Module 6: Classification

The following tutorial contains Python examples for solving classification problems. You should refer to the Chapters 3 and 4 of the "Introduction to Data Mining" book to understand some of the concepts introduced in this tutorial.

Classification is the task of predicting a nominal-valued attribute (known as class label) based on the values of other attributes (known as predictor variables). The goals for this tutorial are as follows:

1. To provide examples of using different classification techniques from the scikit-learn library package.
2. To demonstrate the problem of model overfitting.

Read the step-by-step instructions below carefully. To execute the code, click on the corresponding cell and press the SHIFT-ENTER keys simultaneously.

Vertebrate Dataset

We use a variation of the Admission Predict data given in the Assignment 2. Each columns are defined into Gre Score, TOEFL Score, Universal Rating, SOP ,LOR ,CGPA ,Research and Chance of Admit based on a set of explanatory attributes (predictor variables). To illustrate this, we will first load the data into a Pandas DataFrame object and display its content.

```
import pandas as pd
import numpy as np
url = 'https://raw.githubusercontent.com/josephdonati/CSC-177---Project-2/master/Data/Admissi
data_admit = pd.read_csv(url)
data_admit
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72

Given the limited number of scores,we try to classify the chance of admit into three groups with classification task (High Possibility , Medium Possibility and Low Possibility). We can do so by replacing the class labels of the instances to *High Possibility*,*Medium Possibility* and *Low Possibility*.

```
data_admit['Chance of Admit '] = data_admit['Chance of Admit '].replace([0.78,0.79,0.8,0.81,0
data_admit['Chance of Admit '] = data_admit['Chance of Admit '].replace([0.58,0.59,0.6,0.61,0
data_admit['Chance of Admit '] = data_admit['Chance of Admit '].replace([0.34,0.36,0.37,0.38,

data_admit
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	High Possibility
1	2	324	107	4	4.0	4.5	8.87	1	Medium Possibility
2	3	316	104	3	3.0	3.5	8.00	1	Medium Possibility
3	4	322	110	3	3.5	2.5	8.67	1	High Possibility
4	5	314	103	2	2.0	3.0	8.21	0	Medium Possibility
...	...	...	...	...	...	...	...	...	...
495	496	332	108	5	4.5	4.0	9.02	1	High Possibility
496	497	337	117	5	5.0	5.0	9.87	1	High Possibility

We can apply Pandas cross-tabulation to examine the relationship between the GRE Score and TOEFL Score attributes with respect to the Chance of Admit.

```
pd.crosstab([data_admit['GRE Score'],data_admit['TOEFL Score']],data_admit['Chance of Admit '
```

GRE Score	TOEFL Score	Chance of Admit: High Possibility	Low Possibility	Medium Possibility
290	**100**	0	1	0
	104	0	1	0
293	**97**	0	0	1
294	**93**	0	1	0
	95	0	1	0
...	**...**	...	...	...
340	**112**	1	0	0
	113	1	0	0
	114	1	0	0
	115	2	0	0
	120	4	0	0

The results above show that it is possible to distinguish High Possibility from Medium Possibility and Low Possibility using these three attributes alone since each combination of their attribute values would yield only instances that belong to the same class. For example, High Possibility can be identified as High GRE Score, with good TOEFL Score and excellent CGPA. Such a relationship can also be derived using a decision tree classifier, as shown by the example given in the next subsection.

Decision Tree Classifier

In this section, we apply a decision tree classifier to the vertebrate dataset described in the previous subsection.

```
from sklearn import tree
import numpy as np
data_admit = data_admit.fillna(data_admit.median(axis=0))
Y = pd.DataFrame(data_admit, columns=['Chance of Admit '])
X = data_admit.drop(['Serial No.','Chance of Admit '],axis=1)


clf = tree.DecisionTreeClassifier(criterion='entropy',max_depth=3)
clf = clf.fit(X, Y)
clf
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
X
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337	118	4	4.5	4.5	9.65	1
1	324	107	4	4.0	4.5	8.87	1
2	316	104	3	3.0	3.5	8.00	1
3	322	110	3	3.5	2.5	8.67	1
4	314	103	2	2.0	3.0	8.21	0
...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1
496	337	117	5	5.0	5.0	9.87	1
497	330	120	5	4.5	5.0	9.56	1
498	312	103	4	4.0	5.0	8.43	0
499	327	113	4	4.5	4.5	9.04	0

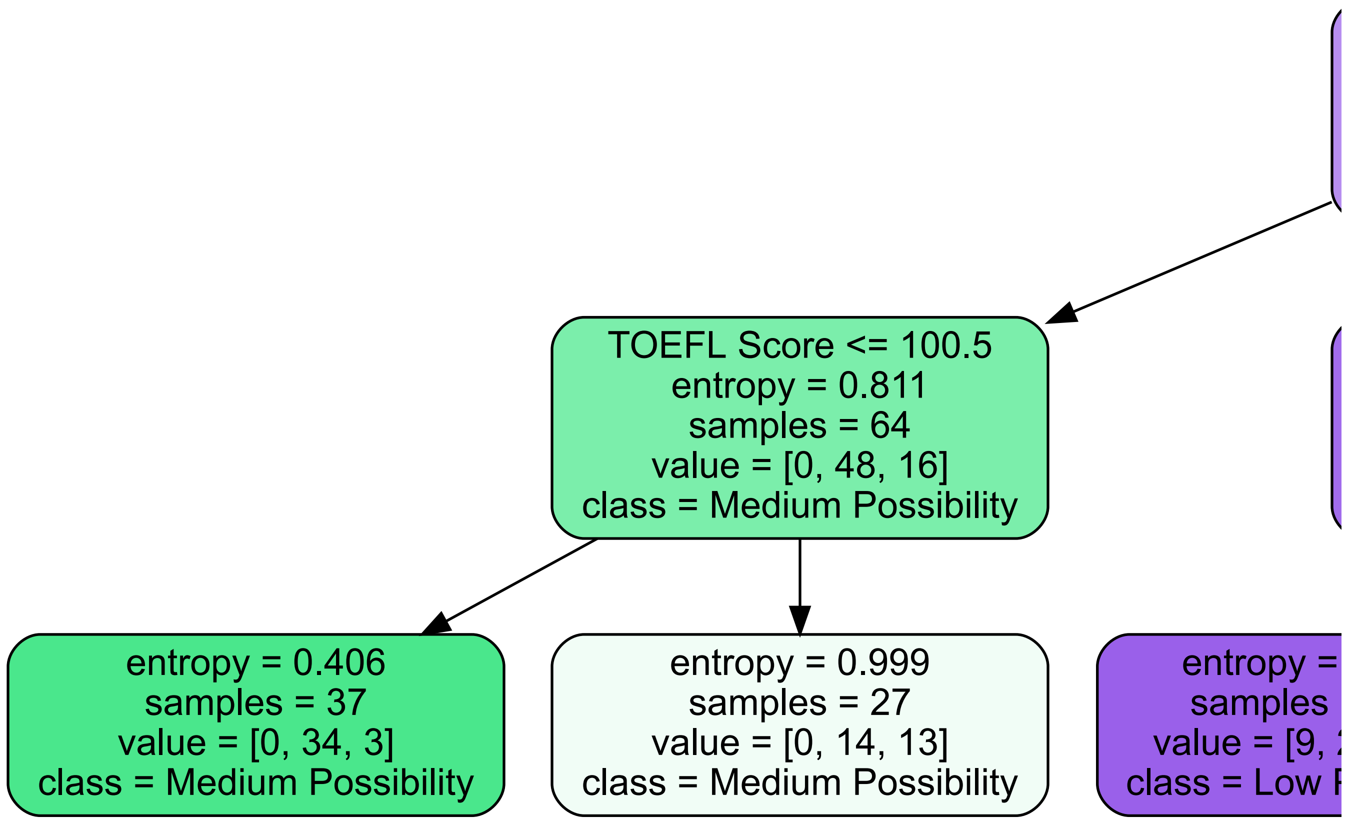
500 rows × 7 columns

The preceding commands will extract the predictor (X) and target class (Y) attributes from the vertebrate dataset and create a decision tree classifier object using entropy as its impurity measure for splitting criterion. The decision tree class in Python sklearn library also supports using 'gini' as impurity measure. The classifier above is also constrained to generate trees with a maximum depth equals to 3. Next, the classifier is trained on the labeled data using the fit() function.

We can plot the resulting decision tree obtained after training the classifier. To do this, you must first install both graphviz (http://www.graphviz.org) and its Python interface called pydotplus (http://pydotplus.readthedocs.io/).

```
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
```

```
dot_data = tree.export_graphviz(clf, out_file=None, feature_names=X.columns, class_names=['Hi
graph = graphviz.Source(dot_data)
graph
```



Next, suppose we apply the decision tree to classify the following test examples.

```
testData = [[2,324, 107, 4, 4.0, 4.5, 8.87, 1, 'Medium Possibility'],
           [3,316, 104, 3, 3.0, 3.5, 8.00, 1, 'Medium Possibility'],
           [4,322, 110, 3, 3.5, 2.5, 8.67, 1, 'High Possibility'],
           [5,314, 103, 2, 2.0, 3.0, 8.21, 0, 'Medium Possibility']]
testData = pd.DataFrame(testData, columns=data_admit.columns)
testData
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	2	324	107	4	4.0	4.5	8.87	1	Medium Possibility
									High

We first extract the predictor and target class attributes from the test data and then apply the decision tree classifier to predict their classes.

```
testY = pd.DataFrame(testData, columns=['Chance of Admit '])
testX = testData.drop(['Serial No.','Chance of Admit '],axis=1)
predY = clf.predict(testX)
predictions = pd.concat([testData['Chance of Admit '],pd.Series(predY,name='Predicted Chance
predictions
```

	Chance of Admit	Predicted Chance of Admit
0	Medium Possibility	Medium Possibility
1	Medium Possibility	Medium Possibility
2	High Possibility	Medium Possibility
3	Medium Possibility	Medium Possibility

Except for Serial No 2, Rest all Chance of Admit where with GRE Score 324 and TOEFL Score of 107 with a good CGPA of 8.87 have Medium Chance and classifier correctly predicts the class label of the test examples. We can calculate the accuracy of the classifier on the test data as shown by the example given below.

```
from sklearn.metrics import accuracy_score

print('Accuracy on test data is %.2f' % (accuracy_score(testY, predY)))
```

```
Accuracy on test data is 0.75
```

Artificial Neural Network

```
from keras import Sequential
from keras.layers import Dense
import numpy as np
from sklearn import preprocessing
```

```
Using TensorFlow backend.
```

```
# Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1] for red,green,blue)
```

```
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = "{}-{}".format(name, x)
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)

# Encode text values to indexes(i.e. [1],[2],[3] for red,green,blue).
def encode_text_index(df, name):
    le = preprocessing.LabelEncoder()
    df[name] = le.fit_transform(df[name])
    return le.classes_

# Convert a Pandas dataframe to the x,y inputs that TensorFlow needs
import collections
def to_xy(df, target):
    result = []
    for x in df.columns:
        if x != target:
            result.append(x)
    # find out the type of the target column.
    target_type = df[target].dtypes
    target_type = target_type[0] if isinstance(target_type, collections.Sequence) else target_
    # Encode to int for classification, float otherwise. TensorFlow likes 32 bits.
    if target_type in (np.int64, np.int32):
        # Classification
        dummies = pd.get_dummies(df[target])
        return df[result].values.astype(np.float32), dummies.values.astype(np.float32)
    else:
        # Regression
        return df[result].values.astype(np.float32), df[target].values.astype(np.float32)
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Admission_Predict_.csv.csv to Admission_Predict_.csv.csv

```
import io
```

```
data_df = pd.read_csv(io.StringIO(uploaded['Admission_Predict_.csv.csv'].decode('utf-8')),na_
```

```
data_df = data_df.drop('Serial No.', axis=1)
```

```
data_df['Chance of Admit '] = data_df['Chance of Admit '].replace([0.78,0.79,0.8,0.81,0.82,0.
data_df['Chance of Admit '] = data_df['Chance of Admit '].replace([0.58,0.59,0.6,0.61,0.62,0.
data_df['Chance of Admit '] = data_df['Chance of Admit '].replace([0.34,0.36,0.37,0.38, 0.39,
```

```
Classes = encode_text_index(data_df,'Chance of Admit ')
```

```
data_df
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0
1	324	107	4	4.0	4.5	8.87	1	2
2	316	104	3	3.0	3.5	8.00	1	2
3	322	110	3	3.5	2.5	8.67	1	0
4	314	103	2	2.0	3.0	8.21	0	2
...	...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1	0
496	337	117	5	5.0	5.0	9.87	1	0
497	330	120	5	4.5	5.0	9.56	1	0
498	312	103	4	4.0	5.0	8.43	0	2
499	327	113	4	4.5	4.5	9.04	0	0

500 rows × 8 columns

```
testData = [[2,324, 107, 4, 4.0, 4.5, 8.87, 1, 'Medium Possibility'],
           [3,316, 104, 3, 3.0, 3.5, 8.00, 1, 'Medium Possibility'],
           [4,322, 110, 3, 3.5, 2.5, 8.67, 1, 'High Possibility'],
           [5,314, 103, 2, 2.0, 3.0, 8.21, 0, 'Medium Possibility']]
testData = pd.DataFrame(testData, columns=data_admit.columns)
testData
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	2	324	107	4	4.0	4.5	8.87	1	Medium Possibility
1	3	316	104	3	3.0	3.5	8.00	1	Medium Possibility
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	High

```
testData = testData.drop('Serial No.', axis=1)
encode_text_index(testData,'Chance of Admit ')
```

```
array(['High Possibility', 'Medium Possibility'], dtype=object)
```

```
testData
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	324	107	4	4.0	4.5	8.87	1	1
1	316	104	3	3.0	3.5	8.00	1	1
2	322	110	3	3.5	2.5	8.67	1	0
3	314	103	2	2.0	3.0	8.21	0	1

```
Classes
```

```
array(['High Possibility', 'Low Possibility', 'Medium Possibility'],
      dtype=object)
```

```
X,Y = to_xy(data_df,'Chance of Admit ')
testX, testY = to_xy(testData,'Chance of Admit ')
```

```
print(X.shape)
print(Y.shape)
Y
```

```
(500, 7)
(500, 3)
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       ...,
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.]], dtype=float32)
```

```
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras import Sequential
```

```
model = Sequential()
model.add(Dense(12, input_dim = X.shape[1], activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam')
model.fit(X,Y,verbose=2, epochs=100)
```

```
Epoch 1/100
16/16 - 0s - loss: 35.4825
Epoch 2/100
16/16 - 0s - loss: 13.3414
Epoch 3/100
16/16 - 0s - loss: 7.8128
Epoch 4/100
16/16 - 0s - loss: 3.7137
Epoch 5/100
16/16 - 0s - loss: 1.5589
Epoch 6/100
16/16 - 0s - loss: 1.3660
Epoch 7/100
16/16 - 0s - loss: 1.2828
Epoch 8/100
16/16 - 0s - loss: 1.2636
Epoch 9/100
16/16 - 0s - loss: 1.2233
Epoch 10/100
16/16 - 0s - loss: 1.2263
Epoch 11/100
16/16 - 0s - loss: 1.2104
Epoch 12/100
16/16 - 0s - loss: 1.1523
Epoch 13/100
16/16 - 0s - loss: 1.1328
Epoch 14/100
16/16 - 0s - loss: 1.1795
Epoch 15/100
16/16 - 0s - loss: 1.1023
Epoch 16/100
16/16 - 0s - loss: 1.0912
Epoch 17/100
16/16 - 0s - loss: 1.0682
Epoch 18/100
16/16 - 0s - loss: 1.0882
Epoch 19/100
16/16 - 0s - loss: 1.0221
Epoch 20/100
16/16 - 0s - loss: 0.9978
Epoch 21/100
16/16 - 0s - loss: 0.9908
Epoch 22/100
16/16 - 0s - loss: 0.9741
Epoch 23/100
16/16 - 0s - loss: 0.9810
Epoch 24/100
16/16 - 0s - loss: 0.9553
Epoch 25/100
16/16 - 0s - loss: 0.9463
Epoch 26/100
16/16 - 0s - loss: 0.9312
Epoch 27/100
16/16 - 0s - loss: 0.9262
Epoch 28/100
16/16 - 0s - loss: 0.9198
Epoch 29/100
```

```
16/16 - 0s - loss: 0.9027
Epoch 30/100
16/16 - 0s - loss: 0.8846
Epoch 31/100
16/16 - 0s - loss: 0.8965
Epoch 32/100
16/16 - 0s - loss: 0.8802
Epoch 33/100
16/16 - 0s - loss: 0.8714
Epoch 34/100
16/16 - 0s - loss: 0.8763
Epoch 35/100
16/16 - 0s - loss: 0.8797
Epoch 36/100
16/16 - 0s - loss: 0.8770
Epoch 37/100
16/16 - 0s - loss: 0.8532
Epoch 38/100
16/16 - 0s - loss: 0.8436
Epoch 39/100
16/16 - 0s - loss: 0.8267
Epoch 40/100
16/16 - 0s - loss: 0.8337
Epoch 41/100
16/16 - 0s - loss: 0.8258
Epoch 42/100
16/16 - 0s - loss: 0.8403
Epoch 43/100
16/16 - 0s - loss: 0.8333
Epoch 44/100
16/16 - 0s - loss: 0.8628
Epoch 45/100
16/16 - 0s - loss: 0.8325
Epoch 46/100
16/16 - 0s - loss: 0.8108
Epoch 47/100
16/16 - 0s - loss: 0.8718
Epoch 48/100
16/16 - 0s - loss: 0.8403
Epoch 49/100
16/16 - 0s - loss: 0.8329
Epoch 50/100
16/16 - 0s - loss: 0.8075
Epoch 51/100
16/16 - 0s - loss: 0.7987
Epoch 52/100
16/16 - 0s - loss: 0.8333
Epoch 53/100
16/16 - 0s - loss: 0.8196
Epoch 54/100
16/16 - 0s - loss: 0.8170
Epoch 55/100
16/16 - 0s - loss: 0.8171
Epoch 56/100
16/16 - 0s - loss: 0.8093
Epoch 57/100
16/16 - 0s - loss: 0.7935
Epoch 58/100
```

```
16/16 - 0s - loss: 0.7950
Epoch 59/100
16/16 - 0s - loss: 0.7985
Epoch 60/100
16/16 - 0s - loss: 0.7955
Epoch 61/100
16/16 - 0s - loss: 0.7887
Epoch 62/100
16/16 - 0s - loss: 0.7864
Epoch 63/100
16/16 - 0s - loss: 0.7744
Epoch 64/100
16/16 - 0s - loss: 0.7829
Epoch 65/100
16/16 - 0s - loss: 0.7696
Epoch 66/100
16/16 - 0s - loss: 0.7759
Epoch 67/100
16/16 - 0s - loss: 0.7816
Epoch 68/100
16/16 - 0s - loss: 0.7979
Epoch 69/100
16/16 - 0s - loss: 0.7704
Epoch 70/100
16/16 - 0s - loss: 0.7707
Epoch 71/100
16/16 - 0s - loss: 0.7631
Epoch 72/100
16/16 - 0s - loss: 0.7686
Epoch 73/100
16/16 - 0s - loss: 0.7795
Epoch 74/100
16/16 - 0s - loss: 0.7737
Epoch 75/100
16/16 - 0s - loss: 0.7502
Epoch 76/100
16/16 - 0s - loss: 0.7680
Epoch 77/100
16/16 - 0s - loss: 0.7647
Epoch 78/100
16/16 - 0s - loss: 0.7754
Epoch 79/100
16/16 - 0s - loss: 0.7612
Epoch 80/100
16/16 - 0s - loss: 0.7664
Epoch 81/100
16/16 - 0s - loss: 0.7663
Epoch 82/100
16/16 - 0s - loss: 0.7957
Epoch 83/100
16/16 - 0s - loss: 0.7700
Epoch 84/100
16/16 - 0s - loss: 0.7758
Epoch 85/100
16/16 - 0s - loss: 0.7867
Epoch 86/100
16/16 - 0s - loss: 0.7498
Epoch 87/100
```

```
Epoch 87/100
16/16 - 0s - loss: 0.7915
Epoch 88/100
16/16 - 0s - loss: 0.8147
Epoch 89/100
16/16 - 0s - loss: 0.7881
Epoch 90/100
16/16 - 0s - loss: 0.7689
Epoch 91/100
16/16 - 0s - loss: 0.7414
Epoch 92/100
16/16 - 0s - loss: 0.7582
Epoch 93/100
16/16 - 0s - loss: 0.7509
Epoch 94/100
16/16 - 0s - loss: 0.7490
Epoch 95/100
16/16 - 0s - loss: 0.7511
Epoch 96/100
16/16 - 0s - loss: 0.7940
Epoch 97/100
16/16 - 0s - loss: 0.7579
```

```
pred = model.predict(testX)
```

```
Epoch 99/100
```

```
pred = np.argmax(pred, axis=1)
```

```
16/16 - 0s - loss: 0.7415
```

```
true = np.argmax(testY, axis=1)
```

```
Classes[pred]
```

```
array(['High Possibility', 'Medium Possibility', 'Medium Possibility',
       'Low Possibility'], dtype=object)
```

```
Classes[true]
```

```
array(['Low Possibility', 'Low Possibility', 'High Possibility',
       'Low Possibility'], dtype=object)
```

```
print('Accuracy on the test data is %.2f' % (accuracy_score(true, pred)))
```

```
Accuracy on the test data is 0.25
```