STK-IN4300 Mandatory Assignment 2

Gabriel Sigurd Cabrera

(Dated: Monday 18th November, 2019)

I. INTRODUCTION

A. Packages

To solve the given exercises, all scripting was performed using `python 3.6.8`; a variety of packages were also used:

- `scikit-learn` [1]
- `pyGAM` [2]
- `Matplotlib` [3]
- `SciPy` [4]
- `Pandas` [5]
- `NumPy` [6]

B. Datasets

1. *Forced Vital Capacity*

For **Problem 1**, the dataset used is available (with some access restrictions) at:

`https://www-adm.uio.no/studier/emner/matnat/math/STK-IN4300/h19/ozone_496obs_25vars.txt`

It consists of 24 features and 248 datapoints (children), with 248 single-labeled continuous outputs. The outputs represent the *forced vital capacity* of a child, based on aspects of their health recorded in a study. These can be seen in TABLE II.

2. *Diabetes*

The dataset for **Problem 2** was obtained at:

`https://www.kaggle.com/uciml/pima-indians-diabetes-database`

This dataset consists of 8 features and 383 datapoints (people), with 383 single-labeled binary outputs. These outputs tell us whether or not a person has diabetes, based on a set of health-related measurements made on each person. These can be seen in TABLE I.

Further information on these datasets is available at `https://www.uio.no/studier/emner/matnat/math/STK-IN4300/h19/obligforside_2.pdf`

Additionally, a variant of this dataset is used in Problem 2, Part 5. Here, the datapoints with invalid values (for example, zeros for BMI) are removed, leaving us with 196 datapoints, rather than 383.

II. PROBLEM 1

Task 1

In order to divide the features and labels ($\mathbf{X}$, $\mathbf{y}$) into a training set ($\mathbf{X}_{\text{train}}$, $\mathbf{y}_{\text{train}}$) and testing set ($\mathbf{X}_{\text{test}}$, $\mathbf{y}_{\text{test}}$), a `python` function was used to both *shuffle* and *split* the dataset into equal parts (a 50-50 split.)

Next, $\mathbf{X}_{\text{train}}$ was centered on its mean value, and scaled to *unit variance*, while $\mathbf{X}_{\text{test}}$ was then centered and scaled with the *training* set's mean and standard deviation. $\mathbf{y}_{\text{train}}$ was then scaled to the range $[0, 1]$, while $\mathbf{y}_{\text{test}}$ was scaled according the same parameters.

Finally, a *bias* column of ones was prepended to $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{test}}$, to account for any bias that may occur in the categorical variables that are underrepresented.

A sample script that performs all these tasks for our given dataset, is shown in FIG 1.

Task 2

Using `scikit-learn`'s `linear_model.LinearRegression` object, a linear regression was performed on the preprocessed arrays $\mathbf{X}_{\text{train}}$ and $\mathbf{y}_{\text{train}}$. By calculating the *standard error*, then integrating over the t-distribution to find their corresponding *p-values*), each coefficent in the *vector of coefficients* was ranked by how strongly they associate with the forced vital capacity.

All these values can be seen in TABLE III – most notably, features `FLGROSS`, `SEX`, and `FLGEW` have the smallest p-values by far, indicating that these are the most significantly correlated features to the forced vital capacity.

Task 3

Backward elimination and forward selection were implemented for stopping criteria $p \geq 0.7$ and $p \geq 0.8$. In TABLE IV, V, VI,VII, we see that backward elimination was more restrictive in selecting features, accepting one feature less than the forward selection for both stopping criteria.

While the selected models for each stopping criterion is certainly different, they appear to overlap. Additionally, they each suggest that exclusively using the three features `FLGROSS`, `SEX`, and `FLGEW` yields the best MSE; in fact the best MSE for three features (9.7452×10^{-3}) is smaller than the MSE when including all features (9.8266×10^{-3}). This is likely due to the fact that considering too many features can lead to significant overfitting.

Task 4

With the help of the `scikit-learn` function `utils.resample`, as well as `model_selection.cross_validate`, a *bootstrap* procedure and *k-nearest-neighbors* cross-validation procedure was used to evaluate the effectiveness of a LASSO regression for one-thousand logarithmically spaced hyperparameters $\alpha \in [10^{-5}, 10^{-0.5}]$.

The results in FIG 2 show that the ideal hyperparameter is in the range of 3.45×10^{-2} and 1.25×10^{-2}, though the latter appears to yield higher results overall when using the bootstrap procedure.

Task 5

A *generalized additive model* (GAM) was created with the `pyGAM` package's `LinearGAM` object, which allows the user to customize the model used on a feature-to-feature basis. By examining the data by hand, it was determined that features 9 (AGEBGEW) and 11 (FLGROSS) were linearly correllated to $\mathbf{y}$, and could therefore be modelled both *linearly* and as a *polynomial.* The other features were approximated using *splines*, due to their irregular natures.

When using a linear model for the correlated features, the resulting MSE was found to be 1.92×10^{-1}; for the polynomial model, the MSE was 1.91×10^{-1}. We can conclude that increasing the model complexity in the aforementioned way does have an effect on the GAM's prediction ability.

Task 6

Using the `scikit-learn` object `ensemble.AdaBoostRegressor`, a boosting procedure was implemented for a linear model, and a decision tree; boosting for a GAM was attempted, but no available packages were found, and `pyGAM` was found to be incompatible with `scikit-learn`'s boosting capabilities.

Boosting yielded noticeable improvements to the vanilla linear regression performed in Task 1; whereas a single linear regression gave us an MSE of 9.8266×10^{-3}, boosting improved this score further by reducing it to 9.5266×10^{-3}. The decision tree boost gave markedly worse results, with an MSE of 1.0904×10^{-2}.

As for the vector of coefficients for the linear regression, this can be found in TABLE VIII.

Task 7

For an overview of the results, all MSE values are shown in TABLE IX. It is apparent that linear regression is the superior model for the given dataset, as the best MSE values are all variants of this method. Ultimately, implementing linear regression with boosting yielded the best fit.

Decision trees were the next best method, with a comparable MSE; this may be a good choice when considering training performance. The remaining methods (GAM and LASSO) performed poorly in comparison to the others, with their scores an order of magnitude larger than linear regression and decision trees.

III. PROBLEM 2

Task 1

The patients in the diabetes dataset were classified using k-nearest-neighbors classification with the `scikit-learn` object `neighbors.KNeighborsClassifier`, with 5-fold cross-validation using the `model_selection.cross_validate` and *leave-one-out* cross-validation with `model_selection.LeaveOneOut`.

This was performed for a number of parameters k, and the results are shown in FIG 3; here, we see that the ideal number of neighbors k seems to be in the range of $k \in [14, 22]$, with a top accuracy of 7.6402×10^{-1}, given by LOO cross-validation.

Task 2

One again, `pyGAM` was used (exclusively with splines) to create a generalized additive model; using forward selection, the cumulative accuracy given by each feature (and those preceding it) is shown in TABLE X. In this table, we see that only including the *top three* features (`glucose`, `mass`, `pressure`) yields the best accuracy score (7.6562×10^{-1}.) Once again, this is likely due to overfitting in the less important features.

Task 3

To implement these five models (classification tree, bagging, random forest, neural network, and AdaBoost), `scikit-learn` has five useful objects that accomplish this:

- `ensemble.BaggingClassifier`

- `ensemble.RandomForestClassifier`
- `ensemble.AdaBoostClassifier`
- `neural_network.MLPClassifier`
- `tree.DecisionTreeClassifier`

The resulting accuracy scores given by these five methods can be seen in TABLE XI – the selected neural network's *hidden layer* configuration is (100, 66, 44). Given these results, it appears that the best method to use when including all features is AdaBoost; it is likely that this is due to the algorithm's ability to reduce overfitting, which was shown to be an issue in Problem 2 Task 2.

Task 4

Although AdaBoost performed well in Task 3, it is worth noting that KNN yielded better results than the classifiers in Task 4, and that omitting all but the three best performing features leads to better accuracy scores as well; overall, this implies that using KNN with 14-22 neighbors would be our best method, since this reduces overfitting and is also computationally effective.

Task 5

Using the trimmed version of the dataset with 196 datapoints, the code can be rerun to gauge the effects of removing invalid datapoints. We see that the performance of our classifiers is significantly improved – this time, we see that implementing AdaBoost yields the absolute best results, with an accuracy score of 7.9082×10^{-1}, seem in TABLE XIII.

Another interesting thing to note is that implementing the forward selection algorithm (results in TABLE XII) shows that only using the most important feature (`pregnant`) is sufficient to predict our outputs, as all subsequent features lead to equal or worse accuracy scores.

IV. APPENDIX

A. Tables

Feature	Description
pregnant	Number of Pregnancies
glucose	Plasma Glucose Concentration
pressure	Diastolic Blood Pressure
triceps	Triceps Skin Fold Thickness
insulin	2-H Serum Insulin
mass	Body Mass Index
pedigree	Diabetes Pedigree Function
age	Age

TABLE I. The features of the *diabetes* dataset.

Feature	Description
ALTER	Age
ADHEU	Allergic Coryza
SEX	Gender
HOCHOZON	High Ozone Village
AMATOP	Maternal Atopy
AVATOP	Paternal Atopy
ADEKZ	Neurodermatitis
ARAUCH	Smoker
AGEBGEW	Birth Weight
FSNIGHT	Night/Morning Cough
FLGROSS	Height
FMILB	Dust Sensitivity
FNOH24	Max. NO2
FTIER	Fur Sensitivity
FPOLL	Pollen Sensitivity
FLTOTMED	No. of Medis/Lufu
FO3H24	24h Max Ozone Value
FSPT	Allergic Reaction
FTEH24	24h Max Temperature
FSATEM	Shortness of Breath
FSAUGE	Itchy Eyes
FLGEW	Weight
FSPFEI	Wheezy Breath
FSHLAUF	Cough

TABLE II. The features of the *forced vital capacity* dataset.

Feature	**Coefficient**	**Standard Error**	**P-Value**
FLGROSS	0.18	0.11	0.09
SEX	-0.10	0.07	0.12
FLGEW	0.07	0.10	0.46
FO3H24	0.06	0.13	0.66
FTEH24	-0.05	0.12	0.67
FPOLL	-0.05	0.13	0.73
FLTOTMED	-0.02	0.05	0.77
FSNIGHT	0.02	0.07	0.78
FTIER	-0.02	0.08	0.79
FNOH24	-0.02	0.10	0.81
FSPFEI	0.02	0.10	0.81
FSHLAUF	-0.01	0.06	0.83
AGEBGEW	0.01	0.07	0.83
ARAUCH	0.01	0.07	0.83
FSPT	0.03	0.16	0.83
ADEKZ	0.01	0.07	0.84
HOCHOZON	-0.02	0.09	0.84
FMILB	-0.02	0.09	0.86
FSAUGE	-0.01	0.07	0.88
ALTER	0.01	0.08	0.90
AMATOP	0.01	0.07	0.94
ADHEU	-0.01	0.07	0.94
FSATEM	0.00	0.09	0.97
AVATOP	-0.00	0.07	1.00

TABLE III. Information on each coefficient in the *vector of coefficients*, for a simple linear regression. $MSE = 9.8266 \times 10^{-3}$.

Feature	**Coefficient**	**Standard Error**	**P-Value**
FLGROSS	0.08	0.11	0.42
SEX	-0.05	0.07	0.47

TABLE IV. Remaining features after backward elimination with stopping criterion $p \geq 0.7$. $MSE = 9.8286 \times 10^{-3}$

Feature	**Coefficient**	**Standard Error**	**P-Value**
FLGROSS	0.08	0.09	0.36
SEX	-0.05	0.06	0.46
FLGEW	0.04	0.09	0.65

TABLE V. Remaining features after forward selection with stopping criterion $p \geq 0.7$. $MSE = 9.7452 \times 10^{-3}$

Feature	**Coefficient**	**Standard Error**	**P-Value**
FLGROSS	0.08	0.11	0.42
SEX	-0.05	0.07	0.47
FLGEW	0.03	0.10	0.73

TABLE VI. Remaining features after backward elimination with stopping criterion $p \geq 0.8$. $MSE = 9.8754 \times 10^{-3}$

Feature	**Coefficient**	**Standard Error**	**P-Value**
FLGROSS	0.08	0.09	0.36
SEX	-0.05	0.06	0.45
FLGEW	0.04	0.09	0.65
FTIER	-0.02	0.06	0.78

TABLE VII. Remaining features after forward selection with stopping criterion $p \geq 0.8$. $MSE = 1.0055 \times 10^{-2}$

Feature	**Coefficient**
ALTER	0
ADHEU	5.91×10^{-3}
SEX	-4.93×10^{-3}
HOCHOZON	-5.87×10^{-2}
AMATOP	-1.27×10^{-2}
AVATOP	6.21×10^{-3}
ADEKZ	-2.72×10^{-2}
ARAUCH	-1.31×10^{-2}
AGEBGEW	5.72×10^{-3}
FSNIGHT	1.93×10^{-2}
FLGROSS	1.91×10^{-2}
FMILB	8.83×10^{-2}
FNOH24	-4.03×10^{-2}
FTIER	-2.35×10^{-2}
FPOLL	-4.57×10^{-2}
FLTOTMED	-5.37×10^{-2}
FO3H24	6.29×10^{-3}
FSPT	-1.53×10^{-2}
FTEH24	7.31×10^{-2}
FSATEM	1.48×10^{-3}
FSAUGE	-6.41×10^{-4}
FLGEW	1.61×10^{-2}
FSPFEI	3.06×10^{-2}
FSHLAUF	2.83×10^{-2}

TABLE VIII. The vector of coefficients for a linear regression boosting model.

Method	**MSE**
Linear Regression (Boosting)	9.5266×10^{-3}
Forward Selection ($p < 0.7$)	9.7452×10^{-3}
Linear Regression	9.8266×10^{-3}
Backward Elimination ($p < 0.7$)	9.8286×10^{-3}
Backward Elimination ($p < 0.8$)	9.8754×10^{-3}
Forward Selection ($p < 0.8$)	1.0055×10^{-2}
Decision Tree (Boosting)	1.0904×10^{-2}
GAM (Polynomial)	1.9103×10^{-1}
GAM (Linear)	1.9198×10^{-1}
LASSO (K-Nearest-Neighbors)	5.7547×10^{-1}
LASSO (Bootstrap)	6.0815×10^{-1}

TABLE IX. The MSE for each method applied in Problem 1, ordered from best to worst performance.

Feature	**Cumulative Accuracy**
glucose	7.2656×10^{-1}
mass	7.6302×10^{-1}
pressure	7.6562×10^{-1}
insulin	7.5521×10^{-1}
triceps	7.6302×10^{-1}
pedigree	7.5781×10^{-1}
pregnant	7.5260×10^{-1}
age	7.2135×10^{-1}

TABLE X. The cumulative accuracies for each feature in the diabetes dataset (ordered from highest p-value to lowest).

Classifier	**Accuracy Score**
ADABoost	7.6042E-01
Random Forest	7.5000E-01
Decision Tree	7.2917E-01
Bagging	7.1875E-01
Neural Network	7.0312E-01

TABLE XI. The accuracy scores for a variety of classification methods, sorted from best to worst.

Feature	**Cumulative Accuracy**
pregnant	7.0408×10^{-1}
glucose	7.0408×10^{-1}
pressure	7.0408×10^{-1}
triceps	7.0408×10^{-1}
insulin	7.0408×10^{-1}
mass	7.0408×10^{-1}
age	7.0408×10^{-1}
pedigree	6.9388×10^{-1}

TABLE XII. The cumulative accuracies for each feature in the diabetes dataset (ordered from highest p-value to lowest). For the filtered dataset.

Classifier	**Accuracy Score**
ADABoost	7.9082×10^{-1}
Random Forest	7.8061×10^{-1}
Bagging	7.6531×10^{-1}
Decision Tree	7.3980×10^{-1}
Neural Network	7.1939×10^{-1}

TABLE XIII. The accuracy scores for a variety of classification methods, sorted from best to worst. For the filtered dataset.

B. Figures

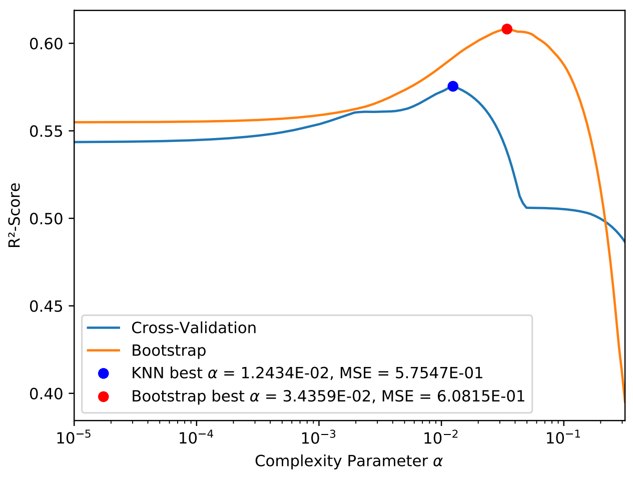

FIG. 1. A performance comparison between k-nearest-neighbors cross-validation and bootstrap, as a function of the complexity parameter α.

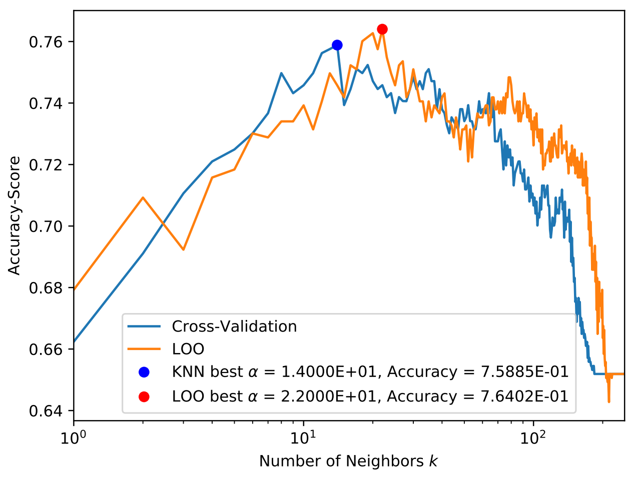

FIG. 2. A performance comparison between k-nearest-neighbors cross-validation and leave-one-out cross-validation, as a function of the number of neighbors k.

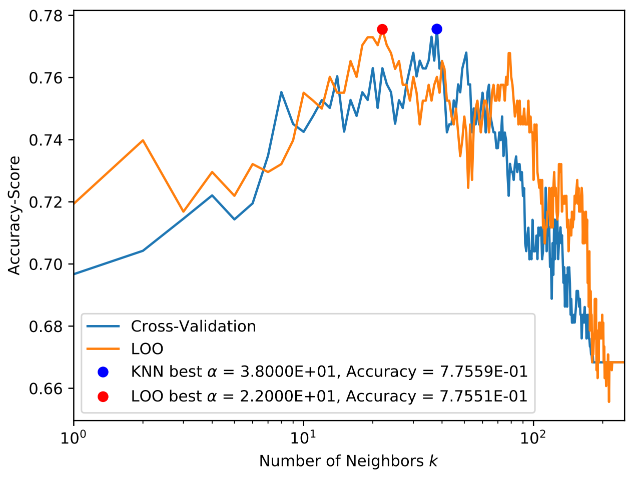

FIG. 3. A performance comparison between k-nearest-neighbors cross-validation and leave-one-out cross-validation, as a function of the number of neighbors k. For the filtered dataset.

C. Source Code

1. Snippets

```
from sklearn.preprocessing import ↩
    PolynomialFeatures
from sklearn.model_selection import ↩
    train_test_split
from sklearn.model_selection import ↩
    cross_validate
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import numpy as np

with open("ozone_496obs_25vars.txt", "r") as ↩
    infile:
    data = infile.readlines()[1:]

X,Y = [],[]

for line in data:
    line = line.split()
    X.append(line[:-1])
    Y.append(line[-1].strip())

X = np.array(X).astype(np.float64)
Y = np.array(Y).astype(np.float64)[:,np.newaxis]

X_train, X_test, Y_train, Y_test = ↩
    train_test_split(X, Y, test_size = 0.5)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

scaler = MinMaxScaler()
Y_train = scaler.fit_transform(Y_train)
Y_test = scaler.transform(Y_test)

poly = PolynomialFeatures(1)
X_test = poly.fit_transform(X_test)
X_train = poly.fit_transform(X_train)
```

FIG. 4. A preprocessing script for the *forced vital capacity* dataset.

2. Complete Source Code

The `python` scripts used to get these results is available on `github`:

Problem 1 `https://github.com/GabrielSCabrera/MachineLearning/blob/master/STK-IN4300/Oblig2/problem1.py`
Problem 2 `https://github.com/GabrielSCabrera/MachineLearning/blob/master/STK-IN4300/Oblig2/problem2.py`

References

[1] "scikit-learn 0.21.2." https://scikit-learn.org.
[2] "pygam 0.8.0." https://pygam.readthedocs.io/en/latest/.
[3] "Matplotlib 3.1.1." https://matplotlib.org/.
[4] "Scipy 1.3.1." https://scipy.org/.
[5] "Pandas 0.24.2." https://pandas.pydata.org.
[6] "Numpy 1.17.4." https://numpy.org/.