R10H41003-HW2-游筑鈞

Answer the questions marked blue boldface in my report, Thks.

In [1]:
```
import os, time, glob, socket
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
import csv
import pickle
path=os.getcwd()
pd.set_option("display.max_columns", None)
import sys
import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score,accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import plot_roc_curve,roc_curve
from sklearn.metrics import average_precision_score, precision_recall_curve
from sklearn.metrics import auc, plot_precision_recall_curve
```

Read Data

In [3]:
```
df = pd.read_csv("Data.csv")
print(df.shape)
df.head()
```

```
(1400, 31)
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	127459.0	-0.489190	0.783289	-1.659097	-1.366814	2.576846	3.513254	0.271305	1.304641	-0.238853
1	128505.0	1.293556	-1.302381	-2.241085	0.393974	0.680825	0.821662	0.501478	0.104208	0.111651
2	128393.0	-0.755894	0.121305	0.852314	-2.303416	-0.233670	-0.244191	-0.285440	0.424009	-1.072689
3	128738.0	2.038750	-0.159488	-1.096570	0.425224	-0.214944	-1.151940	0.107112	-0.250273	0.701067
4	140293.0	0.951025	3.252926	-5.039105	4.632411	3.014501	-1.349570	0.980940	-1.819539	-2.099049

In [4]:
```
Y=df.Class
X=df.drop(['Class'], axis=1)
```

In [5]:
```
X_train, X_val, y_train, y_val = train_test_split(X, Y, test_size=0.2)
print(X_train.shape)
print(y_train.shape)
print(X_val.shape)
print(y_val.shape)
```

```
(1120, 30)
(1120,)
(280, 30)
(280,)
```

1-(i)

Please construct a DNN for binary classification according to the cross-entropy error function

You should decide the following hyperparameters:

- number of hidden layers
- number of hidden units
- learning rate
- number of iterations
- mini-batch size

In [6]:
```
from tensorflow import keras
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras import Sequential
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.models import load_model
```

In [7]:
```
import tensorflow
```

In [8]:
```
def create_model(learning_rate,activations,units,num_hidden_layers):
    # create model

    model = Sequential()# Initialize the constructor
    model.add(Dense(X_train.shape[1],input_dim=X_train.shape[1],kernel_initializer="rand
om_normal",activation=activations))

    for i in range(num_hidden_layers):
          # Add one hidden layer
        model.add(Dense(units, activation=activations))

    model.add(Dense(1, activation='sigmoid')) #output layer
    opt = tensorflow.keras.optimizers.Adam(learning_rate=learning_rate)

    # Compile model
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model
```

Grid_search

In [ ]:
```
# create model
model = KerasClassifier(build_fn=create_model)

# define the grid search parameters

batch_size = [10, 20,50]
epochs = [10,30,50]
learning_rate = [0.0001,0.001, 0.01, 0.1, 0.2, 0.3]
activations = ['softmax','sigmoid', 'relu']
units = [1, 5, 10, 15, 20,30]
num_hidden_layers=range(1,5)

param_grid = dict(batch_size=batch_size, epochs=epochs,learning_rate=learning_rate,activ
ations=activations,units=units,num_hidden_layers=num_hidden_layers)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=10)
grid_result = grid.fit(X_train, y_train)
```

In [27]:
```
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.923214 using {'activations': 'relu', 'batch_size': 20, 'epochs': 50, 'learning_
rate': 0.0001, 'num_hidden_layers': 2, 'units': 15}
```

fit model

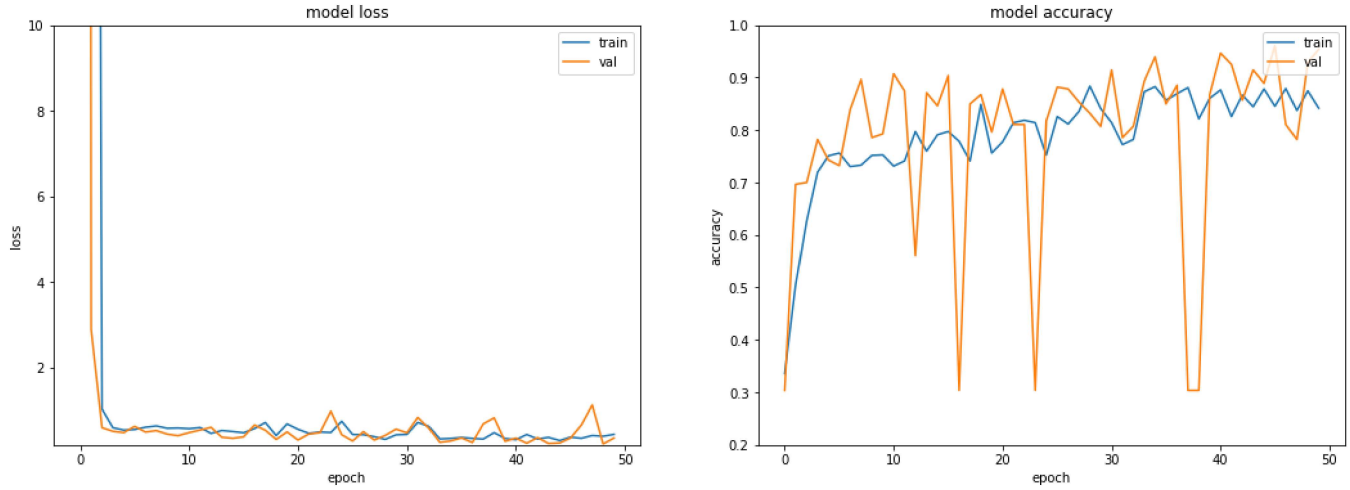
In [9]:
```
model = create_model(learning_rate= 0.0001,activations='relu',units=15,num_hidden_layers
=2)

DNN_model = model.fit(X_train, y_train,validation_data=(X_val,y_val), epochs= 50,
                      batch_size= 20, verbose=2)

y_dnn_pred=model.predict_classes(X_val).flatten()
y_dnn_true=model.predict_classes(X_train).flatten()
```

```
Epoch 1/50
56/56 - 1s - loss: 697.5054 - accuracy: 0.3357 - val_loss: 275.4269 - val_accuracy: 0.3
036
Epoch 2/50
56/56 - 0s - loss: 68.3296 - accuracy: 0.5045 - val_loss: 2.8939 - val_accuracy: 0.6964
Epoch 3/50
56/56 - 0s - loss: 1.0350 - accuracy: 0.6268 - val_loss: 0.6033 - val_accuracy: 0.7000
Epoch 4/50
56/56 - 0s - loss: 0.6031 - accuracy: 0.7196 - val_loss: 0.5192 - val_accuracy: 0.7821
Epoch 5/50
56/56 - 0s - loss: 0.5474 - accuracy: 0.7509 - val_loss: 0.4879 - val_accuracy: 0.7429
Epoch 6/50
56/56 - 0s - loss: 0.5608 - accuracy: 0.7563 - val_loss: 0.6351 - val_accuracy: 0.7321
Epoch 7/50
56/56 - 0s - loss: 0.6142 - accuracy: 0.7304 - val_loss: 0.5022 - val_accuracy: 0.8393
Epoch 8/50
56/56 - 0s - loss: 0.6448 - accuracy: 0.7330 - val_loss: 0.5357 - val_accuracy: 0.8964
Epoch 9/50
56/56 - 0s - loss: 0.5896 - accuracy: 0.7518 - val_loss: 0.4523 - val_accuracy: 0.7857
Epoch 10/50
56/56 - 0s - loss: 0.5960 - accuracy: 0.7527 - val_loss: 0.4165 - val_accuracy: 0.7929
Epoch 11/50
56/56 - 0s - loss: 0.5802 - accuracy: 0.7312 - val_loss: 0.4888 - val_accuracy: 0.9071
Epoch 12/50
56/56 - 0s - loss: 0.6065 - accuracy: 0.7411 - val_loss: 0.5473 - val_accuracy: 0.8750
Epoch 13/50
56/56 - 0s - loss: 0.4695 - accuracy: 0.7973 - val_loss: 0.6121 - val_accuracy: 0.5607
Epoch 14/50
56/56 - 0s - loss: 0.5372 - accuracy: 0.7598 - val_loss: 0.3802 - val_accuracy: 0.8714
Epoch 15/50
56/56 - 0s - loss: 0.5138 - accuracy: 0.7911 - val_loss: 0.3547 - val_accuracy: 0.8464
Epoch 16/50
56/56 - 0s - loss: 0.4844 - accuracy: 0.7973 - val_loss: 0.3891 - val_accuracy: 0.9036
Epoch 17/50
56/56 - 0s - loss: 0.5828 - accuracy: 0.7786 - val_loss: 0.6653 - val_accuracy: 0.3036
Epoch 18/50
56/56 - 0s - loss: 0.7239 - accuracy: 0.7411 - val_loss: 0.5436 - val_accuracy: 0.8500
Epoch 19/50
56/56 - 0s - loss: 0.4198 - accuracy: 0.8491 - val_loss: 0.3285 - val_accuracy: 0.8679
Epoch 20/50
56/56 - 0s - loss: 0.6916 - accuracy: 0.7563 - val_loss: 0.5040 - val_accuracy: 0.7964
Epoch 21/50
56/56 - 0s - loss: 0.5663 - accuracy: 0.7777 - val_loss: 0.3155 - val_accuracy: 0.8786
Epoch 22/50
56/56 - 0s - loss: 0.4719 - accuracy: 0.8143 - val_loss: 0.4551 - val_accuracy: 0.8107
Epoch 23/50
56/56 - 0s - loss: 0.5015 - accuracy: 0.8188 - val_loss: 0.4871 - val_accuracy: 0.8107
Epoch 24/50
56/56 - 0s - loss: 0.4903 - accuracy: 0.8143 - val_loss: 0.9859 - val_accuracy: 0.3036
Epoch 25/50
56/56 - 0s - loss: 0.7534 - accuracy: 0.7527 - val_loss: 0.4389 - val_accuracy: 0.8179
Epoch 26/50
56/56 - 0s - loss: 0.4444 - accuracy: 0.8259 - val_loss: 0.2919 - val_accuracy: 0.8821
Epoch 27/50
56/56 - 0s - loss: 0.4383 - accuracy: 0.8116 - val_loss: 0.5109 - val_accuracy: 0.8786
Epoch 28/50
56/56 - 0s - loss: 0.3931 - accuracy: 0.8357 - val_loss: 0.3135 - val_accuracy: 0.8536
Epoch 29/50
56/56 - 0s - loss: 0.3295 - accuracy: 0.8839 - val_loss: 0.4225 - val_accuracy: 0.8321
Epoch 30/50
56/56 - 0s - loss: 0.4366 - accuracy: 0.8411 - val_loss: 0.5693 - val_accuracy: 0.8071
```

```
Epoch 31/50
56/56 - 0s - loss: 0.4457 - accuracy: 0.8143 - val_loss: 0.4845 - val_accuracy: 0.9143
Epoch 32/50
56/56 - 0s - loss: 0.7262 - accuracy: 0.7723 - val_loss: 0.8425 - val_accuracy: 0.7857
Epoch 33/50
56/56 - 0s - loss: 0.6364 - accuracy: 0.7821 - val_loss: 0.5943 - val_accuracy: 0.8071
Epoch 34/50
56/56 - 0s - loss: 0.3407 - accuracy: 0.8732 - val_loss: 0.2614 - val_accuracy: 0.8929
Epoch 35/50
56/56 - 0s - loss: 0.3520 - accuracy: 0.8830 - val_loss: 0.2931 - val_accuracy: 0.9393
Epoch 36/50
56/56 - 0s - loss: 0.3741 - accuracy: 0.8571 - val_loss: 0.3627 - val_accuracy: 0.8500
Epoch 37/50
56/56 - 0s - loss: 0.3529 - accuracy: 0.8696 - val_loss: 0.2572 - val_accuracy: 0.8857
Epoch 38/50
56/56 - 0s - loss: 0.3376 - accuracy: 0.8813 - val_loss: 0.6919 - val_accuracy: 0.3036
Epoch 39/50
56/56 - 0s - loss: 0.4866 - accuracy: 0.8214 - val_loss: 0.8331 - val_accuracy: 0.3036
Epoch 40/50
56/56 - 0s - loss: 0.3511 - accuracy: 0.8607 - val_loss: 0.2860 - val_accuracy: 0.8679
Epoch 41/50
56/56 - 0s - loss: 0.3211 - accuracy: 0.8768 - val_loss: 0.3590 - val_accuracy: 0.9464
Epoch 42/50
56/56 - 0s - loss: 0.4443 - accuracy: 0.8259 - val_loss: 0.2406 - val_accuracy: 0.9250
Epoch 43/50
56/56 - 0s - loss: 0.3386 - accuracy: 0.8670 - val_loss: 0.3713 - val_accuracy: 0.8571
Epoch 44/50
56/56 - 0s - loss: 0.3775 - accuracy: 0.8446 - val_loss: 0.2344 - val_accuracy: 0.9143
Epoch 45/50
56/56 - 0s - loss: 0.3032 - accuracy: 0.8786 - val_loss: 0.2456 - val_accuracy: 0.8893
Epoch 46/50
56/56 - 0s - loss: 0.3831 - accuracy: 0.8455 - val_loss: 0.3610 - val_accuracy: 0.9607
Epoch 47/50
56/56 - 0s - loss: 0.3549 - accuracy: 0.8795 - val_loss: 0.6659 - val_accuracy: 0.8107
Epoch 48/50
56/56 - 0s - loss: 0.4210 - accuracy: 0.8375 - val_loss: 1.1288 - val_accuracy: 0.7821
Epoch 49/50
56/56 - 0s - loss: 0.4034 - accuracy: 0.8750 - val_loss: 0.2231 - val_accuracy: 0.9286
Epoch 50/50
56/56 - 0s - loss: 0.4474 - accuracy: 0.8420 - val_loss: 0.3609 - val_accuracy: 0.9536
```

In [10]:
```
plt.figure(figsize=(18,6))

plt.subplot(121)

plt.plot(DNN_model.history['loss'],label='train')
plt.plot(DNN_model.history['val_loss'],label='val')
plt.ylim(0.2,10)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.title('model loss')
plt.legend(loc=1)

plt.subplot(122)
plt.plot(DNN_model.history['accuracy'],label='train')
plt.plot(DNN_model.history['val_accuracy'],label='val')
plt.ylim(0.2,1)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.title('model accuracy')
plt.legend(loc=1)
```

Out[10]:
```
<matplotlib.legend.Legend at 0x1dc98f9bb88>
```

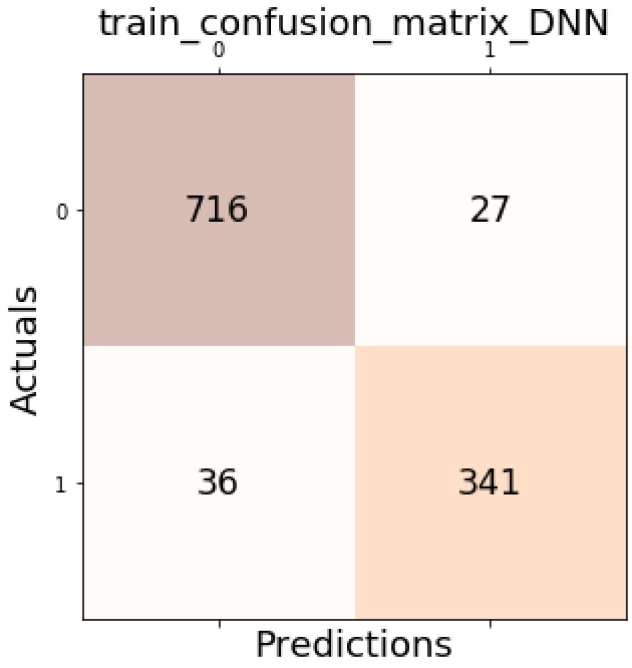


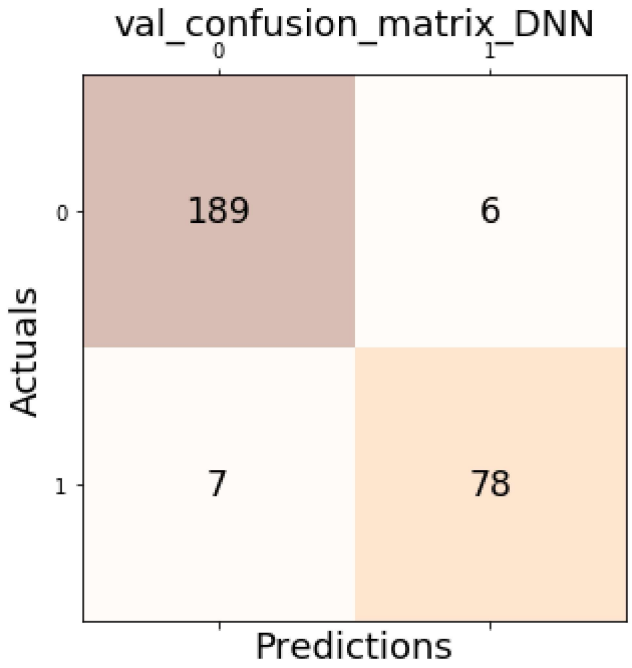
1-(ii)

Please plot the confusion matrices for (i) as the example in Figure 2.

In [11]:
```
def plot_confusion_matrix(y_true,y_pred,title):
    conf_matrix = confusion_matrix(y_true, y_pred)
    fig, ax = plt.subplots(figsize=(5, 5))
    ax.matshow(conf_matrix, cmap=plt.cm.Oranges, alpha=0.3)
    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-lar
ge')

    plt.xlabel('Predictions', fontsize=18)
    plt.ylabel('Actuals', fontsize=18)
    plt.title(title, fontsize=18)
    plt.show()
```

In [12]:
```
plot_confusion_matrix(y_train,y_dnn_true,'train_confusion_matrix_DNN')
plot_confusion_matrix(y_val,y_dnn_pred,'val_confusion_matrix_DNN')
```





1-(iii)

Precision, recall, F1-score are other ways to evaluate model performance on validation set. For each class, please record precision, recall and F1-score as well as the averages of those criteria over all classes in your report.

In [13]:
```
print('Precision: %.3f' % precision_score(y_val, y_dnn_pred))
print('Recall: %.3f' % recall_score(y_val, y_dnn_pred))
print('Accuracy: %.3f' % accuracy_score(y_val, y_dnn_pred))
print('F1 Score: %.3f' % f1_score(y_val, y_dnn_pred))
```

```
Precision: 0.929
Recall: 0.918
Accuracy: 0.954
F1 Score: 0.923
```

1-(iv)

What is the difference between decision tree and random forest?

隨機森林(RF, random forest)是用隨機的方式建立許多決策樹，而很多的決策樹組合起來就變成一個森林，對於分類算法來說，對隨機森林輸入新的樣本，森林裡的每個決策樹就會把該樣本分類，以投票的方式看哪一類別被選擇最多次，就預測那一類的樣本。

1-(v)

Please use decision tree and random forest to learn the binary classification task. Calculate the corresponding Accuracy, Precision, Recall and F1-Score on validation set.

Decision tree

In [91]:
```
dt_gsc = GridSearchCV(DecisionTreeClassifier(random_state=0),
              param_grid={
                  'min_samples_split': range(2, 10),
                  'max_depth': range(1,7),
                  "criterion":["gini", "entropy"] },
                   verbose=0,n_jobs=-1, cv=10, refit=True)

dt_gsc.fit(X_train,y_train)
dt_gsc.best_params_
```

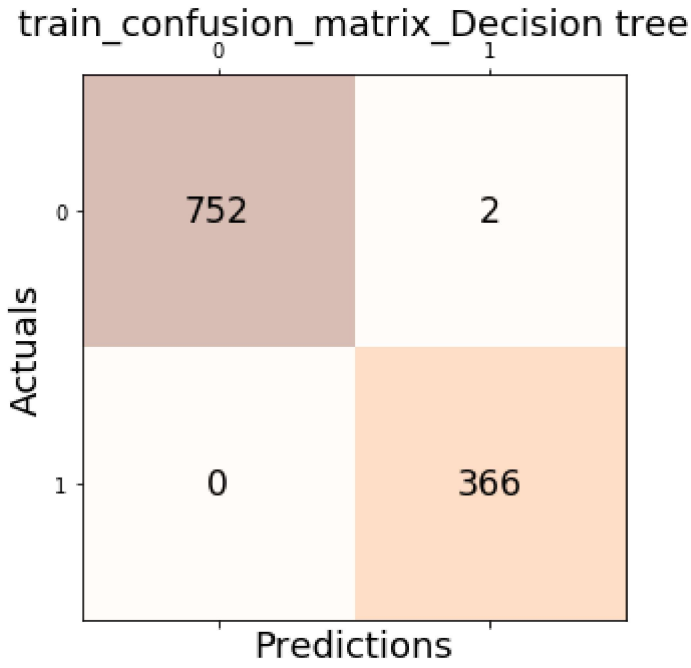
Out[91]:
```
{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2}
```

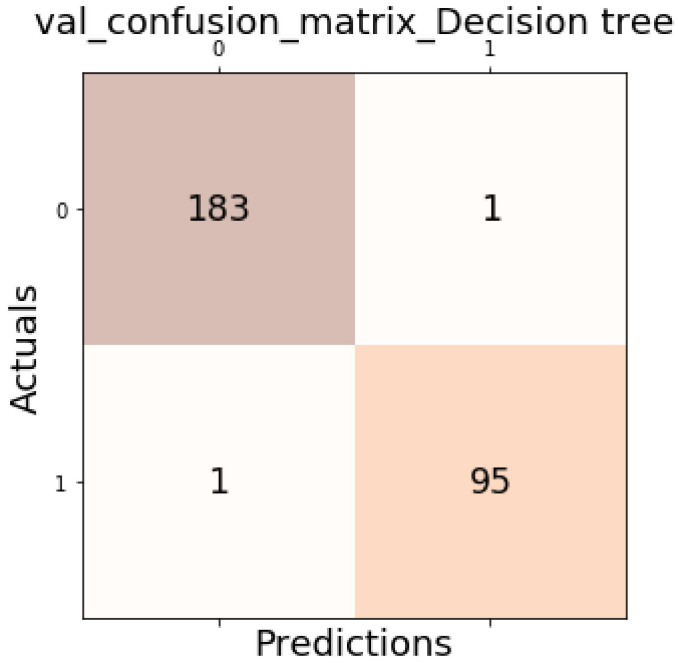
In [238]:
```
dt = DecisionTreeClassifier(criterion=dt_gsc.best_params_['criterion'],max_depth=dt_gsc.
best_params_["max_depth"],
                            min_samples_split=dt_gsc.best_params_["min_samples_split"])
dt.fit(X_train,y_train)
y_dt_pred=dt.predict(X_val).flatten()
y_dt_true=dt.predict(X_train).flatten()
```

In [239]:
```
print('Precision: %.3f' % precision_score(y_val, y_dt_pred))
print('Recall: %.3f' % recall_score(y_val, y_dt_pred))
print('Accuracy: %.3f' % accuracy_score(y_val, y_dt_pred))
print('F1 Score: %.3f' % f1_score(y_val, y_dt_pred))
```

```
Precision: 0.990
Recall: 0.990
Accuracy: 0.993
F1 Score: 0.990
```

In [240]:
```
plot_confusion_matrix(y_train,y_dt_true,'train_confusion_matrix_Decision tree')
plot_confusion_matrix(y_val,y_dt_pred,'val_confusion_matrix_Decision tree')
```





Random Forest

In [96]:
```
gsc = GridSearchCV(estimator=RandomForestClassifier(),param_grid={
            'max_features':['sqrt','auto','log2',0.5,0.7],
            'max_depth': range(1,6),
            'n_estimators': range(1,101,10)},
             cv=10,verbose=0,n_jobs=-1,)

grid_result = gsc.fit(X_train,y_train)
best_params = grid_result.best_params_
print(best_params)
```

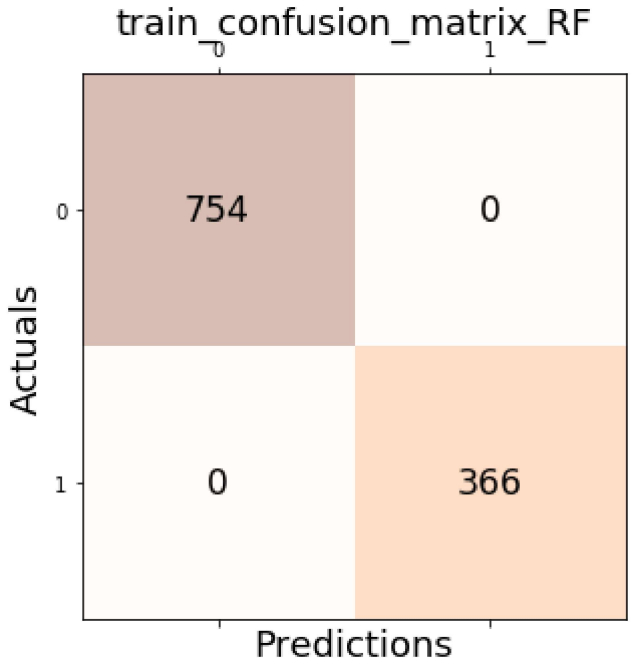
```
{'max_depth': 4, 'max_features': 0.7, 'n_estimators': 11}
```

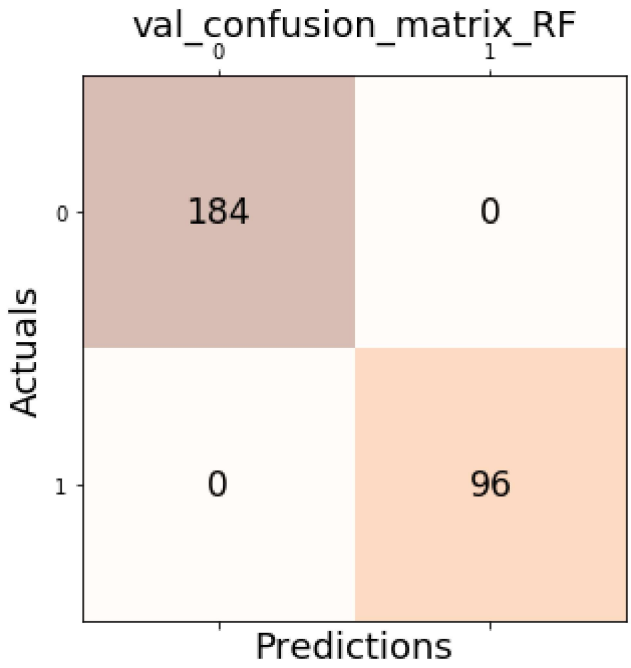
In [241]:
```
rfr = RandomForestClassifier(
    max_depth=best_params["max_depth"],
    n_estimators=best_params["n_estimators"],
    max_features=best_params['max_features'],
    random_state=False, verbose=False)

rfr.fit(X_train,y_train)

## predict_y
y_rfr_pred=rfr.predict(X_val).flatten()
y_rfr_true=rfr.predict(X_train).flatten()
```

In [242]:
```
print('Precision: %.3f' % precision_score(y_val, y_rfr_pred))
print('Recall: %.3f' % recall_score(y_val, y_rfr_pred))
print('Accuracy: %.3f' % accuracy_score(y_val, y_rfr_pred))
print('F1 Score: %.3f' % f1_score(y_val, y_rfr_pred))
```

```
Precision: 1.000
Recall: 1.000
Accuracy: 1.000
F1 Score: 1.000
```

In [243]:
```
plot_confusion_matrix(y_train,y_rfr_true,'train_confusion_matrix_RF')
plot_confusion_matrix(y_val,y_rfr_pred,'val_confusion_matrix_RF')
```





1-(vi)

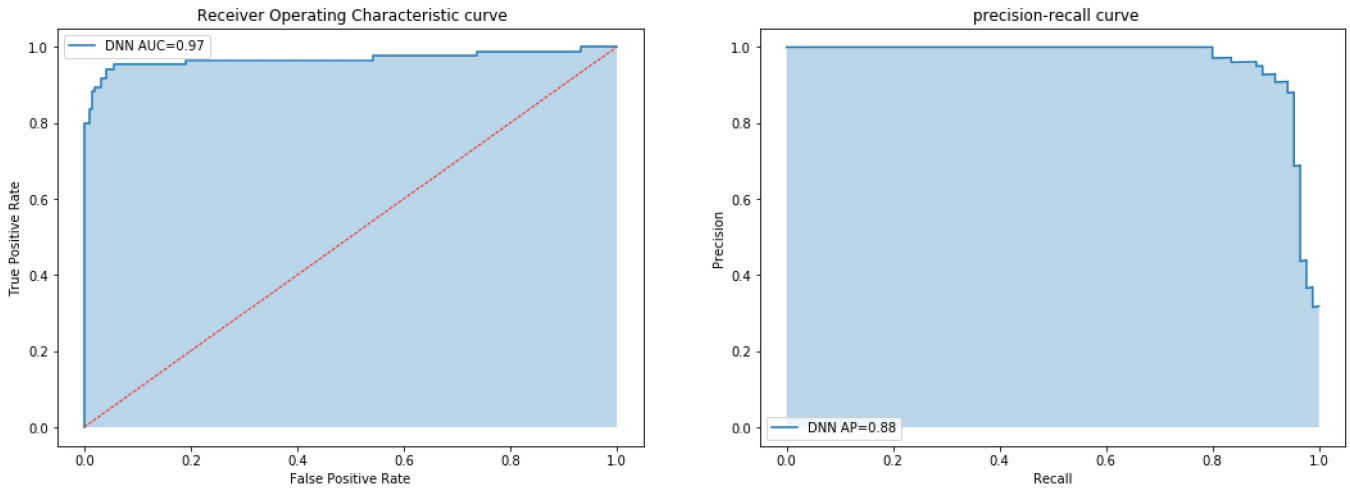
You have to plot the receiver operating characteristic curve (ROC, as shown in Figure 3) and precision-recall curve (PRC, as shown in Figure 3) with their area-under-curve (AUROC and AUPRC) for DNN, decision tree and, random forest on the validation set

In [14]:
```
def plot_static_roc_curve(y_val,y_score,model):

    fpr,tpr,threshold = roc_curve(y_val, y_score)
    plt.plot(fpr, tpr,label=model+' AUC=%0.2f' % auc(fpr, tpr))
    plt.fill_between(fpr, tpr, alpha=.3)

    plt.plot([0,1], [0,1], color="red",linewidth=0.8,linestyle='dashed')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(" Receiver Operating Characteristic curve");
    plt.legend()
```

In [15]:
```
def plot_static_PR_curve(y_val,y_score,y_pred,model):

    AP=average_precision_score(y_val,y_pred)
    precision,recall,thresholds =  precision_recall_curve(y_val, y_score)

    plt.plot(recall,precision,label=model+' AP=%0.2f' % AP)
    plt.fill_between(recall,precision, alpha=.3)
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title(" precision-recall curve");
    plt.legend()
```

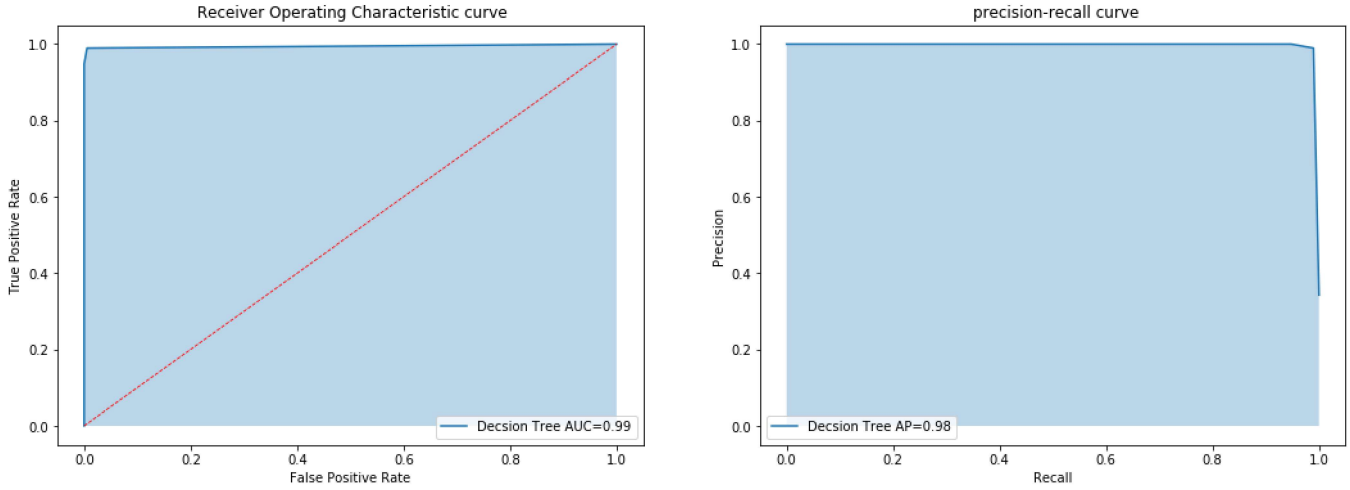
DNN

In [16]:
```
y_score = model.predict(X_val).flatten()

plt.figure(figsize=(18,6))
plt.subplot(121)
plot_static_roc_curve(y_val,y_score,"DNN")
plt.subplot(122)
plot_static_PR_curve(y_val,y_score,y_dnn_pred,"DNN")
```



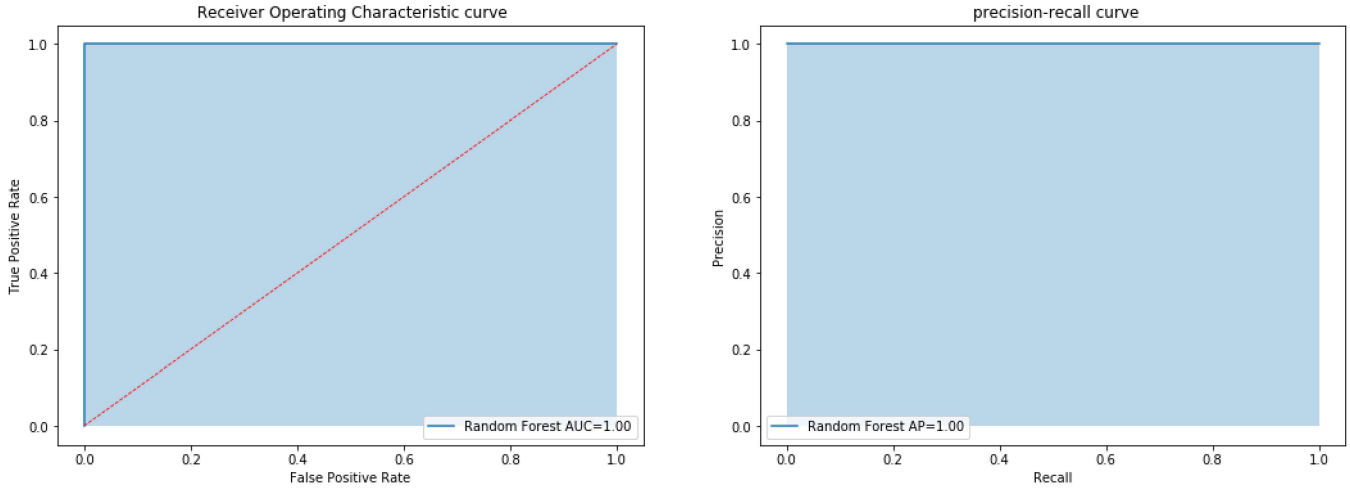
Decsion Tree

In [134]:
```
y_score = dt.predict_proba(X_val)[:, 1]

plt.figure(figsize=(18,6))
plt.subplot(121)
plot_static_roc_curve(y_val,y_score,"Decsion Tree")
plt.subplot(122)
plot_static_PR_curve(y_val,y_score,y_dt_pred,"Decsion Tree")
```



Random Forest

In [136]:
```
y_score = rfr.predict_proba(X_val)[:, 1]

plt.figure(figsize=(18,6))
plt.subplot(121)
plot_static_roc_curve(y_val,y_score,"Random Forest")
plt.subplot(122)
plot_static_PR_curve(y_val,y_score,y_rfr_pred,"Random Forest")
```



In [ ]:

In [ ]: