CSE572-Lab9-key

November 4, 2022

1 CSE 572: Lab 9

In this lab, you will practice implementing ensemble models.

To execute and make changes to this notebook, click File > Save a copy to save your own version in your Google Drive or Github. Read the step-by-step instructions below carefully. To execute the code, click on each cell below and press the SHIFT-ENTER keys simultaneously or by clicking the Play button.

When you finish executing all code/exercises, save your notebook then download a copy (.ipynb file). Submit the following **three** things: 1. a link to your Colab notebook, 2. the .ipynb file, and 3. a pdf of the executed notebook on Canvas.

To generate a pdf of the notebook, click File > Print > Save as PDF.

[1]:
```
# Import libraries
import numpy as np
import pandas as pd

# Set the random seed for reproducibility
seed = 0
np.random.seed(0)
```

1.1 Ensemble of hybrid models

One straighforward approach for constructing an ensemble classifier is to train k separate classifiers using different classification methods and then combine their predictions using majority vote. In the first exercise, you will use Scikit-learn to train a k nearest neighbors, naive Bayes, and logistic regression classifier separately and then combine their predictions using a VotingClassifier.

1.1.1 Load the dataset

We will use the Wisconsin breast cancer dataset with class values 'benign' or 'malignant'. Below, we will load the dataset and perform preprocessing as in previous labs.

[2]:
```
import numpy as np

# Load the Wisconsin breast cancer dataset
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/
↪breast-cancer-wisconsin/breast-cancer-wisconsin.data', header=None)
```

```
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size',␣
 ↪'Uniformity of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare␣
 ↪Nuclei', 'Bland Chromatin',
                'Normal Nucleoli', 'Mitoses','Class']

data = data.drop(['Sample code'],axis=1)

data = data.replace('?',np.NaN)
data['Bare Nuclei'] = pd.to_numeric(data['Bare Nuclei'])

data
```

[2]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	\
0	5	1	1	
1	5	4	4	
2	3	1	1	
3	6	8	8	
4	4	1	1	
..	…	…	…	
694	3	1	1	
695	2	1	1	
696	5	10	10	
697	4	8	6	
698	4	8	8	

	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	\
0	1	2	1.0	
1	5	7	10.0	
2	1	2	2.0	
3	1	3	4.0	
4	3	2	1.0	
..	…	…	…	
694	1	3	2.0	
695	1	2	1.0	
696	3	7	3.0	
697	4	3	4.0	
698	5	4	5.0	

	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	3	1	1	2
1	3	2	1	2
2	3	1	1	2
3	3	7	1	2
4	3	1	1	2
..	…	…	…	…
694	1	1	1	2

```
695                1                1        1       2
696                8               10        2       4
697               10                6        1       4
698               10                4        1       4

[699 rows x 10 columns]
```

After loading the dataset, we clean it by removing samples with missing data, duplicates, or outliers using the code from Labs 2-3.

[3]:
```
def inds_nans(df):
    inds = df.isna().any(axis=1)
    # print('Found {} rows that had NaN values.'.format(inds.sum()))
    return inds

def inds_dups(df):
    inds = df.duplicated()
    # print('Found {} rows that were duplicates.'.format(inds.sum()))
    return inds

def inds_outliers(df):
    # In this example, we defined outliers as values that are +/- 3 standard␣
 ↪deviations
    # from the mean value. To identify such values, we need to compute the Z␣
 ↪score for
    # every value by subtracting the feature-wise mean and dividing by the␣
 ↪feature-wise
    # standard deviation (also known as standardizing the data).
    df = df[df.columns[:-1]]
    Z = (df-df.mean())/df.std()
    # The below code will give a value of True or False for each row. The row␣
 ↪will be
    # True if all of the feature values for that row were within 3 standard␣
 ↪deviations of
    # the mean. The row will be False if at leaset one of the feature values␣
 ↪for that row
    # was NOT within 3 standard deviations of the mean.
    inlier_inds = ((Z > -3).sum(axis=1)==9) & ((Z <= 3).sum(axis=1)==9)
    # The outliers are the inverse boolean values of the above
    outlier_inds = ~inlier_inds
    # print('Found {} rows that were outliers.'.format(outlier_inds.sum()))
    return outlier_inds
```

[4]:
```
# Select only the rows at index locations that were not nans, duplicates, or␣
 ↪outliers
data_clean = data.loc[~((inds_nans(data) | inds_dups(data)) |␣
 ↪inds_outliers(data)),:]
```

```
data_clean
```

[4]:

```
     Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
0                  5                        1                         1
1                  5                        4                         4
2                  3                        1                         1
3                  6                        8                         8
4                  4                        1                         1
..               ...                      ...                       ...
693                3                        1                         1
694                3                        1                         1
696                5                       10                        10
697                4                        8                         6
698                4                        8                         8

     Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
0                    1                            2          1.0
1                    5                            7         10.0
2                    1                            2          2.0
3                    1                            3          4.0
4                    3                            2          1.0
..                 ...                          ...          ...
693                  1                            2          1.0
694                  1                            3          2.0
696                  3                            7          3.0
697                  4                            3          4.0
698                  5                            4          5.0

     Bland Chromatin  Normal Nucleoli  Mitoses  Class
0                  3                1        1      2
1                  3                2        1      2
2                  3                1        1      2
3                  3                7        1      2
4                  3                1        1      2
..               ...              ...      ...    ...
693                2                1        2      2
694                1                1        1      2
696                8               10        2      4
697               10                6        1      4
698               10                4        1      4

[399 rows x 10 columns]
```

Next we normalize the data using the code from Lab 3 so the features will have approximately normal distributions.

[5]:
```
from sklearn import preprocessing

# Normalize the feature columns
data_clean[data_clean.columns[:-1]] = preprocessing.
 ↪normalize(data_clean[data_clean.columns[:-1]], norm='l2')
```

```
/var/folders/wt/gh0jk4cs6ll41pdf91lgjmjh0000gp/T/ipykernel_84377/3197314945.py:4
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_clean[data_clean.columns[:-1]] =
preprocessing.normalize(data_clean[data_clean.columns[:-1]], norm='l2')
```

Split the data into a training and test set with 70% train and 30% test. Use the `seed` variable to set the random state.

[6]:
```
from sklearn.model_selection import train_test_split

# YOUR CODE HERE

X_train, X_test, y_train, y_test = train_test_split(data_clean[data_clean.
 ↪columns[:-1]],
                                                    data_clean[data_clean.
 ↪columns[-1]],
                                                    test_size=0.3,
                                                    random_state=seed)
```

1.1.2 Train a Gaussian Naive Bayes classifier

Use the GaussianNB object in sklearn to fit a Gaussian Naive Bayes classifier and predict the class labels for the test set based on probabilities estimated from the training set.

[7]:
```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

# Fit the model parameters using the training data
gnb = gnb.fit(X_train, y_train)
```

[8]:
```
# Predict the test set classes using the trained model
y_pred_gnb = gnb.predict(X_test)
```

Compute the accuracy of this model on the test set.

[9]:
```
from sklearn.metrics import accuracy_score
# YOUR CODE HERE

print('Accuracy on test data is %.4f' % (accuracy_score(y_test, y_pred_gnb)))
```

```
Accuracy on test data is 0.8667
```

1.1.3 Train a Logistic Regression classifier

Use the LogisticRegression class in sklearn to fit a Logistic Regression classifier and predict the class labels for the test set.

[10]:
```
from sklearn.linear_model import LogisticRegression

# Instantiate a logistic regression classifier and fit it to the training data
lr = LogisticRegression(random_state=seed)
lr = lr.fit(X_train, y_train)
```

[11]:
```
# Predict the test set classes using the trained model

y_pred_lr = lr.predict(X_test) # YOUR CODE HERE
```

Compute the accuracy of this model on the test set.

[12]:
```
# YOUR CODE HERE

print('Accuracy on test data is %.4f' % (accuracy_score(y_test, y_pred_lr)))
```

```
Accuracy on test data is 0.8167
```

1.1.4 Train a k Nearest Neighbors classifier

Use the KNeighborsClassifier class in sklearn to train a kNN classifier and predict the class labels for the test set. We will use Euclidean distance with $k = 5$.

[13]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(X_train, y_train)
```

[13]:
```
KNeighborsClassifier(metric='euclidean')
```

[14]:
```
# Predict the test set classes using the trained model

y_pred_knn = knn.predict(X_test) # YOUR CODE HERE
```

```
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
```

```
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Compute the accuracy of this model on the test set.

[15]:
```
# YOUR CODE HERE

print('Accuracy on test data is %.4f' % (accuracy_score(y_test, y_pred_knn)))
```

```
Accuracy on test data is 0.8333
```

Question 1: What is the test accuracy for each of the 3 models (rounded to 2 decimal places)?

Answer:

YOUR ANSWER HERE

Naive bayes: 0.87, Logistic Regression: 0.82, kNN: 0.83

1.1.5 Creating an ensemble VotingClassifier

We will use the VotingClassifier class in sklearn to combine the predictions of these 3 models using majority vote.

[16]:
```
from sklearn.ensemble import VotingClassifier
```

[17]:
```
ensemble = VotingClassifier(estimators=[('gnb', gnb), ('lr', lr), ('knn',␣
 ↪knn)], voting='soft')

ensemble.fit(X_train, y_train)
```

[17]:
```
VotingClassifier(estimators=[('gnb', GaussianNB()),
                             ('lr', LogisticRegression(random_state=0)),
                             ('knn', KNeighborsClassifier(metric='euclidean'))],
                 voting='soft')
```

Question 2: What is the effect of setting the `voting=soft` vs. `voting=hard` parameter in the VotingClassifier? You can consult the documentation to answer this question.

Answer:

YOUR ANSWER HERE

Voting = hard uses the binary class labels for voting whereas voting = soft uses the class probabilities output by each model for voting.

[18]:
```
for clf, label in zip([gnb, lr, knn, ensemble], ['Naive Bayes', 'Logistic␣
 ↪Regression', 'k Nearest Neighbors', 'Ensemble']):
```

```
    score = accuracy_score(clf.predict(X_test), y_test)
    print("Accuracy: %0.2f [%s]" % (score, label))
```

```
Accuracy: 0.87 [Naive Bayes]
Accuracy: 0.82 [Logistic Regression]
Accuracy: 0.83 [k Nearest Neighbors]
Accuracy: 0.88 [Ensemble]
```

```
/Users/hkerner/anaconda3/envs/cse572/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

1.2 Ensemble using boosting

Another method for ensembling classifiers to improve the performance over any single classifier is using a technique called boosting. Boosting uses an iterative procedure to adaptively change the distribution of the training data by focusing more on previously misclassified samples each time a new classifier is trained. In the second exercise, you will implement the AdaBoost algorithm using a decision tree as the base classifier (the sklearn implementation uses a decision tree by default).

You will use the same Wisconsin breast cancer dataset as in the previous exercise.

1.2.1 Train AdaBoost with decision tree

Use the AdaBoost algorithm to train an ensemble of decision trees. Use 50 trees for the ensemble.

[19]:
```
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(n_estimators=50, random_state=seed)
# Train the model
ada.fit(X_train, y_train)
```

[19]:
```
AdaBoostClassifier(random_state=0)
```

[20]:
```
# Predict the test set classes using the trained model

y_pred_ada = ada.predict(X_test) # YOUR CODE HERE
```

Compute the accuracy of this model on the test set.

[21]:
```
# YOUR CODE HERE

print('Accuracy on test data is %.4f' % (accuracy_score(y_test, y_pred_ada)))
```

```
Accuracy on test data is 0.8917
```