COMP5318 Week 4: Naive Bayes. Model evaluation.

1. Setup

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from scipy import signal

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

#for accuracy_score, classification_report and confusion_matrix
from sklearn import metrics
from sklearn.metrics import accuracy_score

# to make this notebook's output stable across runs
np.random.seed(42)
```

2. Introduction

In this tutorial we will firstly learn how to create a Naive Bayes (NB) classifier in sklearn. Then we will learn how to obtain different performance metrics (precision, recall, F1-score and confusion matrix), how to apply different procedures for evaluating the performance of classifiers (cross-validation and leave-one-out) and finally, how to use grid search with cross-validation for model selection.

3. Creating an NB classifier

There are four main types of NB classifiers in sklearn: **GaussianNB**, **CategoricalNB**, **MultinomialNB** and **BernoulliNB**.

- The first two are the ones we discussed at the lecture - **GaussianNB** is applicable to numeric data, while **CategoricalNB** is applicable to categorical data.
- **BernoulliNB** and **MultinomialNB** are mostly used for text clasification; they assume binary data and count data respectively, e.g. how many times a word appears in a document.

In this tutorial we will create a NB for the iris data which is a numeric dataset, so we will use the **GaussianNB** class to create the classifier.

Let's load the iris data and create the training and test splits:

In [2]:
```
# load the iris dataset
from sklearn.datasets import load_iris
iris = load_iris()

# create the training and test splits
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, stratify=iris.target, random_state=42)
```

Task:

Use GaussianNB from sklearn.naive_bayes to create an NB classifier on the training data and evaluate its acuracy on the test data.

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html)

In [3]:
```
# Solution:

from sklearn.naive_bayes import GaussianNB

# create an NB model using the training set and evaluate its accuracy on the test set
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)
print("Accuracy on test set: {:.3f}".format(accuracy_score(y_test, y_pred)))
```

```
Accuracy on test set: 0.921
```

3. More performance measures: precision, recall and F1 score. Confusion matrix.

In addition to accuracy, we can calculate other performance measures - e.g. precision, recall and their combination - the F1-score. In sklearn this can be convenintly done using the **classification_report** method, which also shows the accuracy.

Task:

1) Continuing on the previous exercise (NB classifier on the iris data), write the Python code to calculate precision, recall, F1 measure and confusion matrix on the test set by using the methods classification_report and confusion_matrix.

See:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

2) Examine the results: -- How are the precision, recall and F1-score calculated - per class or overall for the test set? -- Making sense of the confusion matrix: Where are the correctly classified examples? How many examples from class 1 are incorrectly classified as class 2?

Solution:

1) See the code below

2) Precision, recall and F1-score are calculated per class and the average is also shown (two different version - weighted by the number of examples and not weighted). Recall that there are 3 classes for the iris data: class 0 - setosa, class 1 - versicolor and class 2 - virginica.

The confusion matrix is a 3-by-3 array, where the rows correspond to the true classes and the columns correspond to the predicted class. Thus, in the confusion matrix below (shown at the end in []):

- 1 example from class 1 was incorrectly classified as class 2
- 2 examples from class 2 were incorectly classified as as class 1
- the correctly classified examples are along the main diagonal

In [4]:
```
# calculate other performance measures - F1, precision and recall
#print(nb)
actual = y_test
predicted = nb.predict(X_test)
print(metrics.classification_report(actual, predicted))

# show the confusion matrix
print(metrics.confusion_matrix(actual, predicted))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.86      0.92      0.89        13
           2       0.92      0.85      0.88        13

    accuracy                           0.92        38
   macro avg       0.92      0.92      0.92        38
weighted avg       0.92      0.92      0.92        38

[[12  0  0]
 [ 0 12  1]
 [ 0  2 11]]
```

4. Cross-validation for evaluating performance

Cross-validation, in particular **10-fold stratified cross-validation**, is the standard method in machine learning for evaluating the performance of classification and prediction models. Recall that we are interested in the generalization performance, i.e. how well a classifier will perform on new, previously unseen data.

To perform cross-validation in sklearn, we can use the **cross_val_score** function. It takes as parameters the classifier we would like to evaluate and the data - the feature vectors and the target classes (also caled ground-truth labels). The parameter **cv** specifies the number of folds; the default value is 3, so we need to set it to 10 for 10-fold cross-validation. Note that this function performs **stratified** cross validation for classification tasks.

Let's evaluate our NB classifier using 10-fold cross-validation:

In [5]:
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(nb, iris.data, iris.target, cv=10)
print("Cross-validation scores: {}".format(scores)) #accuracy for each fold
print("Average cross-validation score: {:.2f}".format(scores.mean())) #average accuracy
over all folds
```

```
Cross-validation scores: [0.93333333 0.93333333 1.         0.93333333 0.93333333 0.933
33333
 0.86666667 1.         1.         1.        ]
Average cross-validation score: 0.95
```

The most important result is the average cross-validation score (the average accuracy over the the 10 folds) but it is also useful to look at the accuracy for each fold. In our case, we can see that there is a relatively high variation between the 10 folds - from 86% to 100% accuracy.

Task:

Compare NB's cross-validation accuracy with NB's accuracy when we used a single training/test split. How can you explain the difference? Which is the more reliable measure?

Answer:

- Training/test split - this involves a single data split into training and test set; a classifier is build once, on the training set, and evaluated on the test set
- 10-fold cross validation - this involves building 10 different classifiers, each time using a diferent training and test set(see the lecture notes about how this is done), and then calculating the average accuracy over the 10 runs

Cross validation is a better evaluation procedure than a single training/test evaluation; it gives a more reliable accuracy estimate for the performance on new data.

Leave-one-out cross-validation

Leave-one-out is a special case of cross-validation where each fold is a single example:

In [6]:
```
from sklearn.model_selection import LeaveOneOut
one_out = LeaveOneOut()
scores = cross_val_score(nb, iris.data, iris.target, cv=one_out)
print("Number of evaluations: ", len(scores))
print("Mean accuracy: {:.2f}".format(scores.mean()))
```

```
Number of evaluations:  150
Mean accuracy: 0.95
```

Task:

What are the advantages and disadvantages of leave-one-out?

Answer:

- Advantage: Provides better estimate of the generalization accuracy.
- Disadvantage: Slow, especially for large datasets (number of evaluations = number of examples).

5. Grid search with cross-validation for parameter selection

We can improve the generalization performance of machine learning algorithms by tuning their parameters. NB doesn't have many parameters but in the previous weeks we saw that the performance of k-Nearest Neighbor algorithm depends on the number of neighbors and distance measure type, the performance of regression models depends on the values of **alpha** and **C**.

In sklearn we can use grid search with cross-validation to search through different parameter combinations and select the best one.

Let's consider k-nearest neighbor (k-NN) as an example and tune two of its parameters by considering the following values:

- number of neareast neighbours **n_neighbours** = 1, 3, 5, 11 and 13
- distance measure - Manhattan and Euclidean, which can be controlled by the value of parameter **p**, 1 or 2 respectively

This gives us 5 x 2 combinations of paramneter values. We would like to find the best combination - the one that we expect to generalise well on new examples.

We will use the following procedure, called **grid-search with cross-validation for parameter tuning**:

Pseudocode:

In [ ]:
```
Create the parameter grid (i.e. the parameter combinations)
Split the data into training set and test set
For each parameter combination
    Train a k-NN classifier on the training data using 10-fold cross-validation as an ev
aluation procedure
    Compute the cross-validation accuracy cv_acc
    If cv_acc > best_cv_acc
       best_cv_acc = cv_acc
       best_parameters = current parameters
Rebuild the k-NN model using the whole training data and best_parameters
Evaluate it on the test data and report the results
```

- The data is split into training set and test set
- The cross-validation loop uses the training data. It is performed for every parameter combination. Its purpose is to select the best parameter combination - the one with the highest cross-validation accuracy. This involves, for every parameter combination, building 10 models on 90% of the training data (9 folds) and evaluating them on the remaining 10% (1 fold).
- Once this is done, a new model is trained using the selected best parameter combination on the **whole training set** and evaluated on the test set.

Code for our example:

In [7]:
```
param_grid = {'n_neighbors': [1, 3, 5, 11, 15],
              'p': [1, 2]}
print("Parameter grid:\n{}".format(param_grid))

from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=10,
                           return_train_score=True)


grid_search.fit(X_train, y_train)

print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

```
Parameter grid:
{'n_neighbors': [1, 3, 5, 11, 15], 'p': [1, 2]}
Test set score: 0.95
Best parameters: {'n_neighbors': 11, 'p': 1}
Best cross-validation score: 0.98
Best estimator:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=11, p=1,
                     weights='uniform')

C:\Users\irena\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:814: Dep
recationWarning: The default of the `iid` parameter will change from True to False in
version 0.22 and will be removed in 0.24. This will change numeric results when test-s
et sizes are unequal.
  DeprecationWarning)
```

Explanation: We create an object of type GridSearchCV, which is then fitted to the **training** data. This fitting includes 2 things:

1. Searching for and determining the best parameter combination - the one with the best cross-validation accuracy **and**
2. Building a new model on the whole training set with the best parameter combination from 1.

It is important to understand the difference between **best cross-validation score** and **test set score** :

- **best cross-validation score** is the mean cross-validation accuracy, with cross-validation performed on the **training set**. This step involves building 10 models on the training data, each time using 9 folds together (90% of the training data) to create the model and testing this model the remaining 10th fold (10% of the training data). The purpose of this step is to select the best parameter combination, which is the one with the highest cross-validation accuracy.
- **test set score** - this is the the accuracy on the test of a model that was created using the **whole training set** (100%) with the selected parameters. This is the result that we report as a measure of generalization performance.

Summary

```
In [ ]: nb = GaussianNB().fit(X_train, y_train)
        scores = cross_val_score(nb, iris.data, iris.target, cv=10)
        scores = cross_val_score(nb, iris.data, iris.target, cv=one_out)

        param_grid = {'n_neighbors': [1, 3, 5, 11, 15], 'p': [1, 2]}
        grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=10, return_train_score
        =True)
        grid_search.fit(X_train, y_train)
        print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))
```

Acknowledgements

This tutorial is based on:

Aurelien Geron (2019). Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow, O'Reilly.

Andreas C. Mueller and Sarah Guido (2016). Introduction to Machine Learning with Python: A Guide for Data Scientists, O'Reilly.