APPLIED MACHINE LEARNING
LAB ACTIVITIES (LAB 4)

WEEK 4, TERM 2 2023-24

COMPARE MACHINE LEARNING ALGORITHMS

This workbook is designed to guide you through the activities proposed for today's lab. As you will be working independently, feel free to proceed through the text at your own pace, spending more time on the parts that are less familiar to you. The workbook contains both hands-on tasks and links to learning materials such as tutorials, articles and videos. When you are unsure about something, feel free to ask your group teaching assistant or use Internet resources to look for a solution. At the end of each section, there will be questions and exercises to verify your understanding of the presented information. You may need to do some research to answer the questions.

1. Evaluation Metrics

The metrics that you choose to evaluate your machine learning algorithms are very important. Choice of metrics influences how the performance of machine learning algorithms is measured and compared. They influence how you weight the importance of different characteristics in the results and your ultimate choice of which algorithm to choose.

In this lab, various algorithm evaluation metrics are demonstrated for classification type machine learning problems. All recipes evaluate the same algorithms, Logistic Regression. A 10-fold cross-validation test harness is used to demonstrate each metric, because this is the most likely scenario you will use when employing different algorithm evaluation metrics.

A caveat in these recipes is the `cross_validation.cross val score` function used to report the performance in each recipe. It does allow the use of different scoring metrics that will be discussed, but all scores are reported so that they can be sorted in ascending order (largest score is best). Some evaluation metrics (like mean squared error) are naturally descending scores (the smallest score is best) and as such are reported as negative by the `cross validation.cross val score()` function. This is important to note, because some scores will be reported as negative that by definition can never be negative. I will remind you about this caveat as we work through the lab.

You can learn more about machine learning algorithm performance metrics supported by scikit-learn on the page Model evaluation: quantifying the quality of predictions. See below.

https://scikit-learn.org/stable/modules/model_evaluation.html

Classification Metrics

Classification problems are perhaps the most common type of machine learning problem and as such there is a myriad of metrics that can be used to evaluate predictions for these problems. In this section we will review how to use the following metrics:

- Classification Accuracy.
- Logarithmic Loss.
- Area Under ROC Curve.
- Confusion Matrix.
- Classification Report.

Classification Accuracy. Classification accuracy is the number of correct predictions made as a ratio of all predictions made. This is the most common evaluation metric for classification

problems, it is also the most misused. It is really only suitable when there are an equal number of observations in each class (which is rarely the case) and that all predictions and prediction errors are equally important, which is often not the case. Below is an example of calculating classification accuracy.

```
# Cross Validation Classification Accuracy
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7,shuffle=True)
model = LogisticRegression(solver='liblinear')
scoring = 'accuracy'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))
```

Verify your understanding:

(a) *What is the classification accuracy?*

(b) *Covert the classification accuracy into a percentage by multiplying the value by 100, giving an accuracy score of approximately 77% accurate.*

(c) *How is the standard deviation calculated? Discuss why it is an important measur.*

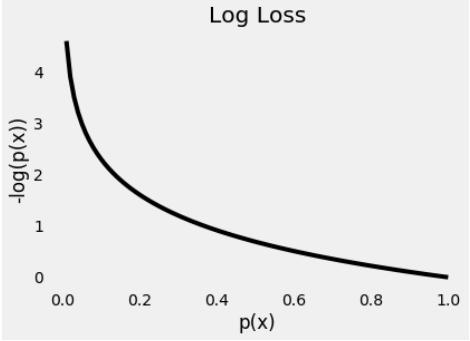
Logarithmic Loss. Logarithmic loss (or logloss) is a performance metric for evaluating the predictions of probabilities of membership to a given class. The evaluation metric used is LogLoss, described as below. The evaluation metric used is LogLoss, described as below.

$$LogLoss = -\frac{1}{n}\sum_{i=0}^{n}[y_i log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

where

- N is the number of test samples
- $\hat{y}_i$ is the predicted probability of the sample being positive ('Yes')
- y_i is 1 if the sample is a positive ('Yes'), 0 if it's a negative ('No')
- $log()$ is the natural (base e) logarithm

A smaller is better. The plot below gives us a clear picture. Logloss can be seen as $-1 *$ $the\ log\ of\ the\ likelihood$. As the predicted probability of the true class gets closer to zero, the loss increases exponentially:



Logloss for different probabilities

Predictions that are correct or incorrect are rewarded or punished proportionally to the confidence of the prediction. Below is an example of calculating logloss for Logistic regression predictions on the Pima Indians onset of diabetes dataset. You can learn more about logloss. See below page.

https://www.kaggle.com/dansbecker/what-is-log-loss

Verify your understanding:

(d) Using the information on the above link, modify the above code block to calculate logloss.

(e) *What is the logloss?* Interpret the result.

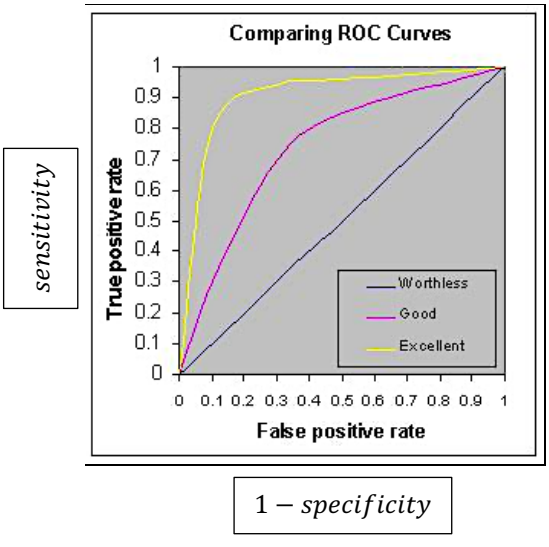
Area Under ROC Curve. Area under ROC Curve (or AUC for short) is a performance metric for binary classification problems. The AUC represents a model's ability to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predictions perfectly. An area of 0.5 represents a model that is as good as random. ROC can be broken down into sensitivity and specificity. A binary classification problem is really a trade-off between sensitivity and specificity.

- Sensitivity is the true positive rate also called the recall. It is the number of instances from the positive (first) class that actually predicted correctly. $TPR = \frac{TP}{TP+FN}$.

- Specificity is also called the true negative rate. Is the number of instances from the negative (second) class that were actually predicted correctly. $TNR = \frac{TN}{TN+FP}$.

Since an ROC curve is a plot of the true positive rate against the false positive rate for the different possible cutpoints of a diagnostic test, it demonstrates several things:

1) It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).
2) The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
3) The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.
4) The slope of the tangent line at a cutpoint gives the likelihood ratio (LR) for that value of the test.
5) The area under the curve is a measure of test accuracy.



The above graph shows three ROC curves representing excellent, good, and worthless tests plotted on the same graph. The accuracy of the test depends on how well the test separates the group being tested into those with and without the disease in question. Accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test; an area of .5 represents a worthless test. A rough guide for classifying the accuracy of a diagnostic test is the traditional academic point system:

1) .90-1 = excellent (A)
2) .80-.90 = good (B)
3) .70-.80 = fair (C)
4) .60-.70 = poor (D)
5) .50-.60 = fail (F)

Verify your understanding:

(f) Using the information on the same page, modify the above code block to calculate AUC.

(g) What is the AUC? Interpret the result.

We can plot a ROC curve for a model in Python using the $roc\_curve()$ scikit-learn function. The function takes both the true outcomes (0,1) from the test set and the predicted probabilities for the 1 class. The function returns the false positive rates for each threshold, true positive rates for each threshold and thresholds. The AUC for the ROC can be calculated using the $roc\_auc\_score()$ function. Like the $roc\_curve()$ function, the AUC function takes both the true outcomes (0,1) from the test set and the predicted probabilities for the 1 class. It returns the AUC score between 0.0 and 1.0 for no skill and perfect skill respectively.

```
# Cross Validation Classification ROC AUC
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, Y, test_size=0.1, random_state=2)
# fit a model
model = LogisticRegression(solver='liblinear')
model.fit(trainX, trainy)
# predict probabilities
probs = model.predict_proba(testX)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(testy, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(testy, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

Verify your understanding:

(h) Run the above code block and explain the plot.

Confusion Matrix. The confusion matrix is a handy presentation of the accuracy of a model with two or more classes. The table presents predictions on the x-axis and accuracy outcomes on the y-axis. The cells of the table are the number of predictions made by a machine learning algorithm. For example, a machine learning algorithm can predict 0 or 1 and each prediction may actually have been a 0 or 1. Predictions for 0 that were actually 0 appear in the cell for prediction = 0 and actual = 0, whereas predictions for 0 that were actually 1 appear in the cell for prediction = 0 and actual = 1. And so on. Below is an example of calculating a confusion matrix for a set of predictions by a Logistic Regression on the Pima Indians onset of diabetes dataset.

```
# Cross Validation Classification Confusion Matrix
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
    random_state=seed)
model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

Verify your understanding:

(i) Calculate Type I and II errors using the below information.

https://towardsdatascience.com/taking-the-confusion-out-of-confusion-matrices-c1ce054b3d3e

(j) Which one is more important in the given scenario?

Classification Report. The scikit-learn library provides a convenience report when working on classification problems to give you a quick idea of the accuracy of a model using a number of measures. The *classification_report*() function displays the precision, recall, F1-score and support for each class. The example below demonstrates the report on the binary classification problem.

```
# Cross Validation Classification Report
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
    random_state=seed)
model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print(report)
```

Verify your understanding:

(k) *You should be able to explain the report.*

2. Shortlisting Algorithms

You cannot know which algorithm will work best on your dataset beforehand. You must use trial and error to discover a shortlist of algorithms that do well on your problem that you can then double down on and tune further. I call this process spot-checking.

The question is not: What algorithm should I use on my dataset? Instead it is: What algorithms should I spot-check on my dataset? You can guess at what algorithms might do well on your dataset, and this can be a good starting point. I recommend trying a mixture of algorithms and see what is good at picking out the structure in your data. Below are some suggestions when spot-checking algorithms on your dataset:

- Try a mixture of algorithm representations (e.g. instances and trees).
- Try a mixture of learning algorithms (e.g. different algorithms for learning the same type of representation).
- Try a mixture of modelling types (e.g. linear and nonlinear functions or parametric and nonparametric).

We are going to take a look at four classification algorithms that you can spot-check on your dataset.

- Logistic Regression
- k-Nearest Neighbors.
- Classification and Regression Trees (CART).
- Support Vector Machines.

This time, we will not go into the API or parameterisation of each algorithm.

Logistic Regression. Logistic regression assumes a Gaussian distribution for the numeric input variables and can model binary classification problems. You can construct a logistic regression model using the *LogisticRegression* class. See the details below.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

```
# Logistic Regression Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_folds = 10
kfold = KFold(n_splits=10, random_state=7,shuffle=True)
model = LogisticRegression(solver='liblinear')
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

k-Nearest Neighbors (KNN). KNN uses a distance metric to find the k most similar instances in the training data for a new instance and takes the mean outcome of the neighbors as the prediction. You can construct a KNN model using the *KNeighborsClassifier* class. See the details below.

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

```
# KNN Classification
from pandas import read_csv
```

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_folds = 10
kfold = KFold(n_splits=10, random_state=7,shuffle=True)
model = KNeighborsClassifier()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Support Vector Machines (SVM). SVM seek a line that best separates two classes. Those data instances that are closest to the line that best separates the classes are called support vectors and influence where the line is placed. SVM has been extended to support multiple classes. Of particular importance is the use of different kernel functions via the kernel parameter. A powerful Radial Basis Function is used by default. You can construct an SVM model using the SVC class. See the details below.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Verify your understanding:
(l) Using the information on the above link, modify the above code block to build a SVC.

Classification and Regression Trees (CART or just decision trees). CART constructs a binary tree from the training data. Split points are chosen greedily by evaluating each attribute and each value of each attribute in the training data in order to minimize a cost function (like the Gini index). You can construct a CART model using the $DecisionTreeClassifier$ class. See the details below.

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

```
# CART Classification
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier, plot_tree
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.1
seed = 7

# split into train/test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
random_state=seed)

# fit a model
model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print(report)

from sklearn.tree import export_graphviz
export_graphviz(
            model,
            out_file='pima_tree.dot',
            feature_names=names[0:8],
            rounded=True,
```

```
            filled=True
        )

# convert .dot to .png
from subprocess import check_call
check_call(['dot','-Tpng','pima_tree.dot','-o','pima_tree.png'])

# if pydot is installed use the below
!dot -Tpng pima_tree.dot -o pima_tree.png -Gdpi-600

# display in python
import matplotlib.pyplot as plt
plt.figure(figsize = (14, 18))
plt.imshow(plt.imread('pima_tree.png'))
plt.axis('off');
plt.show();
```

Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem. For pruning decision tree, see the below.

https://statinfer.com/204-3-10-pruning-a-decision-tree-in-python/

Verify your understanding:
(m) *Run the above ML algorithms and compare the results.*
(n) *Write a code block that prunes the tree above.*

3. Comparing ML Algorithms

When you work on a machine learning project, you often end up with multiple good models to choose from. Each model will have different performance characteristics. Using resampling methods like cross-validation, you can get an estimate for how accurate each model may be on unseen data. You need to be able to use these estimates to choose one or two best models from the suite of models that you have created.

When you have a new dataset, it is a good idea to visualise the data using different techniques in order to look at the data from different perspectives. The same idea applies to model selection. You should use a number of different ways of looking at the estimated accuracy of your machine learning algorithms in order to choose the one or two algorithm to finalise. A way to do this is to use visualisation methods to show the average accuracy, variance and other properties of the distribution of model accuracies. In this section you will discover exactly how you can do that in Python with scikit-learn.

The key to a fair comparison of machine learning algorithms is ensuring that each algorithm is evaluated in the same way on the same data. You can achieve this by forcing each algorithm to be evaluated on a consistent test harness. In the example below six different classification algorithms are compared on a single dataset:

- Logistic Regression.
- Linear Discriminant Analysis.
- k-Nearest Neighbors.
- Classification and Regression Trees.
- Naive Bayes.
- Support Vector Machines.

The dataset is the Pima Indians onset of diabetes problem. The problem has two classes and eight numeric input variables of varying scales. The 10-fold cross-validation procedure is used to

evaluate each algorithm, importantly configured with the same random seed to ensure that the same splits to the training data are performed and that each algorithm is evaluated in precisely the same way. Each algorithm is given a short name, useful for summarising results afterward.

```
# Compare Algorithms
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
# load dataset
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# prepare models
models = []
models.append(('LR', LogisticRegression(solver='liblinear')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
  kfold = KFold(n_splits=10, random_state=7,shuffle=True)
  cv_results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

Verify your understanding:

(o) *Run the above code block and interpret the results.*

(p) What are the algorithms perhaps worthy of further study on this problem? Explain why.

4. More about logistic regression and the log loss functions

Logistic regression is a statistical method used for binary classification. It's called "logistic" because it uses the logistic function (or sigmoid function) to model and transform the output. The sigmoid function forces the output between 0 and 1, making it suitable to represent probabilities.

What is log loss? Log Loss is a slight twist on something called the Likelihood Function. In fact, Log Loss is '-1 the log of the likelihood function'. So, we will start by understanding the likelihood function.

The likelihood function answers the question "How likely did the model think the actually observed set of outcomes was." If that sounds confusing, an example should help.

For example, a model predicts probabilities of [0.8, 0.4, 0.1] for three houses. The first two houses were sold, and the last one was not sold. So the actual outcomes could be represented numeically as [1, 1, 0].

Let's step through these predictions one at a time to iteratively calculate the likelihood function.

The first house sold, and the model said that was 80% likely. So, the likelihood function after looking at one prediction is 0.8.

The second house sold, and the model said that was 40% likely. There is a rule of probability that the probability of multiple independent events is the product of their individual probabilities. So, we get the combined likelihood from the first two predictions by multiplying their associated probabilities. That is 0.8 * 0.4, which happens to be 0.32.

Now we get to our third prediction. That home did not sell. The model said it was 10% likely to sell. That means it was 90% likely to not sell. So, the observed outcome of not selling was 90% likely according to the model. So, we multiply the previous result of 0.32 by 0.9.

We could step through all of our predictions. Each time we'd find the probability associated with the outcome that actually occurred, and we'd multiply that by the previous result. That's the likelihood.

From Likelihood to Log Loss Each prediction is between 0 and 1. If you multiply enough numbers in this range, the result gets so small that computers can't keep track of it.

So, as a clever computational trick, we instead keep track of the log of the Likelihood. This is in a range that's easy to keep track of. We multiply this by negative 1 to maintain a common convention that lower loss scores are better.

5. Likelihood log and the exponential function

To convert log functions back to likelihood we use the exponential function (e^x). In our previous example it would be e^(-0.494).

```
negative_log_loss = -0.494likelihood
= np.exp(negative_log_loss)print("Likelihood:", likelihood)

Likelihood: 0.6101807830906798
```

The negative sign in front of the log loss value is a convention in scikit-learn for scoring functions. The convention is that higher values are considered better. In the case of log loss, a lower value indicates better performance. By convention, scikit-learn negates this value when calculating log loss. So, when you see a negative log loss, it actually means a positive log loss as per the typical mathematical definition. The negative sign is used so that higher values are better when using scikit-learn's scoring functions.

6. Confusion Matrix

A Confusion matrix is an $N \times N$ matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

The output reads as follows:

[TN FP]
[FN TP]
[TN False Positive]
[False Negative TP]

141 cases predicted negative (0) and actually been negative. 21 cases predicted positive (1) but actually been negative.

41 cases predicted negative (0) and actually been positive (1). 51 cases predicted positive (1) and actually been positve (1).

Type I error is the ratio of false positive to the sum of false positives and false positives, or

FP/(FP+TN)= 21/(21+141)= 0.13 or 13%

Type II error is the ratio of false negatives to the sum of the positives and false negatives, or

FN/ (FN+TP)= 41/(41+51)= 0.44 or 44%

In the context of predicting suffering or not suffering diabetes, false positives would be people that they were told that they have diabetes, but in reality, they do not.

False negatives instead are patients that they were told that they do not have diabetes but instead they do suffer from diabetes.

Although this may be up to personal interpretation, medically speaking, it would be more dangerous telling a patient that they do not have a disease (and not treating it) when they actually suffer from it.

7. Precision, recall, F1 score and support

Precision is measures the accuracy of positive (or negative) predictions. Precision answers the question: Of all the instances predicted as positive, how many were actually positive?

In this case 77% for negative (not belonging or 0) predictions and 71% for positive predictions (belonging or 1).

Recall measures the ability of the classifier to find all the positive (or negative) samples. It answers the questions: Of all the actual positive instances, how many were correctly predicted as positive?" In this case 87% for negative cases and 55% for positive instances.

The F1 Score answers the question How well does the classifier perfomr in terms ob both precision and recall? A high F1 score indicates that the classifier has both high precision and high recall and

conversely, a low F1 score indicates low precision and low score. In our example is 82% for negative cases and 62% for positive cases.

Support is just the number of occurences on each class. In this case 162 negative occurrences and 92 positive occurrences.

Overall, the model gives an accuracy of 76%. Finally, the model calculate the average and the weighted average for each individual metric. For example (0.77+0.71)/2 = 0.74 for precision.

8. References

[1]. Géron A. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media; 2019.
[2]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.
[3]. Grus, J., 2019. Data science from scratch: first principles with python. O'Reilly Media.
[4]. Müller, A.C. and Guido, S., 2016. Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".
[5]. Brownlee, J., 2014. Machine learning mastery.
[6]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.
[7]. SAS, 2018, Advanced Predictive Modelling using SAS
[8]. Géron, A., 2019. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.