**ITP4514 – Artificial Intelligence and Machine Learning**

**Lab 6 – Machine Learning: Classification**

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Student ID: \_\_\_\_\_\_\_\_\_\_\_ Class: \_\_\_\_\_\_

**Intended Learning Outcomes**

Upon completion of this tutorial/lab, you should be able to:

* Using Colab Notebook to load datasets and run AI Classification algorithms.
* Application and comparison of different classifiers provided by scikit-learn to classify a simulated dataset.
* Application of a classifier, e.g. K-Nearest Neighbour to specific datasets, e.g. iris and other datasets.
* Evaluate a classifier in an application, e.g. Support Vector Classifier in recognizing hand-written digits, by using the confusion matrix.
* Simulate a 4-class classification problem and use a classifier, e.g. Logistic Regression, to solve the problem.

**------------------------------------------------------------------------------------------**

Please use the 2 datasets provided.

Please submit the **.ipynb** file and this **doc**(with screen captures and your answers) to moodle.

**Q1.** **Classifiers comparison:**

[Reference:

<https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py> ]

The following code implements different classification algorithms, including K-Nearest Neighbours, Gaussian Naïve Bayes, Decision Tree, Support Vector Machine, Random Forest, AdaBoost etc., to a dataset generated and make comparison. Run the codes and capture the outputs.

|  |
| --- |
| print(\_\_doc\_\_)  # Code source: Gaël Varoquaux  # Andreas Müller  # Modified for documentation by Jaques Grobler  # License: BSD 3 clause  import numpy as np  import matplotlib.pyplot as plt  from matplotlib.colors import ListedColormap  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from sklearn.datasets import make\_moons, make\_circles, make\_classification  from sklearn.neural\_network import MLPClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.svm import SVC  from sklearn.gaussian\_process import GaussianProcessClassifier  from sklearn.gaussian\_process.kernels import RBF  from sklearn.tree import DecisionTreeClassifier  from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier  from sklearn.naive\_bayes import GaussianNB  from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis  h = .02 # step size in the mesh  names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process",  "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",  "Naive Bayes", "QDA"]  classifiers = [  KNeighborsClassifier(3),  SVC(kernel="linear", C=0.025),  SVC(gamma=2, C=1),  GaussianProcessClassifier(1.0 \* RBF(1.0)),  DecisionTreeClassifier(max\_depth=5),  RandomForestClassifier(max\_depth=5, n\_estimators=10, max\_features=1),  MLPClassifier(alpha=1, max\_iter=1000),  AdaBoostClassifier(),  GaussianNB(),  QuadraticDiscriminantAnalysis()]  X, y = make\_classification(n\_features=2, n\_redundant=0, n\_informative=2,  random\_state=1, n\_clusters\_per\_class=1)  rng = np.random.RandomState(2)  X += 2 \* rng.uniform(size=X.shape)  linearly\_separable = (X, y)  datasets = [make\_moons(noise=0.3, random\_state=0),  make\_circles(noise=0.2, factor=0.5, random\_state=1),  linearly\_separable  ]  figure = plt.figure(figsize=(27, 9))  i = 1  # iterate over datasets  for ds\_cnt, ds in enumerate(datasets):  # preprocess dataset, split into training and test part  X, y = ds  X = StandardScaler().fit\_transform(X)  X\_train, X\_test, y\_train, y\_test = \  train\_test\_split(X, y, test\_size=.4, random\_state=42)  x\_min, x\_max = X[:, 0].min() - .5, X[:, 0].max() + .5  y\_min, y\_max = X[:, 1].min() - .5, X[:, 1].max() + .5  xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),  np.arange(y\_min, y\_max, h))  # just plot the dataset first  cm = plt.cm.RdBu  cm\_bright = ListedColormap(['#FF0000', '#0000FF'])  ax = plt.subplot(len(datasets), len(classifiers) + 1, i)  if ds\_cnt == 0:  ax.set\_title("Input data")  # Plot the training points  ax.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cm\_bright,  edgecolors='k')  # Plot the testing points  ax.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap=cm\_bright, alpha=0.6,  edgecolors='k')  ax.set\_xlim(xx.min(), xx.max())  ax.set\_ylim(yy.min(), yy.max())  ax.set\_xticks(())  ax.set\_yticks(())  i += 1  # iterate over classifiers  for name, clf in zip(names, classifiers):  ax = plt.subplot(len(datasets), len(classifiers) + 1, i)  clf.fit(X\_train, y\_train)  score = clf.score(X\_test, y\_test)  # Plot the decision boundary. For that, we will assign a color to each  # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].  if hasattr(clf, "decision\_function"):  Z = clf.decision\_function(np.c\_[xx.ravel(), yy.ravel()])  else:  Z = clf.predict\_proba(np.c\_[xx.ravel(), yy.ravel()])[:, 1]  # Put the result into a color plot  Z = Z.reshape(xx.shape)  ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)  # Plot the training points  ax.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cm\_bright,  edgecolors='k')  # Plot the testing points  ax.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap=cm\_bright,  edgecolors='k', alpha=0.6)  ax.set\_xlim(xx.min(), xx.max())  ax.set\_ylim(yy.min(), yy.max())  ax.set\_xticks(())  ax.set\_yticks(())  if ds\_cnt == 0:  ax.set\_title(name)  ax.text(xx.max() - .3, yy.min() + .3, ('%.2f' % score).lstrip('0'),  size=15, horizontalalignment='right')  i += 1  plt.tight\_layout()  plt.show() |
| ***Output*** |

**Q2.** **Decision Tree:**

[Reference: (modified from)

<https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm> ]

The following codes implements K-Nearest Neighbour classification to the iris dataset.(given)

(a) Run the code and capture the output.

|  |
| --- |
| # importing modules  import numpy as np  import pandas as pd  import tensorflow as tf  import matplotlib.pyplot as plt  from sklearn.preprocessing import OneHotEncoder |
| from google.colab import files  uploaded=files.upload() # browse and upload dataset.csv |
| dd = pd.read\_csv("dataset.csv") |
| print(dd.head())  print("Data Shape: ",dd.shape) |
| ***Output:*** |
| #Extracting Independent and dependent Variable  x= dd.iloc[:, [0,1]].values  y= dd.iloc[:,3].values  #print(x)  #print(y) |
| # Splitting the dataset into training and test set.  from sklearn.model\_selection import train\_test\_split  x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)    #feature Scaling  from sklearn.preprocessing import StandardScaler  st\_x= StandardScaler()  x\_train= st\_x.fit\_transform(x\_train)  x\_test= st\_x.transform(x\_test)  #print(x\_test,y\_test) |
| #Fitting Decision Tree classifier to the training set  from sklearn.tree import DecisionTreeClassifier  classifier= DecisionTreeClassifier(criterion='entropy', random\_state=0) |
| classifier.fit(x\_train, y\_train) |
| ***Output:*** |
| #Predicting the test set result  y\_pred= classifier.predict(x\_test)  print(x\_test,y\_pred) |
| ***Output:*** |
| #Creating the Confusion matrix  from sklearn.metrics import confusion\_matrix  cm= confusion\_matrix(y\_test, y\_pred)  print(cm) |
| ***Output:*** |
| ***Comment on the confusion matrix obtained:*** |
| import numpy as nm  import matplotlib.pyplot as mtp |
| #Visulaizing the trianing set result  from matplotlib.colors import ListedColormap  x\_set, y\_set = x\_train, y\_train  x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),  nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))  mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),  alpha = 0.75, cmap = ListedColormap(('purple','green' )))  mtp.xlim(x1.min(), x1.max())  mtp.ylim(x2.min(), x2.max())  for i, j in enumerate(nm.unique(y\_set)):  mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1], c = ListedColormap(('purple', 'green'))(i), label = j)  mtp.title('Decision Tree Algorithm (Training set)')  mtp.xlabel('Age')  mtp.ylabel('Estimated Salary')  mtp.legend()  mtp.show() |
| ***1120***  ***Output:*** |
| #Visulaizing the test set result  from matplotlib.colors import ListedColormap  x\_set, y\_set = x\_test, y\_test  x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),  nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))  mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),  alpha = 0.75, cmap = ListedColormap(('purple','green' )))  mtp.xlim(x1.min(), x1.max())  mtp.ylim(x2.min(), x2.max())  for i, j in enumerate(nm.unique(y\_set)):  mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1], c = ListedColormap(('purple', 'green'))(i), label = j)  mtp.title('Decision Tree Algorithm(Test set)')  mtp.xlabel('Age')  mtp.ylabel('Estimated Salary')  mtp.legend()  mtp.show() |
| ***Output:*** |

**Q3.** **K-Nearest Neighbour:**

[Reference:

<https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py> ]

The following codes implements K-Nearest Neighbour classification to the iris dataset.(given)

(a) Run the code and capture the output.

|  |
| --- |
| print(\_\_doc\_\_)  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  from matplotlib.colors import ListedColormap  from sklearn import neighbors, datasets  n\_neighbors = 15  # import some data to play with  iris = datasets.load\_iris()  # we only take the first two features. We could avoid this ugly  # slicing by using a two-dim dataset  X = iris.data[:, :2]  y = iris.target  h = .02 # step size in the mesh  # Create color maps  cmap\_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])  cmap\_bold = ['darkorange', 'c', 'darkblue']  for weights in ['uniform', 'distance']:  # we create an instance of Neighbours Classifier and fit the data.  clf = neighbors.KNeighborsClassifier(n\_neighbors, weights=weights)  clf.fit(X, y)  # Plot the decision boundary. For that, we will assign a color to each  # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].  x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),  np.arange(y\_min, y\_max, h))  Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])  # Put the result into a color plot  Z = Z.reshape(xx.shape)  plt.figure(figsize=(8, 6))  plt.contourf(xx, yy, Z, cmap=cmap\_light)  # Plot also the training points  sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=iris.target\_names[y],  palette=cmap\_bold, alpha=1.0, edgecolor="black")  plt.xlim(xx.min(), xx.max())  plt.ylim(yy.min(), yy.max())  plt.title("3-Class classification (k = %i, weights = '%s')"  % (n\_neighbors, weights))  plt.xlabel(iris.feature\_names[0])  plt.ylabel(iris.feature\_names[1])  plt.show() |
| ***Output:*** |

(b) *\*\*Challenge (Try this after you have completed other parts!)*

Apply the code to another dataset, e.g. the iris.csv dataset (given*).*

Predict the Iris type for new sample e.g. sample 151, with petal and sepal data [6.5, 3.2, 5.1, 2.4]. Show the data, the change in codes and the output.



*Hints:*

*Inspect iris.csv and compare it with datasets.csv as below;*

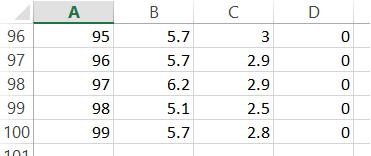
*Amend the codes as above example;*

*No need to plot.*

**dataset.csv**

(100 rows \* 4 columns;

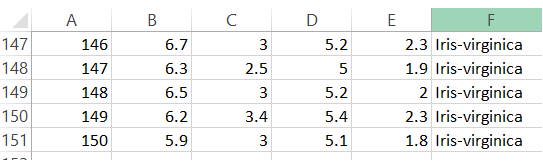
input columns **1 to 2** -> output: column **4**)



**iris.csv**

(151 rows \* 6 columns;

input columns **2 to 5** -> output column **6**)



Codes changed / OR the complete code used

|  |
| --- |
|  |
| ***Predict the Iris type for new sample:*** |

**Q4.** **Support Vector Classifier:**

[Reference:

<https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#sphx-glr-auto-examples-classification-plot-digits-classification-py> ]

The following is the implementation of Support Vector Classifier in recognizing hand-written digits.

(a) Run the codes and capture the output.

(b) From the confusion matrix, is the classifier performing well? Explain.

|  |
| --- |
| print(\_\_doc\_\_)  # Author: Gael Varoquaux <gael dot varoquaux at normalesup dot org>  # License: BSD 3 clause  # Standard scientific Python imports  import matplotlib.pyplot as plt  # Import datasets, classifiers and performance metrics  from sklearn import datasets, svm, metrics  from sklearn.model\_selection import train\_test\_split |
| digits = datasets.load\_digits()  \_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))  for ax, image, label in zip(axes, digits.images, digits.target):  ax.set\_axis\_off()  ax.imshow(image, cmap=plt.cm.gray\_r, interpolation='nearest')  ax.set\_title('Training: %i' % label) |
| ***Output:*** |
| # flatten the images  n\_samples = len(digits.images)  data = digits.images.reshape((n\_samples, -1))  # Create a classifier: a support vector classifier  clf = svm.SVC(gamma=0.001)  # Split data into 50% train and 50% test subsets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  data, digits.target, test\_size=0.5, shuffle=False)  # Learn the digits on the train subset  clf.fit(X\_train, y\_train)  # Predict the value of the digit on the test subset  predicted = clf.predict(X\_test) |
| \_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))  for ax, image, prediction in zip(axes, X\_test, predicted):  ax.set\_axis\_off()  image = image.reshape(8, 8)  ax.imshow(image, cmap=plt.cm.gray\_r, interpolation='nearest')  ax.set\_title(f'Prediction: {prediction}') |
| ***Output:*** |
| print(f"Classification report for classifier {clf}:\n"  f"{metrics.classification\_report(y\_test, predicted)}\n") |
| ***Output:*** |
| disp = metrics.plot\_confusion\_matrix(clf, X\_test, y\_test)  disp.figure\_.suptitle("Confusion Matrix")  print(f"Confusion matrix:\n{disp.confusion\_matrix}")  plt.show() |
| ***Output:*** |
| ***From the confusion matrix, is the classifier performing well? Explain.*** |

**Q5.** **Logistic Regression**:

[Reference: (modified from)

<https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_multinomial.html#sphx-glr-auto-examples-linear-model-plot-logistic-multinomial-py> ]

The following is the implementation of multinomial logistic regression for a simulated 3-classes classification problem.

(a) Run the codes and capture the output.

|  |
| --- |
| print(\_\_doc\_\_)  # Authors: Tom Dupre la Tour <tom.dupre-la-tour@m4x.org>  # License: BSD 3 clause  import numpy as np  import matplotlib.pyplot as plt  from sklearn.datasets import make\_blobs  from sklearn.linear\_model import LogisticRegression  # make 3-class dataset for classification  centers = [[-5, 0], [0, 1.5], [5, -1]]  X, y = make\_blobs(n\_samples=1000, centers=centers, random\_state=40)  transformation = [[0.4, 0.2], [-0.4, 1.2]]  X = np.dot(X, transformation)  clf = LogisticRegression(solver='sag', max\_iter=100, random\_state=42).fit(X, y)  # print the training scores  print("training score : %.3f (%s)")  # create a mesh to plot in  h = .02 # step size in the mesh  x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))  # Plot the decision boundary. For that, we will assign a color to each  # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].  Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])  # Put the result into a color plot  Z = Z.reshape(xx.shape)  plt.figure()  plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)  plt.title("Decision surface of LogisticRegression (%s)")  plt.axis('tight')  # Plot also the training points  colors = "bry"  for i, color in zip(clf.classes\_, colors):  idx = np.where(y == i)  plt.scatter(X[idx, 0], X[idx, 1], c=color, cmap=plt.cm.Paired, edgecolor='black', s=20)  # Plot the three one-against-all classifiers  xmin, xmax = plt.xlim()  ymin, ymax = plt.ylim()  coef = clf.coef\_  intercept = clf.intercept\_  def plot\_hyperplane(c, color):  def line(x0):  return (-(x0 \* coef[c, 0]) - intercept[c]) / coef[c, 1]  plt.plot([xmin, xmax], [line(xmin), line(xmax)], ls="--", color=color)  for i, color in zip(clf.classes\_, colors):  plot\_hyperplane(i, color)  plt.show() |
| ***Output:*** |

(b) \*\*Modify the codes to simulate a **4-class** classification problem. Amend the codes accordingly and apply the multinomial logistic regression classification to the problem.

Codes

|  |
| --- |
|  |
| ***Output:*** |

**\*\*\* End of Lab 6 \*\*\***