FinTech HW2

tags: `FinTech`

年級：資工碩二
學號：R08922125
姓名：張皓鈞

INSTRUCTIONS

> Dataset from kaggle Credit Card Fraud Detection. The features V1, V2, … V28 are the principal
> components obtained with PCA, the only features which have not been transformed with PCA are
> 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and
> the first transaction in the dataset.
> Class value 1 in case of fraud and 0 otherwise.

Data Preprocessing

- train data=80%, test data=20%
- amount 屬性的範圍太大，需要做feature scale的調整所以將此做normalize

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import keras

## Load data and preprocessing
data = pd.read_csv('Data.csv')
scaler = StandardScaler()
data['NormalizedAmount'] = scaler.fit_transform(data['Amount'].values.reshape(-1
data = data.drop(['Amount', 'Time'], axis = 1)
y = data['Class']
x = data.drop(['Class'], axis = 1)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, rando

```

Problem 1. Classification

(i) Construct DNN model and Gridsearch

- 用keras sequential建立四層的DNN model, 總共有$V1...V28, noramlizedAmount$ 29個feature去做training。
- 前三層的activation為ReLu, 最後一層output為sigmoid
- 中間dropout 50%，以防止overfitting
- learn_rate = [0.0001, 0.0003 ,0.0005, 0.001]
  batch_size = [10, 60]
  epochs = [20, 70]
  neurons = [5, 10]
  做gridsearch找出32種參數組合中，accuracy最高的一組參數。

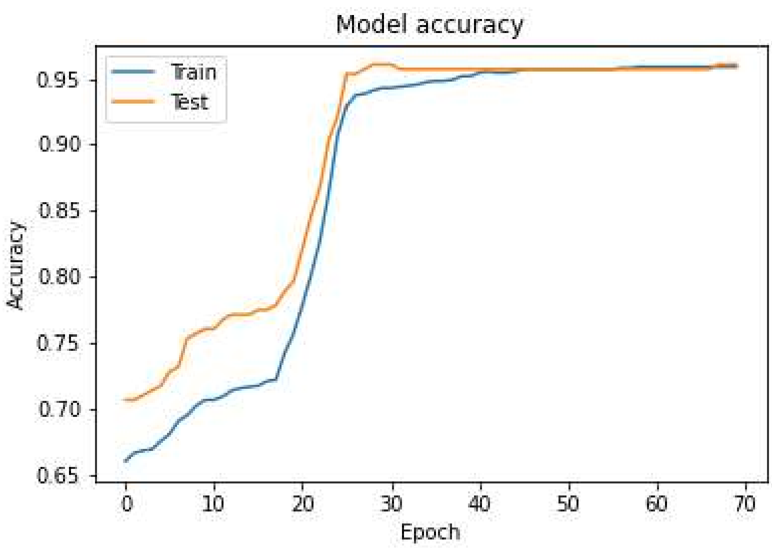
```python
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier

def create_model(learn_rate=0.001, neurons=10):
# create model
  model = Sequential()
  model.add(Dense(neurons,input_dim = 29, activation = 'relu'))
  # model.add(Dense(24, activation = 'relu’))
  model.add(Dense(neurons, activation = 'relu'))
  Dropout(0.5),
  model.add(Dense(neurons, activation = 'relu'))
  model.add(Dense(1, activation = 'sigmoid'))

  opt = keras.optimizers.Adam(learning_rate=learn_rate)
  # Compile model
  model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
  return model

# Gridsearch Params
learn_rate = [0.0001, 0.0003 ,0.0005, 0.001]
batch_size = [10, 60]
epochs = [20, 70]
neurons = [5, 10]
param_grid = dict(batch_size=batch_size, epochs=epochs,learn_rate=learn_rate,neu

model = KerasClassifier(build_fn=create_model)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=None)
grid_result = grid.fit(x_train, y_train)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

需要跑一點時間，已經特地降低grid數目、epochs跟neurons，準確度影響並不大。

- 得出最佳參數組合
  Best: 0.952679
  using {‘batch_size’: 10, ‘epochs’: 70, ‘learn_rate’: 0.0001, ‘neurons’: 10}

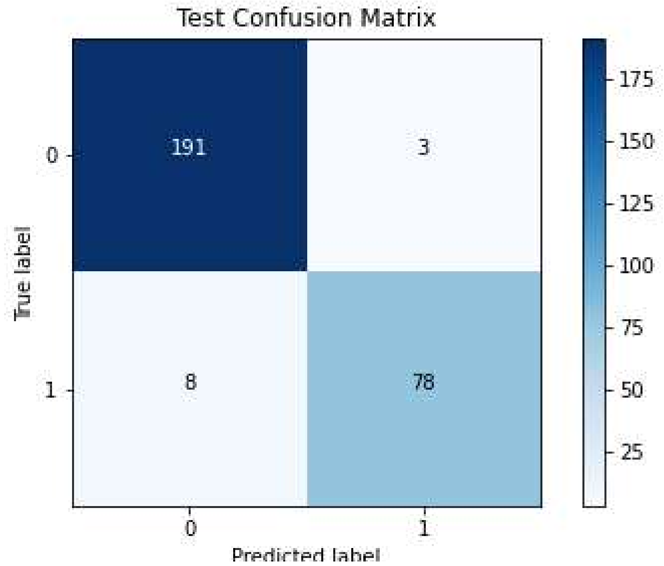
Choose best model from gridsearch and plot

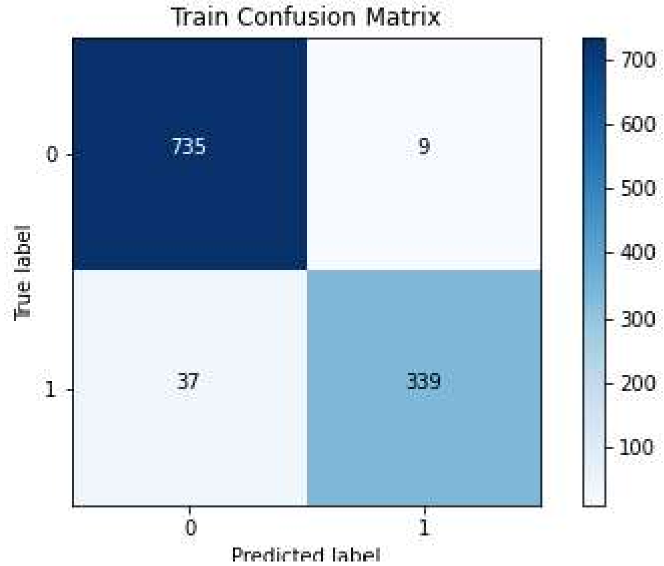
```
grid_best = grid_result.best_estimator_
history = grid_best.fit(x_train, y_train, validation_data=(x_test, y_test))

## DNN Draw accuracy and loss
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



(ii) Plot Confusion matrices

- 用sklearn的
  from sklearn.metrics import confusion_matrix
  from sklearn.metrics import plot_confusion_matrix






```
def confusion(model,x,y):
  y_predict = model.predict(x)
  return confusion_matrix(y,y_predict)

cm_train_DNN = confusion(grid_best,x_train, y_train)
cm_test_DNN = confusion(grid_best,x_test, y_test)

cm_plot_labels = ['0','1']

plot_confusion_matrix(cm_train_DNN, cm_plot_labels, 'Train Confusion Matrix')
plot_confusion_matrix(cm_test_DNN, cm_plot_labels, 'Test Confusion Matrix')
```

(iii) Precision, Recall, F1-Score

- 雖然keras裡有現成的函式，但我想說上題都有Confusion Matrix了就帶入公式自己算。
- 題目是說對each class都要算上述的指標，所以在confusion matrix中，class 1與 class 0的true positive等等數值會完全相反。

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

```
def calculation(cm, class_ = 1):
  TP = cm[1][1]
  FP = cm[0][1]
  FN = cm[1][0]
  TN = cm[0][0]
  if (class_ == 0):
    TP = TN
    tmp = FN
    FN = FP
    FP = tmp

  precision = TP / (TP+FP)
  recall = TP / (TP+FN)
  F1_score = 2 * (precision*recall) / (precision+recall)
  print("For Class "+ str(class_))
  print("precision: ",precision)
  print("recall: ", recall)
  print("F1-score: ",F1_score)
###############################################
output:
----------DNN model training-----------

For Class 1
precision:  0.9741379310344828
recall:  0.901595744680851
F1-score:  0.93646408839779

For Class 0
precision:  0.9520725388601037
recall:  0.9879032258064516
F1-score:  0.9696569920844327
----------testing-----------------------

For Class 1
precision:  0.9629629629629629
recall:  0.9069767441860465
F1-score:  0.934131736526946

For Class 0
precision:  0.9597989949748744
recall:  0.9845360824742269
F1-score:  0.9720101781170485
```

(iv) Difference between Decision Tree and Random Forest

Decision Tree 是以樹狀為基礎的演算法，透過歸納規則將資料從樹根開始分類，一節一節尋找最佳分割點來將資料分成為小單位的集合，中間有時也會透過園丁修剪，而成為一顆樹形美麗的決策樹。

不過當訓練資料集內的數目太少，而變數太多時，分類的效果會變差，另外，決策樹在分類上屬於固定的路徑，沒辦法像類神經在分類過程有容錯能力，所以會有overfitting的狀況產生。

Random Forest是以隨機(重新抽樣)的方法種植出許多決策樹，樹的集合就是森林，接著從決策樹們的投票結果中選出票數最多的候選人作為本屆選舉結果。由於是基於隨機抽樣及多顆樹，效果會比決策樹好許多，但相對運算的時間也比較長。

(v) Decision Tree model

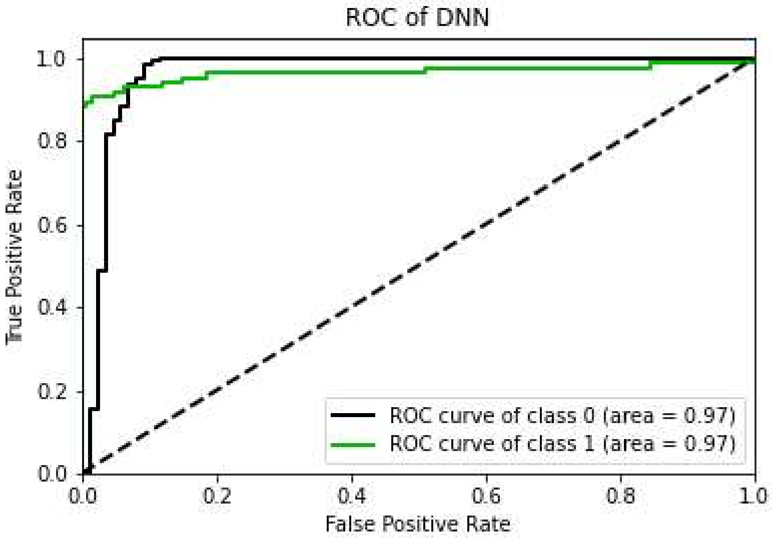
```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=3,min_samples_leaf=5,criterion='entropy'
tree.fit(x_train,y_train)

cm_train_tree = confusion(tree,x_train, y_train)
cm_test_tree = confusion(tree,x_test, y_test)

## Precision, recall, F1-score from decision tree
print("----------Decision Tree Training-----------")
print("acc_train=",tree.score(x_train,y_train))
calculation(cm_train_tree)
print("----------Test--------------------")
print("acc_test=",tree.score(x_test,y_test))
calculation(cm_test_tree)

#############################
output:
----------Decision Tree traing-----------

For Class 1
precision:  0.9938461538461538
recall:  0.8590425531914894
F1-score:  0.9215406562054208

For Class 0
precision:  0.9333333333333333
recall:  0.9973118279569892
F1-score:  0.9642625081221572
acc_train= 0.9508928571428571
----------Test--------------------

For Class 1
precision:  1.0
recall:  0.8837209302325582
F1-score:  0.9382716049382717

For Class 0
precision:  0.9509803921568627
recall:  1.0
F1-score:  0.9748743718592965
acc_test= 0.9642857142857143
```

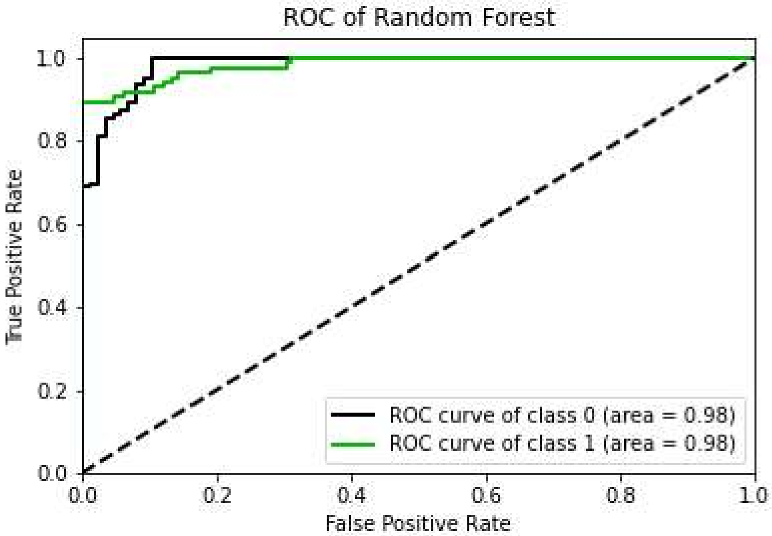
(v) Random Forest model

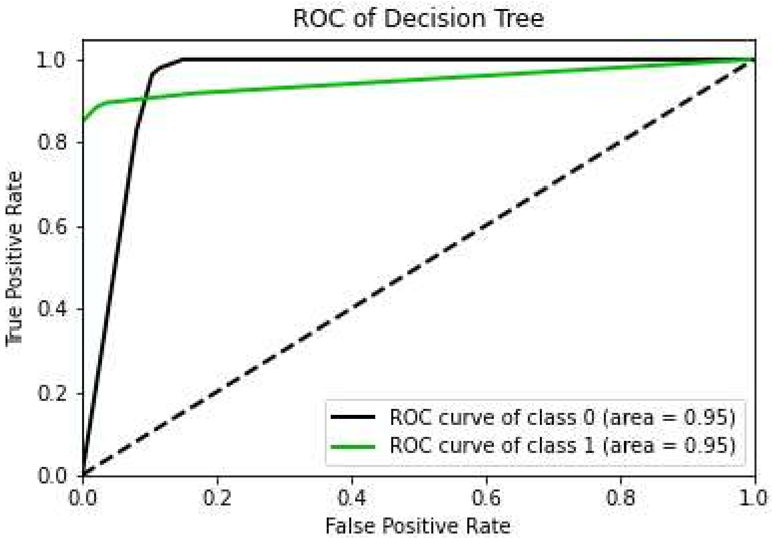
```
## Random Forest model
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(n_estimators=100, max_depth=4)
forest.fit(x_train, y_train)

cm_train_forest = confusion(forest, x_train, y_train)
cm_test_forest = confusion(forest, x_test, y_test)
print("----------Random Forest Training-----------")
calculation(cm_train_forest,class_=1)
calculation(cm_train_forest,class_=0)
print("acc_train=",forest.score(x_train,y_train))

print("----------Test-------------------------------")
calculation(cm_test_forest,class_=1)
calculation(cm_test_forest,class_=0)
print("acc_test=",forest.score(x_test,y_test))
########################
output:
----------Random Forest Training-----------

For Class 1
precision:  0.9939577039274925
recall:  0.875
F1-score:  0.9306930693069307

For Class 0
precision:  0.9404309252217997
recall:  0.9973118279569892
F1-score:  0.9680365296803654
acc_train= 0.95625
----------Test-------------------------------

For Class 1
precision:  1.0
recall:  0.8837209302325582
F1-score:  0.9382716049382717

For Class 0
precision:  0.9509803921568627
recall:  1.0
F1-score:  0.9748743718592965
acc_test= 0.9642857142857143

```

(vi) ROC from DNN, Decision Tree, Random Forest

```
## ROC Curve from DNN, Decision Tree and Random Forest
!pip install scikit-plot
import scikitplot as skplt

def roc_curve(model, x_test, y_test, model_name = "model_name"):
  y_probs = model.predict_proba(x_test)
  skplt.metrics.plot_roc(y_test, y_probs, plot_micro=False, plot_macro=False)
  plt.title("ROC of "+ model_name)
  plt.savefig("ROC of "+ model_name)
  plt.show(block=False)

#Plot
roc_curve(grid_best, x_test, y_test, model_name = "DNN")
roc_curve(tree, x_test, y_test, model_name = "Decision Tree")
roc_curve(forest, x_test, y_test, model_name = "Random Forest")
```



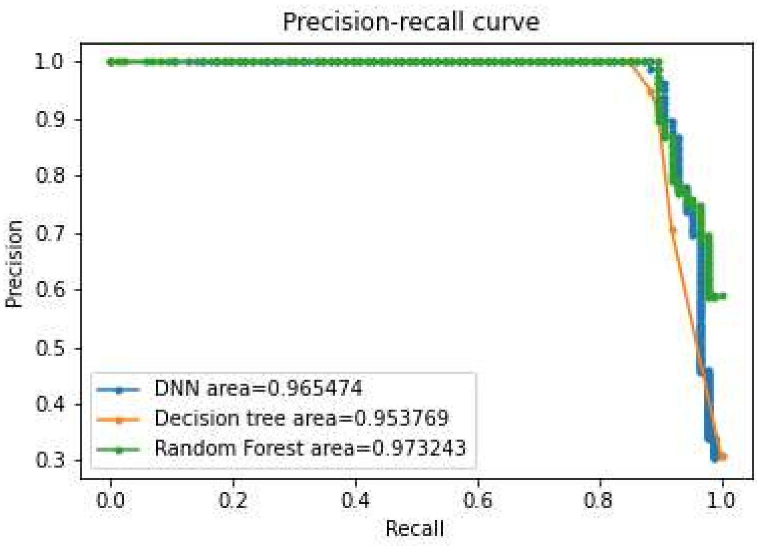




可以看出Random forest的效果最好(可能也跟我的參數有關)，而DNN model在調參過後效果也不錯。

(vi) Precision-recall curve from DNN, Decision Tree, Random Forest

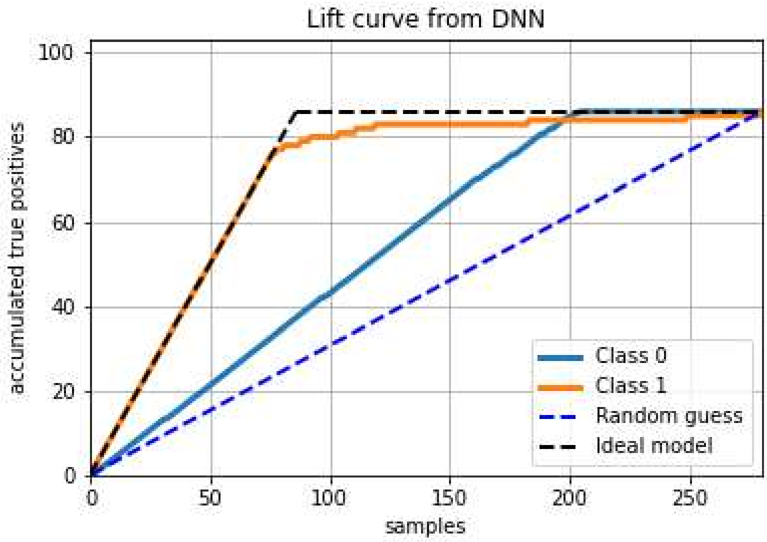
```
## Precision-Recall curve from DNN, Decision Tree and Random Forest
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc

def plot_precisionrecall_curve(model,x_test,y_test, model_name="model"):
  y_probs = model.predict_proba(x_test)
  # y_probs_tree = tree.predict_proba(x_test)
  # y_probs_forest = forest.predict_proba(x_test)

  precision, recall, _ = precision_recall_curve(y_test, y_probs[:,1])
  # precision_tree, recall_tree, _ = precision_recall_curve(y_test, y_probs_tree
  # precision_forest, recall_forest, _ = precision_recall_curve(y_test, y_probs_
  plt.plot(recall, precision, marker='.', label=model_name + ' area=%f'%auc(reca

# Plot
plot_precisionrecall_curve(grid_best,x_test,y_test,model_name="DNN")
plot_precisionrecall_curve(tree,x_test,y_test,model_name="Decision tree")
plot_precisionrecall_curve(forest,x_test,y_test,model_name="Random Forest")
plt.title("Precision-recall curve ")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
plt.savefig("Precision-recall curve")
plt.show()
```

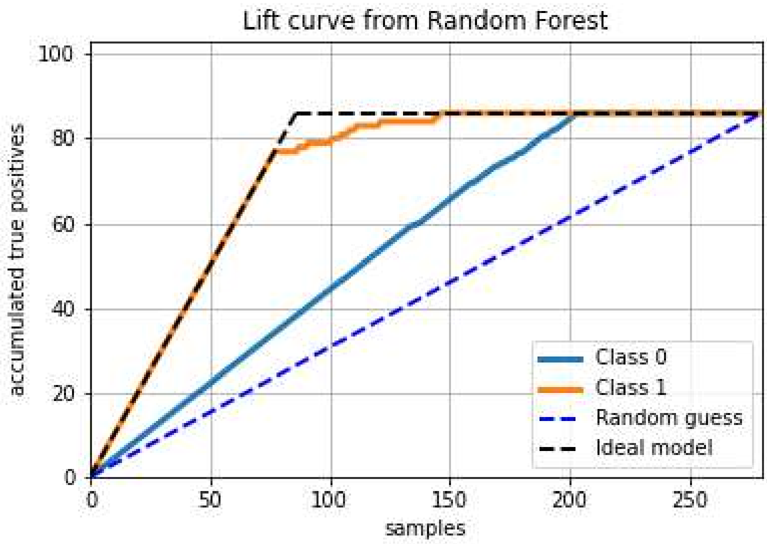


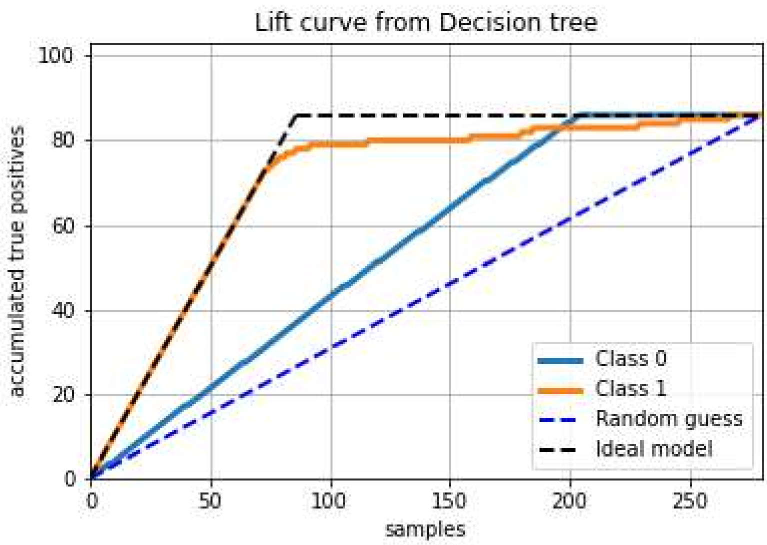
Problem 2.(a) Lift curve

根據助教給的hw2.pdf，其中對於lift curve的定義比較像是不同scale之下的cumulative gain curve，於是我修改原本scikitplot的plot_cumulative_gain使其符合作業中的敘述。

```
## Lift curve (modified by cumulative gain curve)
from scikitplot.helpers import cumulative_gain_curve
def plot_cumulative_gain(model, x_test ,y_true, model_name="model_name",
                         ax=None, figsize=None, title_fontsize="large",
                         text_fontsize="medium"):
    y_true = np.array(y_true)
    y_probas = model.predict_proba(x_test)
    y_probas = np.array(y_probas)

    classes = np.unique(y_true)
    if len(classes) != 2:
        raise ValueError('Cannot calculate Cumulative Gains for data with '
                         '{} category/ies'.format(len(classes)))

    # Compute Cumulative Gain Curves
    total = len(y_true)
    true_positive = y_true.sum()

    percentages1, gains1 = cumulative_gain_curve(y_true, y_probas[:, 0],
                                               classes[0])
    percentages2, gains2 = cumulative_gain_curve(y_true, y_probas[:, 1],
                                                classes[1])
    # print(gains1)
    if ax is None:
        fig, ax = plt.subplots(1, 1, figsize=figsize)

    ax.set_title("Lift curve from "+model_name, fontsize=title_fontsize)

    ax.plot(percentages1*total, gains1*true_positive, lw=3, label='Class {}'.for
    ax.plot(percentages2*total, gains2*true_positive, lw=3, label='Class {}'.for

    ax.set_xlim([0.0, total])
    ax.set_ylim([0.0, true_positive*1.2])

    ax.plot([0, total], [0, true_positive], 'b--', lw=2, label='Random guess')
    ax.plot([0, true_positive], [0, true_positive], 'k--', lw=2, label='Ideal mo
    ax.plot([true_positive, total], [true_positive, true_positive], 'k--', lw=2)
    ax.set_xlabel('samples', fontsize=text_fontsize)
    ax.set_ylabel('accumulated true positives', fontsize=text_fontsize)
    ax.tick_params(labelsize=text_fontsize)
    ax.grid('on')
    ax.legend(loc='lower right', fontsize=text_fontsize)
    plt.savefig("Lift curve from "+model_name)

    return ax

plot_cumulative_gain(grid_best,x_test,y_test, model_name='DNN')
plot_cumulative_gain(tree,x_test,y_test, model_name='Decision tree')
plot_cumulative_gain(forest,x_test,y_test, model_name='Random Forest')
```







- 可以看出 Class 1 是比 Class 0具有代表性的，這也是因為此dataset就是要預測是否為Class 1: Fraud的關係。
- 這三張圖看起來 Random Forest與 DNN 的預測效果比 Decision Tree好，與上面的ROC及Precision-recall都有一樣的結果。