Data Mining and Decision Systems
Workshop 5
Models & Confusion Matrices

Aims of the workshop

Last week's lectures covered confusion matrices and introduced the concept of models, alongside the scikit learn library. The aim of this workshop is to provide a practical setting for creating a model you already know: Linear Regression. To become familiar with the tools and available methods, and how to take output values and compute confusion matrices by hand, and programmatically.

This workshop utilises previous workshops on Seaborn, and will require an understanding of plotting to complete.

Please see 'Useful Information' below on how to lookup certain Python functionality. The concept behind this workshop is about discovery, and experimentation surrounding topics covered so far.

Feel free to discuss the work with peers, or with any member of the teaching staff.

Useful Information

Seaborn

Seaborn is a high-level plotting library for Python. Written on top of Matplotlib it enables rich and intricate plots whilst maintaining readability and ease of use. You can access the library's web page at the following link:
https://seaborn.pydata.org/

For API Reference, consider: https://seaborn.pydata.org/api.html

You may find https://seaborn.pydata.org/tutorial.html useful for understanding more functionality, and introducing you to each aspect.

Matplotlib

Matplotlib is a powerful 2D plotting library for Python. It is the de-facto standard for the community. Whilst matplotlib is considered powerful, this comes at the expense of verbosity. Those wishing to further manipulate their diagrams and plots may wish to seek out matplotlib documentation (https://matplotlib.org/3.1.1/api/index.html) to enable more advanced graphics manipulation.

Scikit Learn (Sklearn)

Scikit-learn is a machine learning library for Python, providing simple tools to work with complex models and to facilitate data analysis. Sklearn is built on top of common frameworks such as numpy (powerful arrays), scipy (scientific python library), and matplotlib(2D plotting).
Most of the techniques covered within lectures will be available in some form through Sklearn, such as regression/classification models, and cross validation.

Functions of note: read_csv, [ ], [[ ]], confusion_matrix, ravel, plot_confusion_matrix, scatterplot, train_test_split, LinearRegression, numpy.asarray, reshape, shape, LinearRegression.fit, LinearRegression.predict, mean_absolute_error, mean_squared_error, LinearRegression.coef_, LinearRegression.intercept_, lineplot.

Reminder

We encourage you to discuss the contents of the workshop with the delivery team, and any findings you gather from the session.

Workshops are not isolated, if you have questions from previous weeks, or lecture content, please come and talk to us.

The contents of this workshop are not intended to be 100% complete within the 2 hours; as such it’s expected that some of this work be completed outside of the session. Exercises herein represent an example of what to do; feel free to expand upon this.

Remember the magic: `%matplotlib inline`
Remember common imports:

Dataframe as df
Seaborn as sns
Matplotlib.pyplot as plt
Numpy as np

Confusion Matrices

Exercise 0: Install the *scikit-learn* library using pip from a command line (or jupyter notebook cell). You may need to refer back to previous workshops.

Exercise 1: Given the following insurance claim model output, representing the ground truth data and the predicted output data:

```
gt = [ "C", "NC", "C", "C", "C", "NC", "NC", "NC", "NC", "C", "NC",
"NC", "C", "C", "NC", "NC", "NC" ]

pred = [ "NC", "NC", "NC", "NC", "C", "NC", "NC", "C", "NC", "NC", "C",
"C", "NC", "NC", "NC", "C", "NC" ]
```

Where C = Claim
NC = No Claim

By-hand:

1. Calculate TP, TN, FP, and FN metrics.
2. Calculate overall accuracy of the provided model
3. Calculate Sensitivity and Specificity metrics.

Verify your answers with a member of the delivery team.

Exercise 2: Produce the following confusion matrices from the results in Ex1, by hand:

1. Un-normalised.
2. Normalised.

Exercise 3: Install scikit-plot. Repeat Ex 2, plotting the confusion matrices using the following code, replacing ground_truth and predicted_y with arrays from Ex1. For Normalised vs Un-normalised, you can pass 'Normalize' keyword argument to the scikit-plot function.

```
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
      ground_truth,
      Predicted_y,
)
```

Note: If you just want the confusion matrix itself, you can obtain the raw confusion matrix using *sklearn.metrics.confusion_matrix*. Confusion Matrix will return a 2-Dimensional array, we can use *ravel* to flatten it. Remember to check the documentation for these functions to find out which order the parameters are called. **Function used below has no concept of which class is positive or negative => Results may be inverted. May wish to use LabelBinarizer to convert to 0, 1.**

```
cm = confusion_matrix(t, p)
tn, fp, fn, tp = confusion_matrix(t,p).ravel()
```

Regression Analysis

Exercise 3: Install the *scikit-learn* library using pip from a command line (or jupyter notebook cell). You may need to refer back to previous workshops.

Exercise 4: Load in *simple_data.csv* (available under Canvas>Modules>Week 6), perform the following:

1. Visualise the simple_data using a scatter plot of X against Y.
2. Using sklearn.model_selection.train_test_split perform a 70-30 linear split on the data, setting shuffle=False.
3. Visualise the training data, visualise the test data. Compare the trends. What do you notice about the ranges on the x-axis?
4. Generate a Linear Regression Model using the *fit* function.
    a. E.g

```
# Import Numpy as we need to use this to reshape our input data.
import numpy as np

# Create a linear regression model, we will need to train this.
# This wraps our Linear Regression model with nice functions such as
fit, predict, etc. So we can use them.
model = LinearRegression()

# Grab out X input data, and Y target data from a training split.
train_x = train['x']
train_y = train['y']
print("Shape before reshaping:")
print("Train X", train_x.shape)
print("Train Y", train_y.shape)
print("\n")

# Fit expects a 2D array, reshape so we have N rows, 1 Column. Rather
than 1D array of N entries.
train_x = np.asarray(train_x).reshape(-1,1)
train_y = np.asarray(train_y).reshape(-1,1)
```

```
print("Shape after reshaping:")
print("Train X: ", train_x.shape)
print("Train Y: ", train_y.shape)
print("\n")

# Call fit to train the Linear Regression model on our training data,
providing training targets.
reg_model = model.fit(
      X=train_x,
      y=train_y
)
```

5. Using the test data, use predict on your trained model.
    a. Hint: If you obtain “ValueError: Expected 2D array, got 1D array instead” error, you need to use numpy to reshape your data (this time the TEST split). See Training above.
    b. The output of predict will be a 2D array. You may want to use ravel here to flatten it.
6. Calculate MAE and MSE error metrics using sklearn.metrics.mean_absolute_error and sklearn.metrics.mean_squared error on:

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
```

    a. Training set. E.g mean_absolute_error(train_y, model.predict(train_x))
    b. Test set
7. Visualise your Test set X data against your Test Set Predicted Y data. Use a lineplot for this. You should see a straight line (May need to use *ravel* on pred_y).
    a. Note: You can use *reg_model.coef_* and *reg_model.intercept_* to find the Y=mx+c parameters.
    b. Does this seem to fit the test data well? What do the error metrics say?

```
sns.lineplot(x=test['x'], y=pred_y.ravel(), label='Model')
sns.scatterplot(x=test['x'], y=test['y'], label='Test Data')
```

Note: We use test[‘x’] and train[‘x’] (our DataFrames) anytime we are plotting with seaborn. If we are using sklearn functions, we use our reshaped numpy arrays train_x, test_x. When plotting the predicted y values with Seaborn we need to convert the 2D output from Sklearn, into a 1-D array it can understand, using ravel.

Exercise 5: Repeat Ex 4 using *not_simple_data.csv*, for the following input data splitting mechanisms:

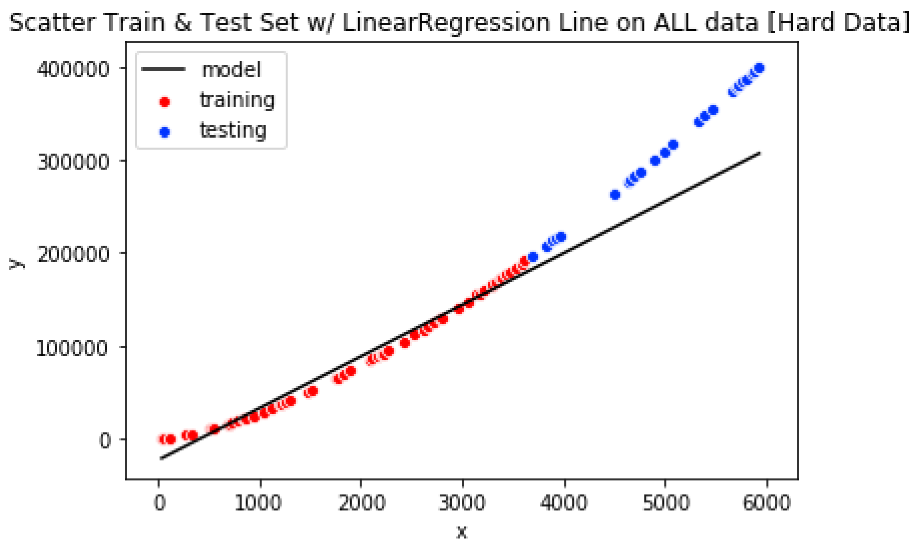
1. 70-30 Linear Split (Shuffle = False)
    a. How does the data differ from Ex 4 data?
    b. Visualising the trend of the training, and the test set, what differences do you notice between sets?
2. 70-30 Random Split (Equivalent to Linear Split with Shuffle = True)

Exercise 6: On the harder dataset, use the following code to plot the training data points, the testing data points, and your trained linear regression model's predicted output on ALL data (train+test).

Note: We want to predict on ALL data, so we can see what our model would predict for all points. It allows us to compare how close the model is to our training data, visually, as well as how close it is to our test data.
As noted in the lecture, we **should expect** training data to perform the best.

```
import matplotlib.pyplot as plt

# Reshape our DataFrame so Sklearn understands it.
whole_x = np.asarray( df['x'] ).reshape(-1,1)
whole_y = np.asarray( df['y'] ).reshape(-1,1)

fig, ax = plt.subplots()
plt.title("Scatter Train & Test Set w/ LinearRegression Line on ALL data
[Hard Data]")

# Line plot of our model applied to ALL our data train + test
sns.lineplot( x=df['x'], y=reg_model.predict( whole_x ).ravel(), ax=ax,
label='model')

# Line plot of just training data. Red Data points
sns.scatterplot(x=train['x'], y=train['y'], ax=ax, c=[[1, 0, 0]],
label='training')

# Line plot of just testing data. Blue-ish data points.
sns.scatterplot(x=test['x'], y=test['y'], ax=ax, c=[[0, 0.2, 1]],
label='testing')

# Colour the line plot in black.
plt.gca().get_lines()[0].set_color("black")

# Show our legend.
ax.legend()
```

You should produce the following plot:



Answering the following questions, discuss these with a member of the delivery team:
Comparing the visualised lines of the generated models,

1. When comparing the toy data to the harder toy data, which Linear Regression model appears more appropriate? Why? Justify your answer, referencing plots, and/or error metrics.
2. If the input range was to exceed the data you have available, to go beyond X = 6000, what implications would this have for utilising the model you have generated?