
from sklearn.metrics import classification_report
clf_reportTrain = classification_report(y_train1, y_predTrain)
print(f'Classification Report for the training set:\n {clf_reportTrain}')
print()

clf_reportTest = classification_report(y_test1, y_predTest)
print(f'Classification Report for the test set: \n {clf_reportTest}')
```

```
Classification Report for the training set:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     34403
           1       1.00      1.00      1.00      2785

    accuracy                           1.00     37188
   macro avg       1.00      1.00      1.00     37188
weighted avg       1.00      1.00      1.00     37188


Classification Report for the test set:
               precision    recall  f1-score   support

           0       0.60      0.93      0.73      2145
           1       0.77      0.28      0.41      1855

    accuracy                           0.63      4000
   macro avg       0.68      0.60      0.57      4000
weighted avg       0.68      0.63      0.58      4000
```

In [ ]:
```
#Todo tala um support á klasanum og að það mun alltaf nanast predicta 0, og þe
gar við erum með svipuð mörg instance af y þa
#er classifierinn með hræðilegt accuracy þar sem það nanast spáir kringum 45%
 vitlaust
```

Þegar við horfum á niðurstöðunar frá training og test settinu okkar, þá sjáum við að klasa breytan okkar(spágildið) er nánast bara að predicta ("0"), þ.a modelið okkar myndi þá í flestum tilfellum spá fyrir að client myndi ekki vilja áskrift. Í test settinu okkar er svo u.þ.b sama support á ("0") og ("1") en það gefur okkur kringum 63% nákvæmni sem eru hræðilegar niðurstöður. Það þyrfti að Preprocessa gögnin betur til að fá betri niðurstöður úr þessi modeli.