Homework 4: Logistic Regression and Decision Trees

Dr. Thomas Gumbsch
thomas.gumbsch@bsse.ethz.ch

Bowen Fan
bowen.fan@bsse.ethz.ch

Dr. Carlos Oliver
carlos.oliver@bsse.ethz.ch

Dr. Sarah Brüningk
sarah.brueningk@bsse.ethz.ch

Prof. Dr. Karsten Borgwardt
karsten.borgwardt@bsse.ethz.ch

Submission deadline: 30.11.2022 at 08:00

Objectives

The goals of this homework are:

- to learn to use the `sklearn` implementation of Logistic Regression and compare its performance to Linear Discriminant Analysis (appendix),
- to understand and implement parts of a decision tree algorithm.

Hints

This homework also provides two skeleton files, `ex1.py` and `ex2.py` that you are supposed to complete in order to solve this homework. Moreover, the sample output contains some hints plus some numbers that you can use to verify your implementation.

Problem overview

This homework investigates two classification methods: Logistic Regression and Decision Trees. In Part 1 you will implement Logistic Regression using Python library `scikit-learn`. In Part 2, you will write Python code for the key step in a decision tree algorithm, namely the computation of the *information gain* for a potential split.

npreg	glu	bp	skin	bmi	ped	age	type
5	86	68	28	30.2	0.364	24	0
7	195	70	33	25.1	0.163	55	1
3	83	58	31	34.3	0.336	25	0
2	128	78	37	43.3	1.224	31	1
0	137	40	35	43.1	2.288	33	1

Table 1: A sample of the diabetes dataset described in [5].

Homework Part 1: Logistic Regression

Introduction

The first part of this homework sheet looks at Logistic Regression [2], which is a classification method for binary output variables. It is closely related to Linear Discriminant Analysis (LDA). You will use the `scikit-learn` [4] implementation in Python to compare the performances of Logistic Regression and LDA on a `diabetes` dataset.

Dataset

The data described in [5] were collected by the US National Institute of Diabetes and Digestive and Kidney Diseases on a population of women living near Phoenix, Arizona, whom were at least 21 years old, of Pima Indian heritage, and whom were tested for diabetes. A sample of the data is shown in Table 1.

In the table, each row represents one patient record, and the first seven columns are the following:

- `npreg`: number of pregnancies
- `glu`: plasma glucose concentration in an oral glucose tolerance test
- `bp`: diastolic blood pressure (in units: mm Hg)
- `skin`: triceps skin fold thickness (mm)
- `bmi`: body mass index (in units: weight in kg/(height in m squared))
- `ped`: diabetes pedigree function
- `age`: age in years

The eighth column, `type`, indicates whether or not the patient is considered to have diabetes according to WHO criteria, with the following values:

- `type = 1` indicates the patient *has* diabetes
- `type = 0` indicates the patient does *not* have diabetes

There are two data files provided:

- `diabetes_train.csv`: a training dataset consisting of 200 samples (rows).
- `diabetes_test.csv`: a dataset consisting of 332 samples (rows) which will be used for testing.

Note: The data files are provided in `CSV` format. It is recommended that the `pandas` module is used to read the data files in Python. The following code accomplishes this task in only a few lines:

```
import numpy as np
import pandas as pd

train_file = 'data/diabetes_train.csv'
#read data from file using pandas
df = pd.read_csv(train_file)

# extract first 7 columns to data matrix X (actually, a numpy ndarray)
X = df.iloc[:, 0:7].values

# extract 8th column (labels) to numpy array
Y = df.iloc[:, 7].values
```

Exercise 1

In this question you will implement Logistic Regression using the Python module `scikit-learn`, also known as `sklearn`, and evaluate its performance on the `diabetes` dataset. Finally, you will compare its performance to the performance of Linear Discriminant Analysis (LDA) on the same dataset (using the results provided in appendix). Please refer to the official documentation[1] to learn more about the logistic regression class of `scikit-learn`.

Exercise 1.a Extend the Python script `ex1.py` to perform a logistic regression using the `LogisticRegression` class of `sklearn`. You are supposed to (i) standardize the data set, using the `StandardScaler` class[2], (ii) train the model using the `diabetes_train.csv` dataset (using a regularization parameter of $C = 1$, which should be the default parameter in `sklearn`), and (iii) use the trained model to compute `y_pred`, the predicted labels of the `diabetes_test.csv` dataset. You may find the following line, placed at the top of your Python script, to be useful:

```
from sklearn.linear_model import LogisticRegression
```

Use the provided function `compute_metrics` to evaluate the performance of logistic regression on the diabetes dataset and print the output to the screen.

Exercise 1.b Compare your results for Logistic Regression with the results for LDA (reproduced at the end of this sheet). If you had to choose between the two methods, which would you choose? Motivate your answer with one sentence.

[1] `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

[2] You should ensure that you `fit` the scaler using the train data set only. It is a common mistake to use both training and test data for this purpose.

Exercise 1.c Perhaps the superior performance of one of the methods is only related to this particular dataset. If you were given a different dataset and asked to choose between LDA and Logistic Regression, which method would you choose? Motivate your answer with one or two sentences.

Exercise 1.d Analyse the coefficients of the logistic regression. Which two attributes appear to contribute most to the prediction? Letting c refer to a coefficient, the value $\exp(c)$ shows the increase (or decrease) in the odds of becoming diabetic when increasing (or decreasing) the variable by one unit. What is the increase or decrease in odds for the `npreg` variable? How much does the risk of getting diabetes increase (in percent)for an additional pregnancy?

Expected output When your script `ex1.py` is run (using `python ex1.py` in a terminal), an output in the format below should be produced:

```
Exercise 1.a
------------
TP: 66
FP: ??
TN: ??
FN: ??
Accuracy: 0.???

Exercise 1.b
------------
For the diabetes dataset I would choose (METHOD X), because...

Exercise 1.c
------------
For another dataset, I would choose (METHOD X), because...

Exercise 1.d
------------
The two attributes which appear to contribute the most to the prediction
are...

The coefficient for npreg is 0.X3. Calculating the exponential function
results in Y.YY, which amounts to an [increase | decrease ] in diabetes
risk of ZZZ percent per additional pregnancy.
```

You can check your implementation by verifying whether you also get 66 true positives. Note that the number of true positives may vary according to the `solver` given as input to `sklearn.linear_model.LogisticRegression()`. If you leave the default value, i.e. `"lbfgs"`, you will get 66 true positives if your implementation is correct.

Sepal length	Sepal width	Petal length	Petal width	Species
5.10	3.50	1.40	0.20	setosa
4.90	3.00	1.40	0.20	setosa
7.00	3.20	4.70	1.40	versicolor
6.40	3.20	4.50	1.50	versicolor
6.30	3.30	6.00	2.50	virginica
5.80	2.70	5.10	1.90	virginica

Table 2: A sample of the Iris flower dataset, originally described in [1].

Homework Part 2: Construction of Decision Trees

Introduction

The second part of this homework sheet looks at the construction of Decision Trees, a classification method that can handle datasets with m class labels, with $m \geq 2$. The key step in a decision tree algorithm is the computation of the splitting criterion, such as the *information gain*. If the threshold θ is used with attribute A to split a dataset $\mathcal{D}$ into two datasets $\{\mathcal{D}_1, \mathcal{D}_2\}$, where

$$\begin{aligned} \mathcal{D}_1 &:= \mathcal{D}_{A,<\theta} = \{\mathbf{x} \in \mathcal{D} | A < \theta\} \\ \mathcal{D}_2 &:= \mathcal{D}_{A,\geq\theta} = \{\mathbf{x} \in \mathcal{D} | A \geq \theta\} \end{aligned} \quad , \tag{1}$$

which is often the case in decision trees, the information gain can be defined by $\text{Gain}(A)$ in Equations (2), (3) and (4):

$$\text{Gain}(A) := \text{Info}(\mathcal{D}) - \text{Info}_A(\mathcal{D}) \tag{2}$$

$$\text{Info}(\mathcal{D}) := -\sum_{i=1}^{m} \text{P}(\mathbf{y} = y_i | \mathbf{x} \in \mathcal{D}) \log_2 \text{P}(\mathbf{y} = y_i | \mathbf{x} \in \mathcal{D}) \tag{3}$$

$$\text{Info}_A(\mathcal{D}) := \sum_{j=1}^{2} \frac{|\mathcal{D}_j|}{|\mathcal{D}|} \text{Info}(\mathcal{D}_j) \tag{4}$$

Dataset

In this exercise, we shall use the Iris flower dataset [1, 3]. This classic dataset was arguably the first dataset to be used by a statistical classification algorithm, namely Fisher's LDA, and has become a standard dataset in the classification literature. A sample of the Iris dataset is shown in Table 2. Each observation in the Iris dataset has four attributes/features: *sepal length*, *sepal width*, *petal length* and *petal width*, which are measurements given in centimetres. There are 150 observations, with 50 observations in each of the three classes (species): *setosa*, *versicolor* and *virginica*.

Loading the Iris dataset

The Iris flower dataset is included in the `sklearn` Python module, and can be loaded using the following code:

```python
from sklearn.datasets import load_iris

# to load the data into X and labels into y
iris = load_iris()
X = iris.data
y = iris.target

# to see feature names and label names:
feature_names = iris.feature_names
target_names = iris.target_names
```

Note: `sklearn` encodes the class labels in the label vector `y` as 0, 1 and 2, but the label names can be accessed using the `target_names` array, defined above.

Exercise 2

Suppose you have a dataset X with attributes (columns) $a_1, a_2, \ldots, a_N$. When constructing a decision tree, we need to work out which attributes would be the best to use for splitting the dataset into smaller datasets. In this question, you are asked to write a Python script to compute the information gain for various splits.

Exercise 2.a Create a Python script named `ex2.py`. Inside this script, write a function called `compute_information_gain(X, y, attribute_index, theta)` to compute the information gain of the split of an attribute for the Iris dataset. Do not use libraries or external source code that calculate the information gain, but implement it following the introduction of this exercise or the lecture slides on Decision Trees. The function should take four arguments: `X`, the Iris data (matrix); `y`, the Iris label vector; `attribute_index`, the attribute/column index (`attribute_index` $\in \{0, 1, 2, 3\}$); and `theta`, which is the split value for the attribute indexed by `attribute_index`. For example, if `attribute_index` $= 3$ and `theta` $= 1.0$, then the split is "Is petal width < 1.0?". The way the Iris data set `X` is arranged by default, as can be seen in Table 2, is as follows:

- `attribute_index` $= 0$: sepal length
- `attribute_index` $= 1$: sepal width
- `attribute_index` $= 2$: petal length
- `attribute_index` $= 3$: petal width

Below is example code showing how the function would be called inside a main script:

```python
attribute_index = 3
theta = 1.0
information_gain = compute_information_gain(X, y, attribute_index, theta)
```

Hint: it may be helpful to write the following functions first (and place them in `ex2.py`):

(i) `split_data(X, y, attribute_index, theta)`, an auxiliary function that splits `X` and `y` into two subsets according to the split defined by the pair (`attribute_index`, `theta`), as above.

(ii) `compute_information_content(y)`, a function to compute the information content of a subset `X` with labels `y`, defined by Equation (3).

(iii) `compute_information_a(X, y, attribute_index, theta)`, a function to compute $\text{Info}_A(X)$ in Equation (4) for a dataset `X` with labels `y` that is split according to the split defined by the pair (`attribute_index`, `theta`).

Exercise 2.b We now want to choose a feature to use for the **first split** of a decision tree for the Iris dataset. Use your function `compute_information_gain(X, y, attribute_index, theta)` contained in the script `ex2.py`, to compute the information gain in Equation (2) for each of the following four splits on the (whole) Iris dataset:

1. sepal length < 5.5,
2. sepal width < 3.0,
3. petal length < 2.0,
4. petal width < 1.0,

Exercise 2.c Given the computed information gains in (2.b), which attribute would you select to construct the first split in the decision tree? Give a reason for your answer.

Exercise 2.d Having calculated the splits manually in the previous exercise, you should now take a look at the classification performance. `scikit-learn` offers the `DecisionTreeClassifier` class[3] from `sklearn.tree` to perform label prediction. Since the dataset does not come with a predefined split, you should also perform *cross-validation* using the `KFold` class from `sklear.model_selection`. More precisely, use `n_splits=5` and `shuffle=True` for the class. Train a decision tree on each fold (using the training data), and perform predictions on the corresponding test data. Use `sklearn.metrics.accuracy_score` to evaluate the results and report the *mean* accuracy (in percent; rounded to two decimal places).

Furthermore, report the `feature_importances_` attribute of each classifier in each fold to obtain the relevance of every feature. Which two features are more important over different folds than others? Having answered this briefly in your output, now *modify* `X` and `y` by removing *all* samples wih `y == 2`. You can do this using the following `numpy` syntax:

```
X = X[y != 2]
y = y[y != 2]
```

[3] See the official documentation `http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html` for more information.

Re-run cross-validation for this reduced dataset and calculate the feature importance again. Which feature is the most important now? What does the importance score imply for the dataset? State your answer briefly in the output.

Output: In order to provide your solution to this exercise, when your script `ex2.py` is run (using `python ex2.py` in a terminal), output in the following format should be produced:

```
Exercise 2.b
------------
Split ( sepal length (cm) < 5.5 ): information gain = 0.55
Split ( sepal width (cm)  < 3.0 ): information gain = 0.25
Split ( petal length (cm) < 2.0 ): information gain = 0.XX
Split ( petal width (cm)  < 1.0 ): information gain = 0.XX

Exercise 2.c
------------
I would select ( XXXXXXXXXX < X.X ) to be the first split,
because...(ONE SENTENCE)

Exercise 2.d
------------
The mean accuracy is XX.33.

For the original data, the two most important features are:
- A
- B

For the reduced data, the most important feature is:
- A
This means that B...
```

Hints: Notice that `cross_val_score` only reports *scores*, i.e. accuracies, for every fold. To obtain feature importances in every fold, you also have to iterate over your cross-validator. Supposing that your cross-validator is called `cv`, this could look as follows:

```
for train_index, test_index in cv.split(X, y):
  X_train, y_train = X[train_index], y[train_index]
  X_test, y_test = X[test_index], y[test_index]

  # Create a classifier, use `fit()` on the `train` data, and
  # `predict()` on the test data.
```

Make sure that you *always* create a new classifier inside the cross-validation loop! Else, you are "spoiling" the results.

Grading and submission guidelines

This homework is worth a total of 100 points. Table 3 shows the points assigned to each exercise. Your submission should consist of a zip file named homework4-lastname.zip, containing ex1.py and ex2.py.

30 pts.	**Exercise 1**	**Contained in file**
10 pts.	Exercise 1.a	`ex1.py`
2 pts.	Exercise 1.b	`ex1.py`
3 pts.	Exercise 1.c	`ex1.py`
15 pts.	Exercise 1.d	`ex1.py`
70 pts.	**Exercise 2**	**Contained in file**
25 pts.	Exercise 2.a	`ex2.py`
10 pts.	Exercise 2.b	`ex2.py`
5 pts.	Exercise 2.c	`ex2.py`
30 pts.	Exercise 2.d	`ex2.py`

Table 3: Grading key for Homework 4

References

[1] E. Anderson. The irises of the Gaspe Peninsula. *Bulletin of the American Iris society*, 59:2–5, 1935.

[2] D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242, 1958.

[3] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications in Medical Care*, pages 261–265. Los Alamitos, CA: IEEE Computer Society Press., 1988.

Appendix: Results for LDA on Diabetes Data Set.

```
# LDA performance on diabetes dataset.

TP: 81
FP: 48
TN: 175
FN: 28
Accuracy: 0.771
```

Acknowledgements

This exercise sheet was created by Dean Bodenham and Karsten Borgwardt and extended by Bastian Rieck.