In [1]:
```
import numpy as np
import pandas as pd
from sklearn import tree,ensemble
```

Load dataset

In [2]:
```
df=pd.read_csv('loandata.csv')
```

Translate categorical predictors to numerical predictors

In [3]:
```
df=pd.get_dummies(df)#encode categorical variables
df=df.drop(['Default_No'], axis=1) #only need Default_Yes
```

Shuffle and split the dataset

In [17]:
```
Xy.shape
```

Out[17]: (2000, 9)

In [18]:
```
trainsize = 1000
trainplusvalsize = 1500
```

In [19]:
```
Xy=np.array(df)
#shuffling
np.random.shuffle(Xy)


X=Xy[:,:-1]
y=Xy[:,-1]
X_train=X[:trainsize,:]
X_val=X[trainsize:trainplusvalsize,:]
X_test=X[trainplusvalsize:,:]
y_train=y[:trainsize]
y_val=y[trainsize:trainplusvalsize]
y_test=y[trainplusvalsize:]
```

Accuracy of majority predictor

In [21]:
```
print ( 'naive guess train/validation', 1-sum(y_train)/len(y_train)
```

```
naive guess train/validation 0.781 0.756
```

Train decision tree

In [27]:

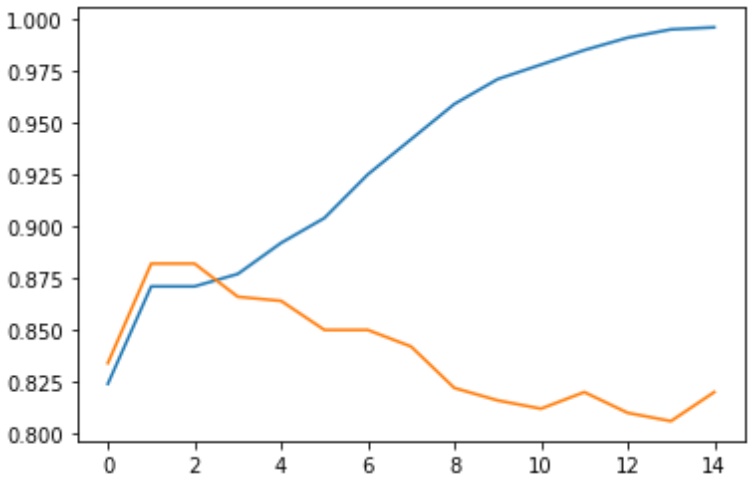
```
clf = tree.DecisionTreeClassifier()
clf.fit(X_train,y_train)

print ( 'full tree guess train/validation ',clf.score(X_train,y_trai

#try different tree depths and pick best

#bestdepth=-1
#bestscore=0
train_score = []
val_score = []
for i in range(15): #choose appropriate range (not more than 20)
    clf = tree.DecisionTreeClassifier(max_depth=i+1)
    clf.fit(X_train,y_train)
    trainscore=clf.score(X_train,y_train)
    train_score.append(trainscore)
    valscore=clf.score(X_val,y_val)
    val_score.append(valscore)
    print ( 'depth',i+1,trainscore,valscore)
```

```
full tree guess train/validation  1.0 0.806
depth 1 0.824 0.834
depth 2 0.871 0.882
depth 3 0.871 0.882
depth 4 0.877 0.866
depth 5 0.892 0.864
depth 6 0.904 0.85
depth 7 0.925 0.85
depth 8 0.942 0.842
depth 9 0.959 0.822
depth 10 0.971 0.816
depth 11 0.978 0.812
depth 12 0.985 0.82
depth 13 0.991 0.81
depth 14 0.995 0.806
depth 15 0.996 0.82
```

In [30]:

```
import matplotlib.pyplot as plt
plt.plot(train_score)
plt.plot(val_score)
```

Out[30]: [<matplotlib.lines.Line2D at 0x7f7d554999d0>]



appropriate depth is 1 or 2 and beyond 2, overfitting occurs

In [31]:
```
bestdepth = 2
```

In [47]:
```
#evaluate test set on best classifier, train on both train and valid
X_trainval=X[:trainplusvalsize,:]
y_trainval=y[:trainplusvalsize]
clf = tree.DecisionTreeClassifier(max_depth=bestdepth)
import time
start = time.time()
clf.fit(X_trainval,y_trainval)
stop = time.time()
print(f"Training time: {stop - start}s")
print('train set score', clf.score(X_train,y_train))
print('val set score', clf.score(X_val,y_val))
print('testing set score', clf.score(X_test,y_test))
```

```
Training time: 0.0028586387634277344s
train set score 0.871
val set score 0.882
testing set score 0.862
```

comments on classifier: comment on depth and complexity, comment on overfitting (compare train and val scores), compare to majority predictor, compare to a fully grown tree

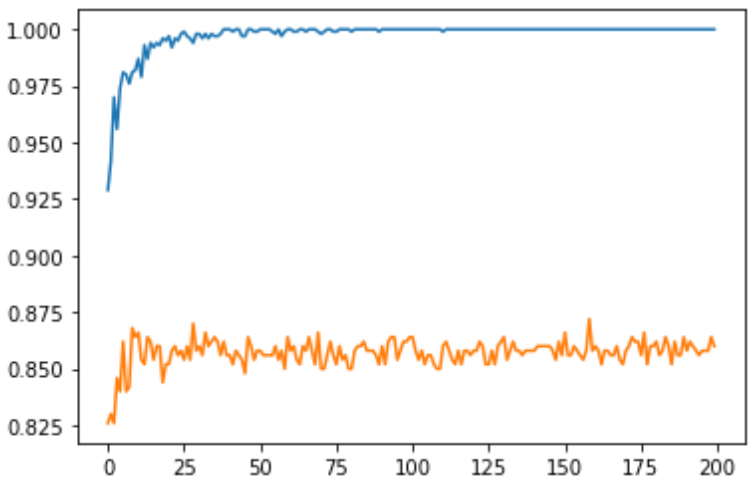
Train random forest

In [49]:
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=0)
clf.fit(X_train,y_train)

print ( 'full tree guess train/validation ',clf.score(X_train,y_trai

#try different tree depths and pick best

#bestdepth=-1
#bestscore=0
train_score = []
val_score = []
time_score = []
for i in range(200): #choose appropriate range (more than 100)
    clf = RandomForestClassifier(n_estimators=i+1)
    start = time.time()
    clf.fit(X_train,y_train)
    stop = time.time()
    traintime = stop - start
    time_score.append(traintime)
    trainscore=clf.score(X_train,y_train)
    train_score.append(trainscore)
    valscore=clf.score(X_val,y_val)
    val_score.append(valscore)
    #print ( 'n_estimators',i+1,trainscore,valscore)
```

```
/cvmfs/sft.cern.ch/lcg/views/LCG_99/x86_64-centos7-gcc8-opt/lib/pytho
n3.8/site-packages/sklearn/ensemble/forest.py:244: FutureWarning: The
default value of n_estimators will change from 10 in version 0.20 to
100 in 0.22.
  warn("The default value of n_estimators will change from "
full tree guess train/validation  0.989 0.854
```

In [50]:
```
plt.plot(train_score)
plt.plot(val_score)
```

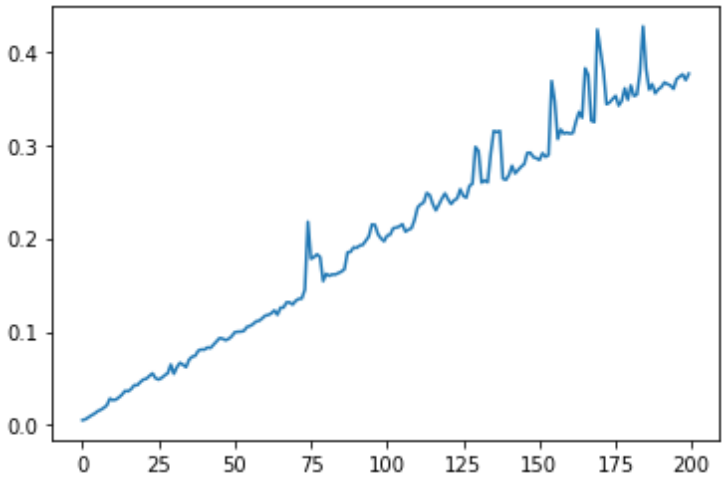
Out[50]:
```
[<matplotlib.lines.Line2D at 0x7f7d552e9700>]
```



Note: The students needs to notice the consistent overfitting over the range of estimators. This is because the default depth of the tree is too big. Bonus marks for changing the depth of the random forest trees and obtaining more reasonable results.

In [51]:
```
plt.plot(time_score)
```

Out[51]:
```
[<matplotlib.lines.Line2D at 0x7f7d552c2d00>]
```



choose n_estimators

Performance seems to peak around 20 (refer to plot), comment on overfitting, comment on systematic disparity between train and val score and justify, choose n_estimators with consideration to these points and to the training time.

In [44]:
```
best_est = 20 #anything between 15 and 25 is ok
```

In [48]:
```
#evaluate test set on best classifier, train on both train and valid
X_trainval=X[:trainplusvalsize,:]
y_trainval=y[:trainplusvalsize]
clf = RandomForestClassifier(n_estimators=20)

start = time.time()
clf.fit(X_trainval,y_trainval)
stop = time.time()
print(f"Training time: {stop - start}s")
print('train set score', clf.score(X_train,y_train))
print('val set score', clf.score(X_val,y_val))
print('testing set score', clf.score(X_test,y_test))
```

```
Training time: 0.06735062599182129s
train set score 0.994
val set score 0.998
testing set score 0.838
```

Comparisons

Comments should address performance, interpretability, training time and generalisability:

1. performance: compare scores between two models and to the majority predictor, comment on train, validation and test scores (recall systematic disparity in random forest)
2. interpretability: discussion of feature importance and variance Random Forest ultimately works by first reducing bias towards the locally optimal strategy of individual trees by splitting over features randomly, and then by aggregating trees to reduce the overall variance of the model.

This reduction in variance stabilises the model, reduces its bias towards choices of training data, and leads to less variable and more accurate predictions. If, as we do with our predictions, we aggregate our measures of feature importance over the trees, in a Random Forest, by taking the mean value for the change in entropy or accuracy attributed to each feature variable, we achieve exactly the same effect.

1. training time: justify differences in training time between models and between different number of estimators in the random forest. Bonus points for investigating the depth random forest and comparing to the DT.

2. generalisability: computes and compares generalisation error for both models, comment on overfitting

In [ ]: