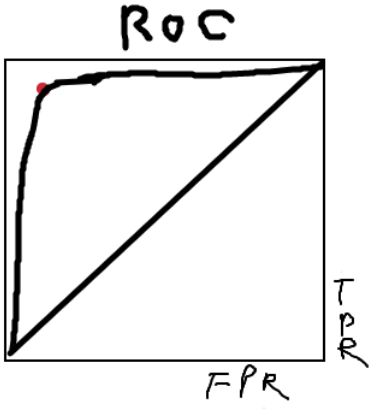
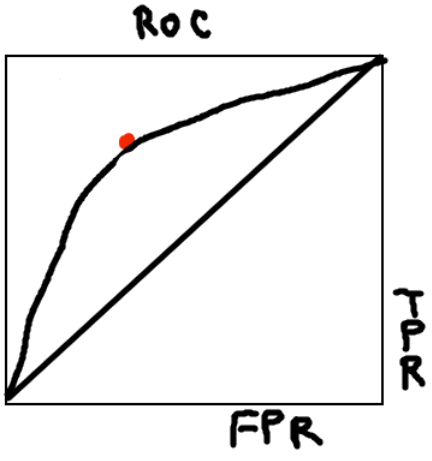
1)
a)
True positive rate = true positive / (true positive + false negative)
= 250/ (250 + 50) = 0.833
False positive rate = false positive / (false positive + true negative)
= 50 / (50 + 250) = 0.166
Positon Roc (FPR, TPR)=(0.166, 0.833)

b)



c)
True positive rate = true positive / (true positive + false negative)
= 200/ (200 + 100) = 0.666
False positive rate = false positive / (false positive + true negative)
= 100 / (100 + 200) = 0.333
Positon Roc (FPR, TPR)=(0.333, 0.666)

d)



e)
The first model is clearly better. The model better separates the blue and red. The second model has a lot more false positive and false negatives. Also the ROC curve on the second model is a lot flatter, which means it's closer to just randomly guessing.

f)
I think making the decision boundary 0.5 is a good choice. The 2 distributions are symmetric and 0.5 is right in the middle of the overlap. There is an equal number of false positives and false negatives.

2)

More bias less bias

Less variance more variance

<---------- >

less flexible (lower dimensionality) more flexible (higher dimensionality)

3) (From Jupyter Notebook)

Implementation with Pipeline

In [8]:
```
# TODO :: Implement the above task by using Pipeline. Your implementation should have the same test accuracy as the
# implementation without Pipeline. 9 lines of code expected.

from sklearn.pipeline import make_pipeline

pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(random_state=1))

pipe_lr.fit(X_train, y_train)
y_pred = pipe_lr.predict(X_test)

print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
```

```
Test Accuracy: 0.956
```

Question 1

Did you notice that the final CVaccuracy of implementation with Pipeline and without Pipeline is different? I made a common mistake in cross validation without Pipeline code. Can you help me to fix this bug and explain why it is a problem?

```
# TODO :: Correct the implementation without pipeline, 13-20 lines of code expected.

# The problem is didn't standardize. Need to do stder transform and pca transform.
# scaling is important since some features with large numbers can be seen as more important.
# After adding it, the CVaccuracy is the same as with the pipeline.

import numpy as np
from sklearn.model_selection import StratifiedKFold


X_train_std = stder.fit_transform(X_train)
X_train_pca = pca.fit_transform(X_train_std)

kfold= StratifiedKFold(n_splits=10,random_state=1).split(X_train_std, y_train)
scores = []
stder = StandardScaler()
pca = PCA(n_components=2)

for k, (train, test) in enumerate(kfold):
    X_train_std = stder.fit_transform(X_train[train])
    X_test_std = stder.transform(X_train[test])

    X_train_pca = pca.fit_transform(X_train_std)
    X_test_pca = pca.transform(X_test_std)

    lr = LogisticRegression()
    lr.fit(X_train_pca, y_train[train])
    score = lr.score(X_test_pca, y_train[test])
    scores.append(score)
    print('Fold: %2d, Class dist.: %s, Acc: %.3f' % (k+1,np.bincount(y_train[train]), score))
    print('CVaccuracy: %.3f +/-%.3f' % (np.mean(scores), np.std(scores)))
```

Question 2

The above plots show the score and score difference of cross validation versus nested cross validation. What obervation can be made in terms of the score of the two methods. Why does that happen? Which one do you think is a better way to evaluate the performance of the model? Why do you think so?

The nested CV score is lower than the non nested CV score. The nested CV better fits the model with the inner loop. The non nested CV can overfit and be overly optimistic. I believe the nested CV is a better way to evaluate the model as it reduces bias.

Question 3

Read the code above especially in the loop for each trial. Explain what does this line : nested_score = cross_val_score(clf, X=X_iris, y=y_iris, cv=outer_cv) do?

It does the nested cross validation using the clf (GridSearchCV with param grid p_grid) in the inner loop to the get best estimator for each run of the cross validation.