lab1

March 27, 2023

Prep

Read the data

[20]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Read the CSV file.
data = pd.read_csv('data.csv', skiprows=1)

# Select the relevant numerical columns.
selected_cols = ['LB', 'AC', 'FM', 'UC', 'DL', 'DS', 'DP', 'ASTV', 'MSTV',␣
 ↪'ALTV',
                 'MLTV', 'Width', 'Min', 'Max', 'Nmax', 'Nzeros', 'Mode',␣
 ↪'Mean',
                 'Median', 'Variance', 'Tendency', 'NSP']
data = data[selected_cols].dropna()

# Shuffle the dataset.
data_shuffled = data.sample(frac=1.0, random_state=0)

# Split into input part X and output part Y.
X = data_shuffled.drop('NSP', axis=1)

# Map the diagnosis code to a human-readable label.
def to_label(y):
    return [None, 'normal', 'suspect', 'pathologic'][(int(y))]

Y = data_shuffled['NSP'].apply(to_label)

# Partition the data into training and test sets.
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.4,␣
 ↪random_state=0)
```

[21]:
```
X.head()
```

[21]:
```
         LB   AC   FM   UC   DL   DS   DP  ASTV  MSTV  ALTV  …  Width  \
658   130.0  1.0  0.0  3.0  0.0  0.0  0.0  24.0   1.2  12.0  …   35.0
1734  134.0  9.0  1.0  8.0  5.0  0.0  0.0  59.0   1.2   0.0  …  109.0
1226  125.0  1.0  0.0  4.0  0.0  0.0  0.0  43.0   0.7  31.0  …   21.0
1808  143.0  0.0  0.0  1.0  0.0  0.0  0.0  69.0   0.3   6.0  …   27.0
825   152.0  0.0  0.0  4.0  0.0  0.0  0.0  62.0   0.4  59.0  …   25.0

        Min    Max  Nmax  Nzeros   Mode   Mean  Median  Variance  Tendency
658   120.0  155.0   1.0     0.0  134.0  133.0   135.0       1.0       0.0
1734   80.0  189.0   6.0     0.0  150.0  146.0   150.0      33.0       0.0
1226  120.0  141.0   0.0     0.0  131.0  130.0   132.0       1.0       0.0
1808  132.0  159.0   1.0     0.0  145.0  144.0   146.0       1.0       0.0
825   136.0  161.0   0.0     0.0  159.0  156.0   158.0       1.0       1.0

[5 rows x 21 columns]
```

Create models

[22]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron

from sklearn.svm import LinearSVC
from sklearn.neural_network import MLPClassifier

clf0 = DecisionTreeClassifier(random_state=0)
clf1 = RandomForestClassifier(random_state=0)
clf2 = GradientBoostingClassifier(random_state=0)
clf3 = LogisticRegression(random_state=0)
clf4 = Perceptron(random_state=0)
clf5 = LinearSVC(random_state=0)
clf6 = MLPClassifier(random_state=0)
```

Cross validate all models

[23]:
```
from sklearn.model_selection import cross_val_score

scores = []

scores.append((cross_val_score(clf0, Xtrain, Ytrain), "DecisionTreeClassifier"))
scores.append((cross_val_score(clf1, Xtrain, Ytrain), "RandomForestClassifier"))
scores.append((cross_val_score(clf2, Xtrain, Ytrain),␣
↪"GradientBoostingClassifier"))
scores.append((cross_val_score(clf3, Xtrain, Ytrain), "LogisticRegression"))
scores.append((cross_val_score(clf4, Xtrain, Ytrain), "Perceptron"))
```

```python
scores.append((cross_val_score(clf5, Xtrain, Ytrain), "LinearSVC"))
scores.append((cross_val_score(clf6, Xtrain, Ytrain), "MLPClassifier"))
```

```
/home/david/.local/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/david/.local/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/david/.local/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/david/.local/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
/home/david/.local/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/david/.local/lib/python3.10/site-packages/sklearn/svm/_base.py:1244:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn(
/home/david/.local/lib/python3.10/site-packages/sklearn/svm/_base.py:1244:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn(
/home/david/.local/lib/python3.10/site-packages/sklearn/svm/_base.py:1244:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn(
/home/david/.local/lib/python3.10/site-packages/sklearn/svm/_base.py:1244:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn(
/home/david/.local/lib/python3.10/site-packages/sklearn/svm/_base.py:1244:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn(
```

Print the highest performing models

[24]:
```
max = [(np.sum(score[0])/5,score[1]) for score in scores]

print(sorted(max, key=lambda x: x[0], reverse=True))
```

```
[(0.9513725490196079, 'GradientBoostingClassifier'), (0.9380392156862746,
'RandomForestClassifier'), (0.916078431372549, 'DecisionTreeClassifier'),
(0.8784313725490197, 'MLPClassifier'), (0.8698039215686275,
'LogisticRegression'), (0.8109803921568627, 'LinearSVC'), (0.7576470588235293,
'Perceptron')]
```

Step 4. Final evaluation

When you have found a classifier that gives a high accuracy in the cross-validation evaluation, train it on the whole training set and evaluate it on the held-out test set.

clf0, clf1, and clf2 are the highest performing from the data-set above so we train them on the entire data-set

[25]:
```python
from sklearn.metrics import accuracy_score

classifiers = [clf0, clf1, clf2]

for clf in classifiers:
    clf.fit(X, Y)
    Yguess = clf.predict(Xtest)
    print(accuracy_score(Ytest, Yguess), clf)
```

```
1.0 DecisionTreeClassifier(random_state=0)
1.0 RandomForestClassifier(random_state=0)
0.9905992949471211 GradientBoostingClassifier(random_state=0)
```

Task 2: Decision trees for classification

We could not get the code to run in this notebook (?) but out code runs in the included Lecture 1.ipynb

Below is a copy of what is written there:

OUR CODE FOR TASK 2

We create a list containing depth values from 1 to 16, and from there we pick and print the one with the highest accuracy.

It seems like 12 is the best value for this scenario, and after a depth of 12 the acurracy wstarts to fall of for this data-set.

We also print and show a desicion tree with depth 3

[26]:
```python
cls = TreeClassifier(max_depth=3)

data = pd.read_csv('data.csv', skiprows=1)
# Select the relevant numerical columns.
selected_cols = ['LB', 'AC', 'FM', 'UC', 'DL', 'DS', 'DP', 'ASTV', 'MSTV',␣
 ↪'ALTV',
                 'MLTV', 'Width', 'Min', 'Max', 'Nmax', 'Nzeros', 'Mode',␣
 ↪'Mean',
                 'Median', 'Variance', 'Tendency', 'NSP']
data = data[selected_cols].dropna()

# Shuffle the dataset.
data_shuffled = data.sample(frac=1.0, random_state=0)

# Split into input part X and output part Y.
XNEW = data_shuffled.drop('NSP', axis=1)

# Map the diagnosis code to a human-readable label.
```

```python
def to_label(y):
    return [None, 'normal', 'suspect', 'pathologic'][(int(y))]

YNEW = data_shuffled['NSP'].apply(to_label)
cls.fit(XNEW, YNEW)

best_classifier = _, 0
for max_depth in range(1, 16):
    print('Classifying tree at', max_depth+1, 'of', 16, end="\r")
    clfTree = TreeClassifier(max_depth)
    clfTree.fit(XNEW, YNEW)
    score = cross_val_score(clfTree, XNEW, YNEW, cv=5, scoring='accuracy').
 ↪mean()
    if score > best_classifier[1]:
        best_classifier = clfTree, score

print(best_classifier)
cls.draw_tree()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 21
     18     return [None, 'normal', 'suspect', 'pathologic'][(int(y))]
     20 YNEW = data_shuffled['NSP'].apply(to_label)
---> 21 cls.fit(XNEW, YNEW)
     23 best_classifier = _, 0
     24 for max_depth in range(1, 16):

Cell In[15], line 20, in TreeClassifier.fit(self, X, Y)
     18 else:
     19     raise Exception(f'Unknown criterion: {self.criterion}')
---> 20 super().fit(X, Y)
     21 self.classes_ = sorted(set(Y))

Cell In[10], line 24, in DecisionTree.fit(self, X, Y)
     22     self.names = None
     23 Y = np.array(Y)
---> 24 self.root = self.make_tree(X, Y, self.max_depth)

Cell In[10], line 66, in DecisionTree.make_tree(self, X, Y, max_depth)
     62 # Select the "most useful" feature and split threshold. To rank the␣
 ↪"usefulness" of features,
     63 # we use one of the classification or regression criteria.
     64 # For each feature, we call best_split (defined in a subclass). We then␣
 ↪maximize over the features.
     65 n_features = X.shape[1]
```

```
---> 66 _, best_feature, best_threshold = max(self.best_split(X, Y, feature) fo
 ↪feature in range(n_features))
     68 if best_feature is None:
     69     return DecisionTreeLeaf(default_value)

TypeError: 'list' object is not callable
```

Task 3: A regression example: predicting apartment prices

Here Links to an external site. is another dataset. This dataset was created by Sberbank and contains some statistics from the Russian real estate market. Here

Links to an external site. is the Kaggle page where you can find the original data.

Since we will just be able to handle numerical features and not symbolic ones, we'll need with a simplified version of the dataset. So we'll just select 9 of the columns in the dataset. The goal is to predict the price of an apartment, given numerical information such as the number of rooms, the size of the apartment in square meters, the floor, etc. Our approach will be similar to what we did in the classification example: load the data, find a suitable model using cross-validation over the training set, and finally evaluate on the held-out test data.

The following code snippet will carry out the basic reading and preprocessing of the data.

[27]:
```
# Read the CSV file using Pandas.
alldata = pd.read_csv('sberbank.csv')

# Convert the timestamp string to an integer representing the year.
def get_year(timestamp):
    return int(timestamp[:4])
alldata['year'] = alldata.timestamp.apply(get_year)

# Select the 9 input columns and the output column.
selected_columns = ['price_doc', 'year', 'full_sq', 'life_sq', 'floor',␣
 ↪'num_room', 'kitch_sq', 'full_all']
alldata = alldata[selected_columns]
alldata = alldata.dropna()

# Shuffle.
alldata_shuffled = alldata.sample(frac=1.0, random_state=0)

# Separate the input and output columns.
X = alldata_shuffled.drop('price_doc', axis=1)
# For the output, we'll use the log of the sales price.
Y = alldata_shuffled['price_doc'].apply(np.log)

# Split into training and test sets.
```

```
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.2,␣
↪random_state=0)
```

0.1 Answer to qustion 3

In the code, four different regression models are used: RandomForestRegressor, Ridge, Lasso, and GradientBoostingRegressor. The hyperparameters of each model are set, and the models are trained on the training dataset (Xtrain, Ytrain) using cross-validation. The negative mean squared error is then calculated for each model on the test dataset.

To compare the performance of the models, you can look at the value of the negative mean squared error for each model. The model with the negative mean squared error that is closest to zero is considered to be the best model.

It is important to note that the performance of a regression model can depend on the specific dataset being used, and the choice of evaluation metric may also affect the comparison of models. The number of combinations are also endlesss so there is no way of knowing what the optimal model is. We only know that we don't have the worst modell.

[30]:
```
from sklearn.linear_model import Ridge, Lasso
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.model_selection import cross_validate

#Test a some regression models

regr = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=0)
print((cross_validate(regr, Xtrain, Ytrain,
      scoring='neg_mean_squared_error').get('test_score')).mean())

regr = Ridge(alpha=1, random_state=0, solver='auto', max_iter=100, tol=1e-1)
print((cross_validate(regr, Xtrain, Ytrain,
          scoring='neg_mean_squared_error').get('test_score')).mean())

regr = Lasso(alpha=1, random_state=0, max_iter=100, tol=1e-1)
print((cross_validate(regr, Xtrain, Ytrain,
          scoring='neg_mean_squared_error').get('test_score')).mean())

regr = GradientBoostingRegressor(n_estimators=100, max_depth=10, random_state=0)
print((cross_validate(regr, Xtrain, Ytrain, scoring = 'neg_mean_squared_error').
↪get('test_score')).mean())

#random forest regressor seems promising, let's test it with different␣
↪parameters
#Test a set of combinations for random forest regressor for a set of variables␣
↪that we think will be important

best_score = -5
n_estimators = 0
```

```
max_depth = 0

for n_estimators in [5, 10, 100]:
    for max_depth in [1, 10, 100]:
        regressor = RandomForestRegressor(n_estimators=n_estimators,␣
↪max_depth=max_depth, random_state=0)
        score = ((cross_validate(regr, Xtrain, Ytrain,␣
↪scoring='neg_mean_squared_error').get('test_score'))).mean()
        print('Iteration: ', n_estimators, max_depth, 'Score: ', score,␣
↪end='\r')
        if score > best_score:
            best_score = score
            n_estimators = n_estimators
            max_depth = max_depth
            regr = regressor


print((cross_validate(regr, Xtrain, Ytrain,
      scoring='neg_mean_squared_error').get('test_score')).mean())
```

```
-0.2659229822429888
-0.3013978423217977
-0.30104708463292107
-0.2929452077385661
-0.31737303376625764Score:  -0.31737303376625764
```

[33]:
```
from sklearn.metrics import mean_squared_error

#Use the best combination of variables to train the model
regr = RandomForestRegressor(n_estimators = n_estimators , max_depth =␣
↪max_depth, random_state=0)
regr.fit(Xtrain, Ytrain)

mean_squared_error(Ytest, regr.predict(Xtest))
```

[33]: 0.29907312957355675