Project Work No. 1

1. Study and run the k-means clustering example, written in Python, provided in Lab1.zip. Notice the separation of the functions (modules) and of the main code in different files.

2. Study the way functions can be defined and called in Python. Build your own function to count the numerical, respectively the categorical attributes, in the file cl_date_financiare_tip_polita4.csv, then call this function and display the result on the screen.

References:

[1] Python functions: https://www.w3schools.com/python/python_functions.asp

[2] Python modules: https://www.datacamp.com/tutorial/modules-in-python

[3] Python array functions:
https://www.w3schools.com/python/gloss_python_array_length.asp

[4] Python data types: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dtypes.html

[5] Python tutorial: https://www.w3schools.com/python/

[6] Download Python | Python.org

[7] Installing Anaconda on Windows Tutorial | DataCamp

[8] Installing Python with Anaconda distribution - Preparation for coding : Setting up AI coding environment | Coursera

Project Work No. 2

1. Download the Pima Indians Diabetes Database. Use Python to upload it into a Pandas dataframe. Replace the *0* values with *NaN* values, then replace the missing values (NaN) using the nearest-neighbor imputation strategy.

2. Compute the correlation and covariance matrix for the *Pima Indians Diabetes* dataset, also for the *Iris* dataset.

3. Download and install WEKA, https://waikato.github.io/weka-wiki/downloading_weka/ . Use the specific Weka functions (CFS, Relief, IGA) to select the relevant features, based on the *Iris* and *Pima Indians Diabetes* datasets.

4. Use at least two specific Python functions, e.g. *SelectKBest*, with various parameter values, to select relevant features from the *Iris* and *Pima Indians Diabetes* datasets.

References:

[1] PIMA Indians Diabetes Database, https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database?resource=download

[2] Python Imputation Methods, https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html

[3] Python Correlation Matrix, https://www.geeksforgeeks.org/create-a-correlation-matrix-using-python/

[4] Python Covariance Matrix, https://numpy.org/doc/2.0/reference/generated/numpy.cov.html#numpy.cov

[5] Waikato Environment for Knowledge Analysis (WEKA), https://ml.cms.waikato.ac.nz/index.html

[6] Python feature selection functions, https://scikit-learn.org/stable/api/sklearn.feature_selection.html

Project Work No. 3

1. Employing Python specific functions, apply the Principal Component Analysis (PCA) and Kernel Principal Component Analysis (KPCA) techniques in Python, upon both Pima Indians Diabetes and Digits datasets. Display the newly resulted data matrices. Represent graphically the first two, respectively the first three principal components.

2. In the WEKA environment, apply the Principal Component Analysis (PCA) technique upon both the Iris and Pima Indians Diabetes datasets. Visualize the result in the environment.

3. Employing Python specific functions, apply k-means clustering and X-means clustering upon the Iris dataset. Label the instances with the labels of the newly discovered clusters. Then, apply feature selection techniques to determine the most relevant features that separate between clusters.

4. In the WEKA environment, apply k-means clustering and Expectation-Maximization based clustering upon the Iris dataset. Visualize the result in both cases.

5. Using Python specific functions, employ k-modes and k-prototype clustering upon the Vintages Industrial Production dataset. Label the instances with the labels of the newly discovered clusters.

6. Using Python specific functions, employ the *Apriori* algorithm for association rules to discover dependencies between the attributes belonging to both Pima Indians Diabetes dataset, respectively to the Vintages Industrial Production dataset. Visualize the result.

References:

[1] Principal Component Analysis in Python: https://builtin.com/machine-learning/pca-in-python

[2] Kernel PCA in Python: https://scikit-learn.org/stable/auto_examples/decomposition/plot_kernel_pca.html

[3] 3D Plots in Python: https://matplotlib.org/stable/gallery/mplot3d/scatter3d.html#sphx-glr-gallery-mplot3d-scatter3d-py

[4] K-means clustering in Python:

https://scikit-learn.org/1.5/modules/generated/sklearn.cluster.KMeans.html

[5] X-means clustering:

https://pyclustering.github.io/docs/0.10.1/html/dd/db4/classpyclustering_1_1cluster_1_1xmeans_1_1xmeans.html

[6] K-means, k-mode and k-prototype clustering in Python:

https://medium.com/@reddyyashu20/k-means-kmodes-and-k-prototype-76537d84a66

[7] Association rules in Python:

https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/

Project Work No. 4

1. Employ simple linear regression in Python for finding a linear relationship between *Petal Width* and *Petal Length* within the Iris dataset. Perform the graphical representation of the training set points and regression line. Try to predict the output considering an arbitrary value as input. Try to do the same tasks by employing polynomial regression.

2. Employ multiple linear regression in Python for finding a linear relationship between the *Glucose* and *Blood Pressure* as independent variables (X1 and X2), respectively the *Diabetes Pedigree Function* as the dependent variable (y), within the Pima Indians Diabetes Dataset. Perform the graphical representation of the training set points and regression line. Try to predict the outputs considering arbitrary values of the input data.

3. Employ linear regression in Weka considering both the Iris and the Pima Indians Diabetes datasets. Remove the class attribute from the Iris dataset. Study the linear dependency between the variables.

4. Employ logistic regression in Python for predicting the *outcome* (values 0 or 1) as a function of the *Glucose* level, within the Pima Indians Diabetes Dataset. Try to predict the output considering an arbitrary value of the input data. Display also the probability of Diabetes (outcome=1 yes; outcome=0 no) as a function of all the *Glucose* values in the dataset.

5. Employ logistic regression in Weka considering both the Iris and the Pima Indians Diabetes datasets. Within the Pima Indians Diabetes dataset rename the *outcome* as *class*. Study the provided output.

References:

[1] Simple Linear Regression in Python:
https://www.w3schools.com/python/python_ml_linear_regression.asp

[2] Polynomial Regression in Python:

https://www.w3schools.com/python/python_ml_polynomial_regression.asp

[3] Multiple Regression in Python:

https://www.w3schools.com/python/python_ml_multiple_regression.asp

[4] Logistic regression in Python:

https://www.w3schools.com/python/python_ml_logistic_regression.asp

Project Work No. 5

1. Employ the supervised classifiers k-nn, Multilayer Perceptron (MLP), Support Vector Machines (SVM) and Decision Trees with various parameters using specific *python* functions. Regarding the input, use both Pima Indians Diabetes Database and Iris datasets. Provide arbitrary values at the classifiers' inputs and predict the corresponding classes. Also display the means and standard deviations of the accuracy values, resulted through cross-validation. Experiment as well the Weka versions of these classifiers on the same datasets.

2. Employ non-parametric, non-linear regression in *python* through the k-nn, Support Vector Machines (SVM), Multilayer Perceptron (MLP) and Decision Trees (DT) regressors. Provide arbitrary values at the classifiers' inputs and predict the values of the dependent variables. In each case, assess the quality of the regression process through the *r2 score*, also by performing graphical representations. Employ the Pima Indians Diabates Database, Iris and Cars datasets.

References:

[1] The k-nn classifier in Python, https://www.w3schools.com/python/python_ml_knn.asp

[2] K-nn regression in Python,

https://scikit-learn.org/1.5/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbos.KNeighborsRegressor

[3] The MLP classifier in Python,

https://scikit-learn.org/dev/modules/generated/sklearn.neural_network.MLPClassifier.html

[4] The MLP regressor in Python,

https://scikit-learn.org/dev/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor

[5] Support Vector Machines classification/regression in Python,

https://scikit-learn.org/1.5/modules/svm.html

Project Work No. 6

1. Employ, in Python, the bagging combination scheme, with various basic classifiers (weak learners), such as Support Vector Classifier (SVC), k-nn and Decision Trees, on the Pima Indians Diabetes Database dataset. Assess the accuracy of the individual weak classifiers, as well as that of the bagging ensemble, in each case.

2. Implement in Python the AdaBoost metaclassifier, on both the Iris and Pima Indians Diabetes Database datasets. Vary the basic learner type (Decision Trees or Support Vector Machines). Assess the accuracies in each case.

3. Employ, in Python, the Stacking combination scheme, with multiple basic classifiers, such as Support Vector Classifier (SVC), k-nn, Decision Trees and Multilayer Perceptron (MLP). on the Pima Indians Diabetes Database dataset. Assess the accuracy of the individual weak classifiers, as well as that of the stacking meta-classifier, in each case.

4. Implement, in Python. both soft voting (by computing the arithmetic mean of the output probabilities of the individual classifiers) and hard (majority) voting, using multiple types of basic classifiers (weak learners), such as Support Vector Classifier (SVC), k-nn, Decision Trees and Multilayer Perceptron (MLP). Assess the accuracy of the individual weak classifiers, as well as that of the voting combination scheme, in each case.

5. Read about the Random Forest classifier and regressor at https://mitu.co.in/wp-content/uploads/2022/04/Ensemble-Learning.pdf. Employ Random Forest classification in Python, upon the Iris dataset, then assess the corresponding accuracy. Employ also Random Forest regression, to predict the sepal width, as a function of the other attributes.

6. Experiment, as well, the Weka 3. versions of all these classifier combination schemes/metaclassifiers.

Bibliographic references

[1] Bagging in Python,
https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.BaggingClassifier.html

[2] AdaBoost in Python,
https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[3] Stacking in Python,
https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.StackingClassifier.html

[4] RandomForest classification in Python,
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[5] RandomForest regression in Python,
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor

Project Work No. 7

1. In Python, with the aid of the *Torchvision* library, train the ResNext CNN (already pretrained on the ImageNet dataset) on the CIFAR10 dataset and test it on the same dataset. Use the following splitting pattern: 75% train set, 25% test set.

2. In both Python and Matlab, train the ResNet and InceptionV3 CNNs, which are already trained on the ImageNet dataset, on the CalTech101 dataset. Test both networks on the same dataset. Use the following splitting pattern: 70% train set, 30% test set.

   [1] The CIFAR10 dataset in Python:
   https://pytorch.org/vision/stable/generated/torchvision.datasets.CIFAR10.html
   [2] The CalTech101 dataset in Python:
   https://pytorch.org/vision/stable/generated/torchvision.datasets.Caltech101.html#torchvision.datasets.Caltech101
   [3] ResNet in Python:
   https://pytorch.org/hub/pytorch_vision_resnet/
   [4] InceptionV3 in Python:
   https://pytorch.org/hub/pytorch_vision_inception_v3/
   [5] ResNext in Python
   https://pytorch.org/vision/main/models/resnext.html
   [6] Matlab, Deep Learning toolbox
   https://www.mathworks.com/products/deep-learning.html
   [7] Matlab, Deep Network Designer
   https://www.mathworks.com/help/deeplearning/ref/deepnetworkdesigner-app.html

Project Work No. 8

Employ a Long-Short Term Memory (LSTM) RNN in Python, for performing predictions:

(a.) Predict the milk production as a function of the production date, based on the *monthly milk production* dataset;

(b.) Predict the age as a function of DiabetesPedigreeFunction, based on the *Pima Indians Diabetes* dataset.

[1] Long-Short Term Memory (LSTM) RNN in Tensorflow, https://www.geeksforgeeks.org/long-short-term-memory-lstm-rnn-in-tensorflow/

[2] tensorflow - Keras LSTM: how to predict beyond validation vs predictions? - Stack Overflow

Project Work No. 9

1. Employ the Support Vector Machines and Multilayer Perceptron in Python, to perform classification on the *Iris* and *Pima Indians Diabetes* datasets.
   (a.) Compute the confusion matrix, as well as the following classification performance assessment metrics in each case: accuracy, precision, recall (sensitivity), specificity, AuC. Perform graphical representations of the ROC curves in each case, as well.
   (b.) Perform relevant feature selection on these datasets, as well, by employing the Recursive Feature Elimination (RFE) method in Python. Thereafter, reassess the classification performance, by re-computing the above mentioned metrics.
   (c.) Perform Principal Component Analayis (PCA) on these datasets, as well, by employing the appropriate method in Python. Thereafter, reassess the classification performance, by re-computing the above-mentioned metrics.

2. (a.) Employ simple linear regression in Python, upon the Iris dataset, for predicting the *petal width* as a function of the *petal length*. Compute the regression assessment metrics: MAE, MSE, RMSE, R^2.
   (b.) Continue exercise 1 (a.) from Project Work No. 8, by computing the MAE, MSE and RMSE assessment metrics.

[1] Confusion Matrix in Python, https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html

[2] Classification performance assessment metrics in Python, https://medium.com/@maxgrossman10/accuracy-recall-precision-f1-score-with-python-4f2ee97e0d6

[3] ROC in Python, https://scikit-learn.org/dev/modules/generated/sklearn.metrics.roc_curve.html

[4] AuC in Python, https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.auc.html

[5] ROC curve graphical representation in Python, https://scikit-learn.org/0.16/auto_examples/model_selection/plot_roc.html ş

[6] Python feature selection functions, https://scikit-learn.org/stable/api/sklearn.feature_selection.html

[7] Principal Component Analysis in Python, https://builtin.com/machine-learning/pca-in-python

[8] MAE in Python, https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.mean_absolute_error.html

[9] MSE in Python, https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.mean_squared_error.html

[10] RMSE in Python, https://www.askpython.com/python/examples/rmse-root-mean-square-error

[11] R^2 Score in Python, https://www.geeksforgeeks.org/python-coefficient-of-determination-r2-score/