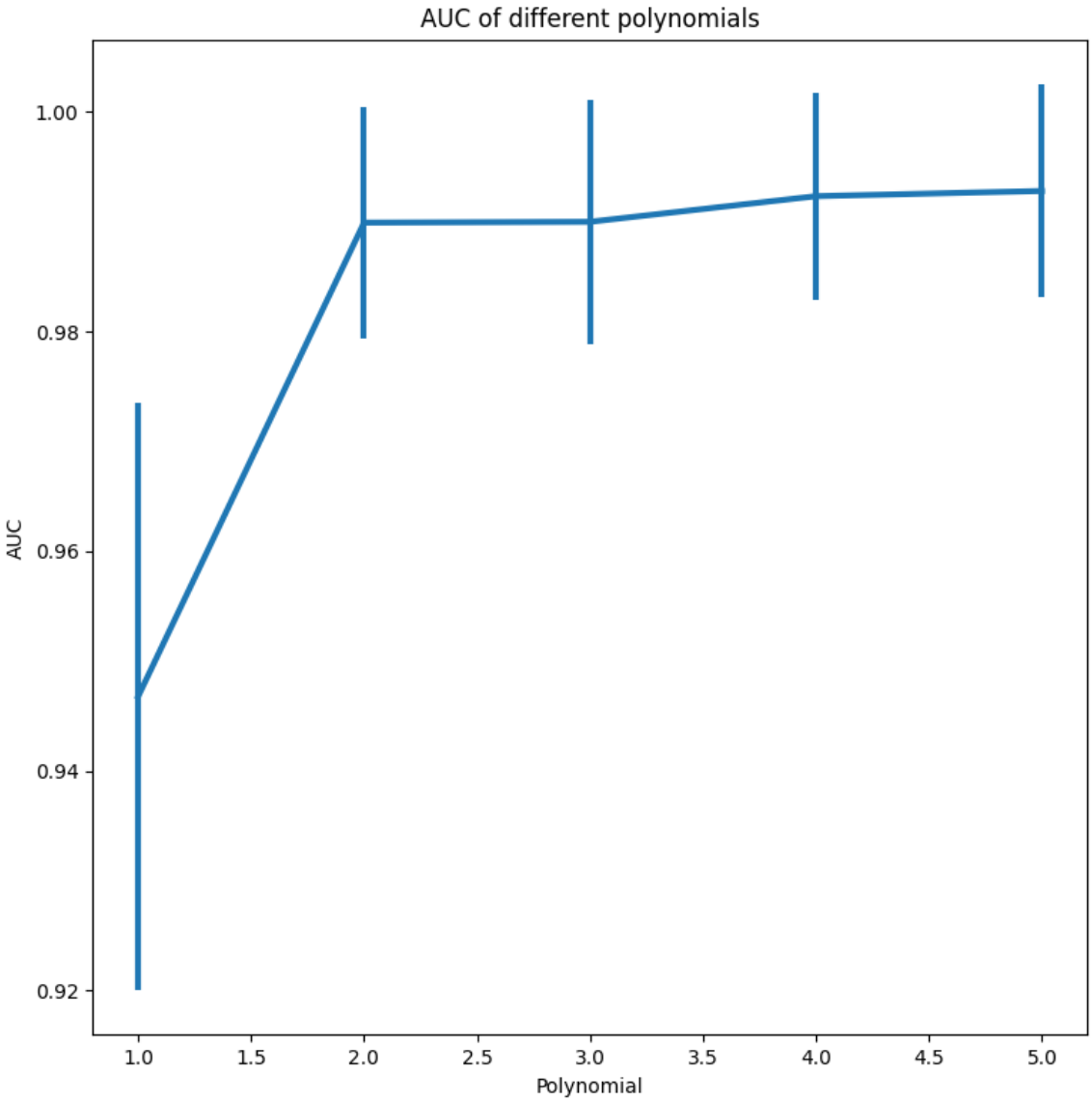
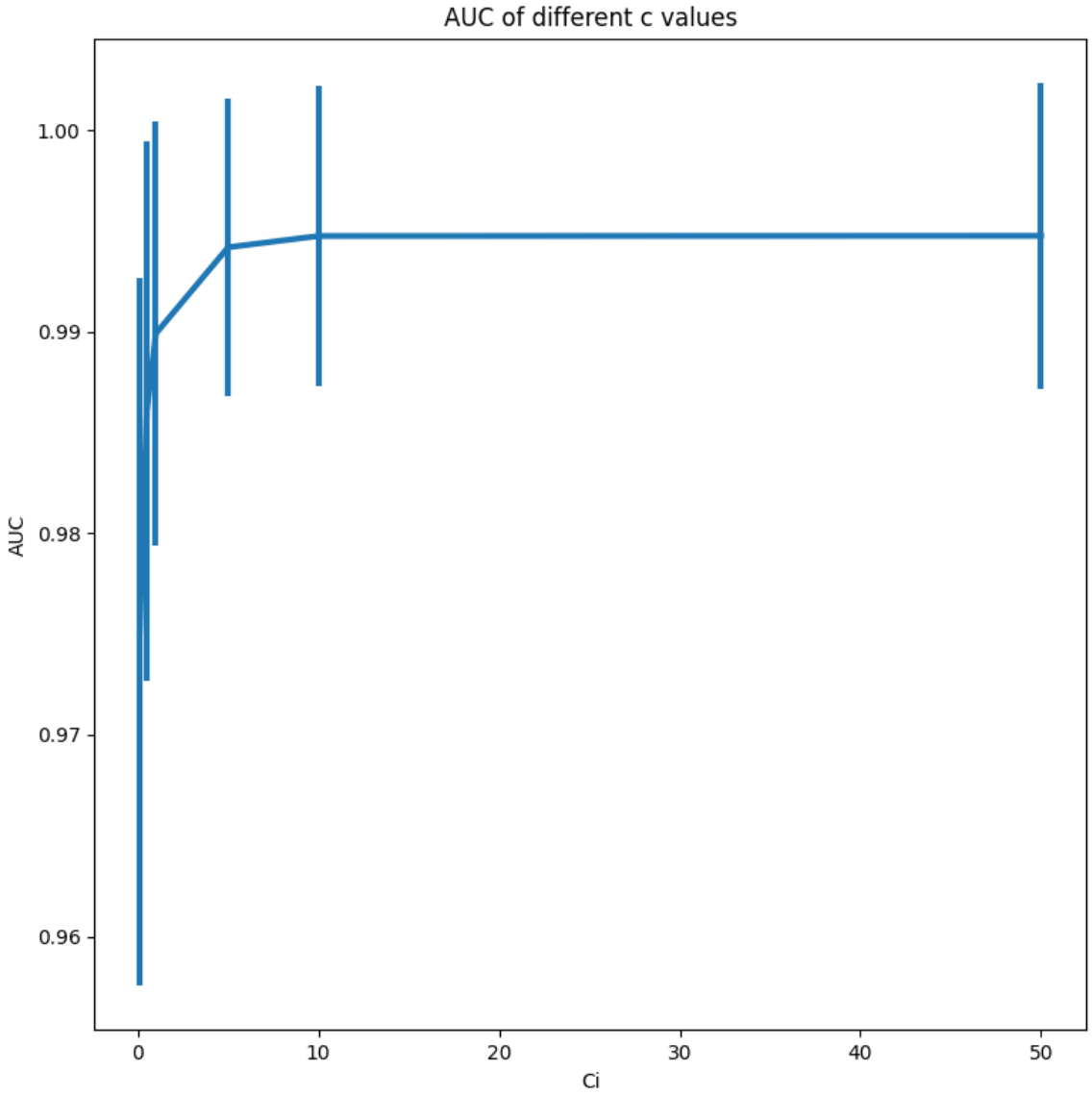
 # id: # id:14--14-14-0

(i) (a) When comparing both differing polynomials and differing c values, I used cross_val_score with auc scoring, this will compare the measure of separability of our model with different hyper-parameters, the closer to 1 the better. However, just because the auc score is larger, doesn't mean it is necessarily better, if the value of a larger polynomial is only slightly better it would not be worth using, as we would increase the chances of over-fitting.

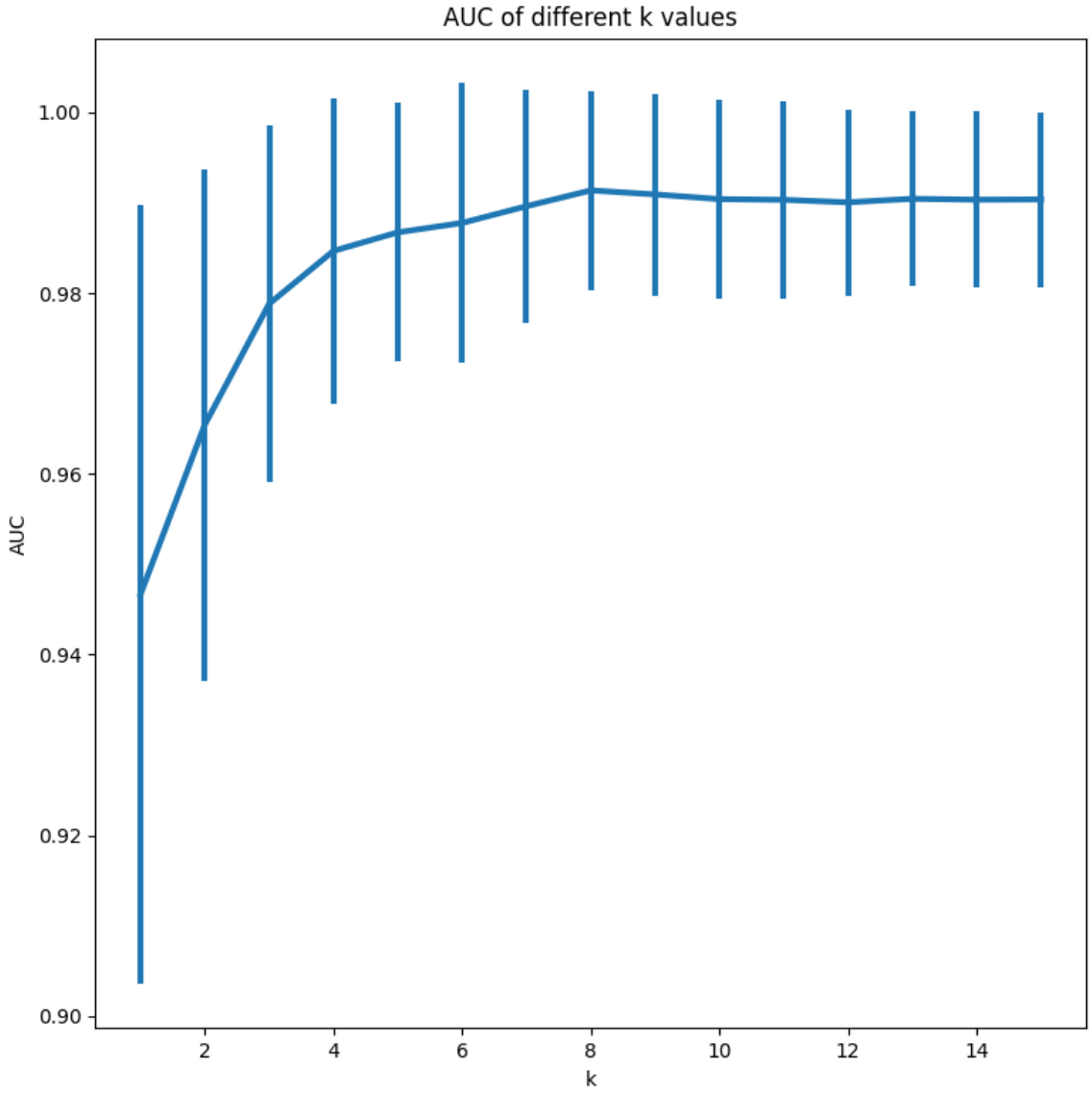


For getting the best polynomial to use, we can see that from 2 onwards, they are all roughly the same with only slight fluctuations in there errors, so we would obviously want to use a polynomial of 2 as that will reduce the likelihood of over-fitting.



For C values we see something similar, after a value of 5, the score stabilises, again, to reduce over-fitting through parameters that are too large and because there isn't a significant enough reduction in the error, I would choose to use a c value of 5.

(b) For the kNN model, I decided to use a uniform weighting system for simplicity, and because when I ran the model with a uniform approach, the results were already extremely good. To find the best value for k, I tested ourmodel with sklearns cross_val_score, where the score returned is the aoc of an roc curve, so the closer to one the score is, the better our classifier is at correctly determining the label.



From looking at this graph, we can that our scores start to stabilise once we reach k = 8, and we also appear to peak there too, because of this, I would choose k = 8 as my k.

(c) To get the confusion matrices, I used the test data that I had separated at the beginning, 20% of the overall data, to see how our model would do against completely unseen data. The confusion matrices I got were:

Logistic Regression:

	PN	PP
TN	192	22
TP	20	81

kNN:

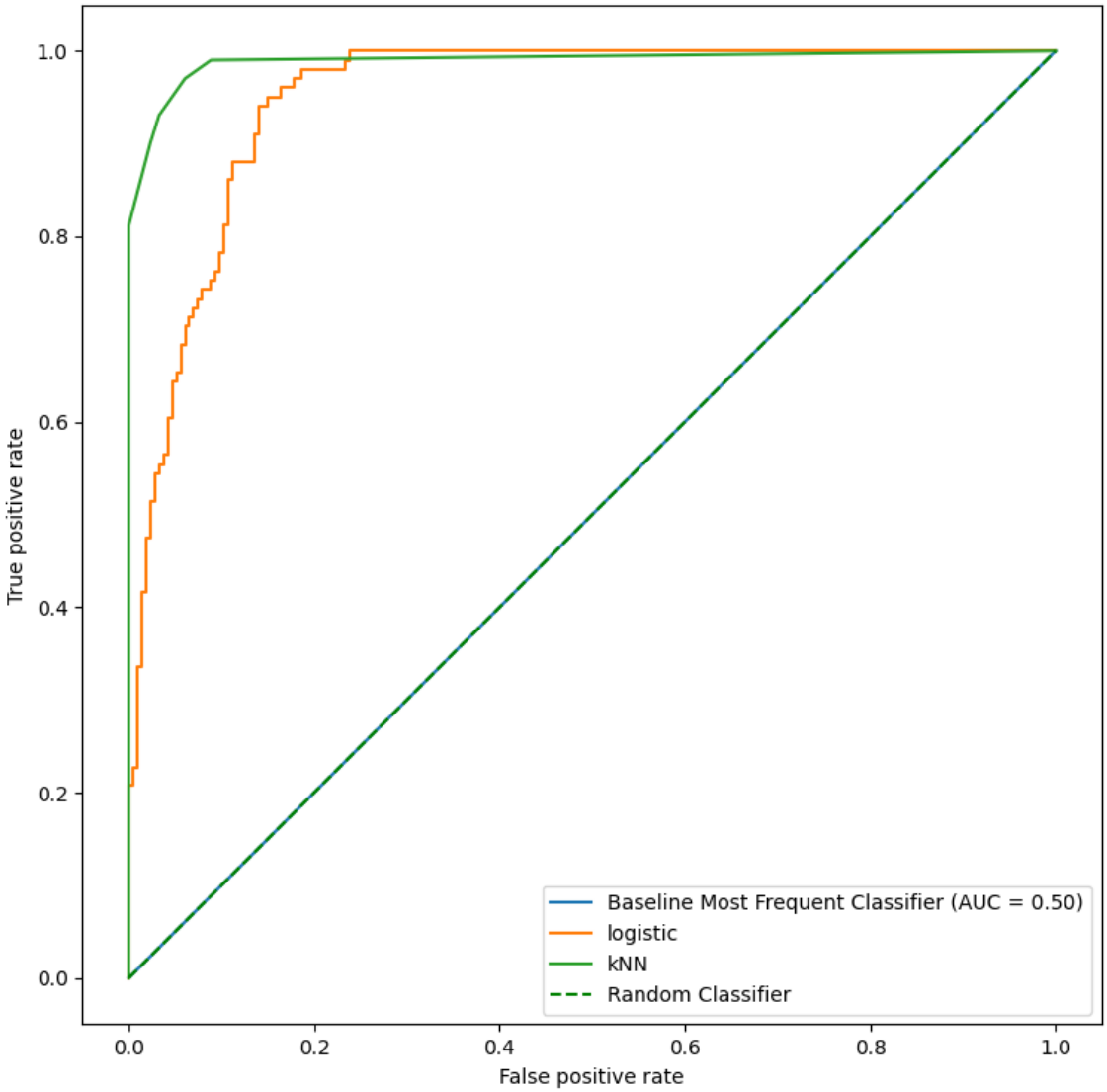
	PN	PP
TN	207	7
TP	7	94

Baseline most frequent classifier:

	PN	PP
TN	214	0
TP	101	0

Interpreting this data we can come up with some clear observations, that our baseline is only correct about two thirds of the time, which while quite good for a baseline predictor, is no match for our other models, and from these we can see that our kNN model performed better than our logistic regression, and while in terms of actual numbers, there is only a small difference, our false positives from logistic regression are almost 3 times that of our kNN model, and our false negatives are about the same as well. With this, we can say from our trained models, a kNN model would be best suited to our problem.

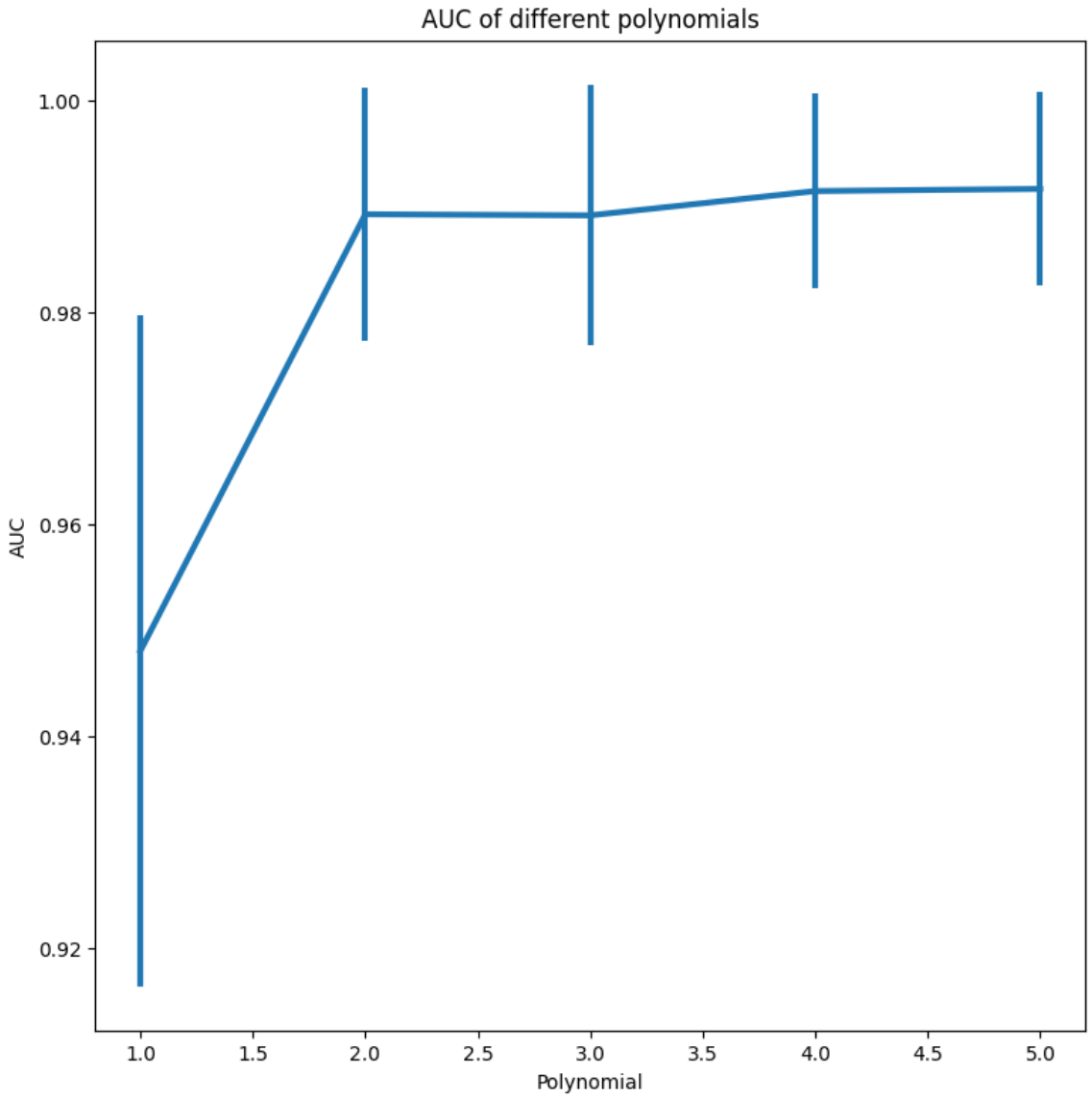
(d)



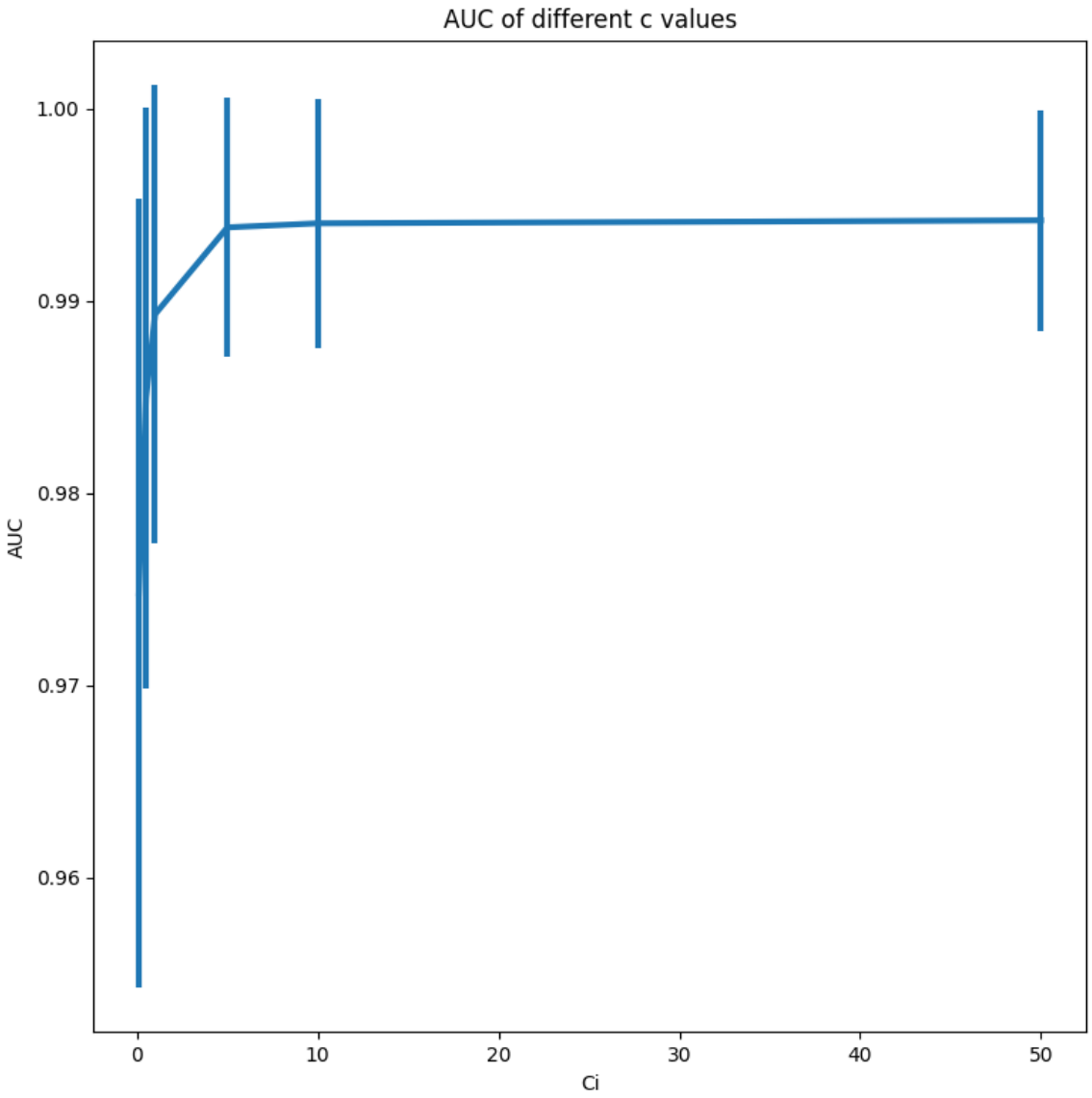
We can see from this graph that our baseline estimate performs exactly the same as a random classifier, and so is essentially useless, as random chance is a better predictor than it, but we can also see that kNN clearly performs better than logistic regression, as it gets closer to the top left corner.

(e) From (c) and (d) we can see that our baseline classifiers are both vastly outmatched by our trained models, with the most frequent classifier getting only two thirds of the labels correct, and having the rest be false negatives. When comparing the two models that we built, we can see that the kNN model is clearly better than the logistic regression model, from what has been shown before, the difference is not massive, but definitely noticeable, and on the scale of our data set it could certainly be described as importantly large a difference.

(ii) (a)

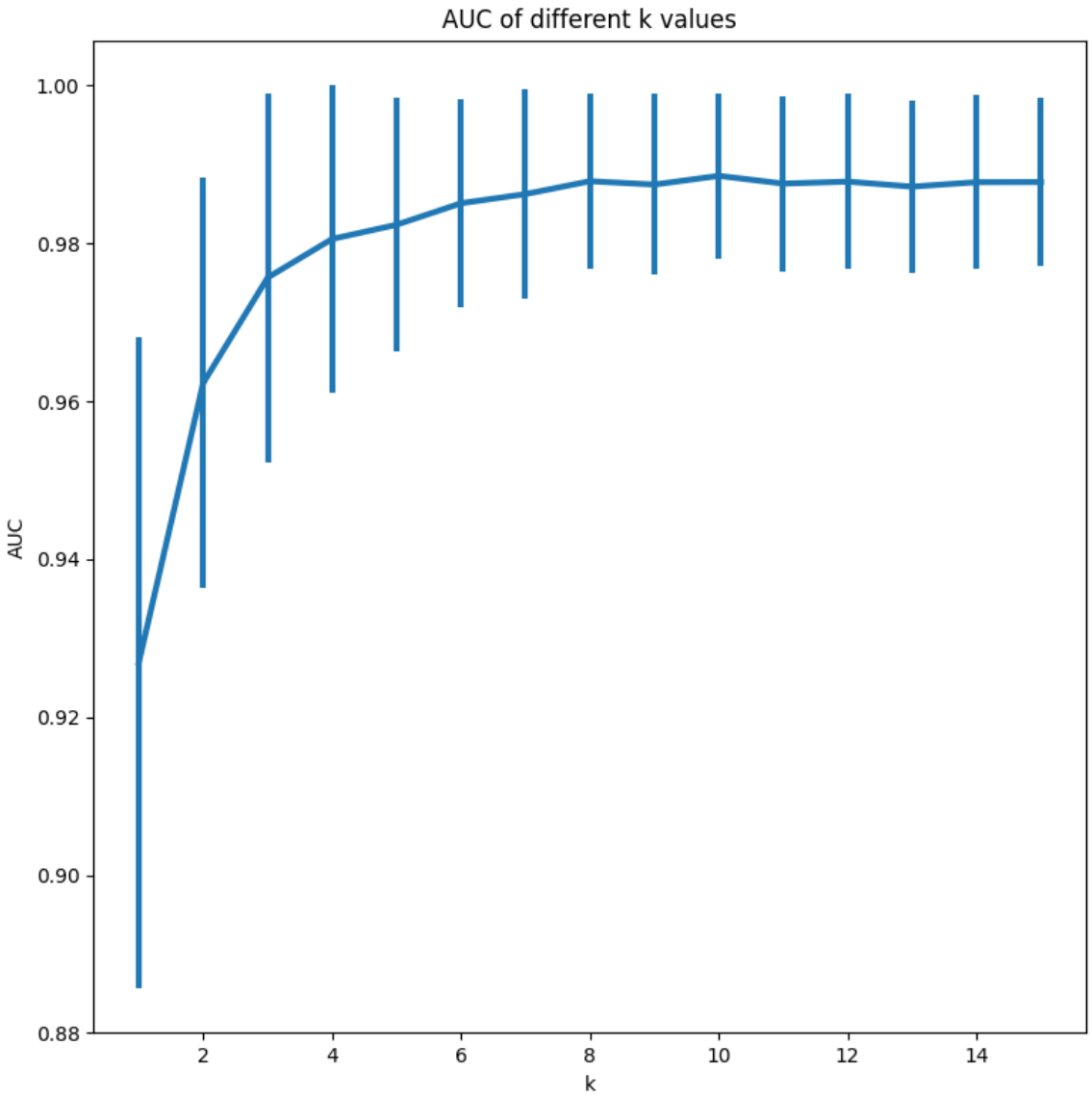


We get almost the same scenario to the first question, where the scores are essential equal past polynomial of 2 with errors that are just about equal as well, leaving us with the obvious choice of using the polynomial of 2 to reduce the complexity of our model, thus preventing over-fitting, at least in comparison to the higher polynomials.



Again, we get almost the same scenario as last time, scores stabilising at a C of 5, with little to no variation between scores and errors past it, so we want to the value of 5, as this will be the best for regularising our model and preventing over-fitting.

(b)



From the data, we can see that we again get a lot of values with roughly the same score and error past k=8, so I would choose that.

(c) The confusion matrices I got this time were:

Logistic Regression:

	PN	PP
TN	170	21
TP	27	97

kNN:

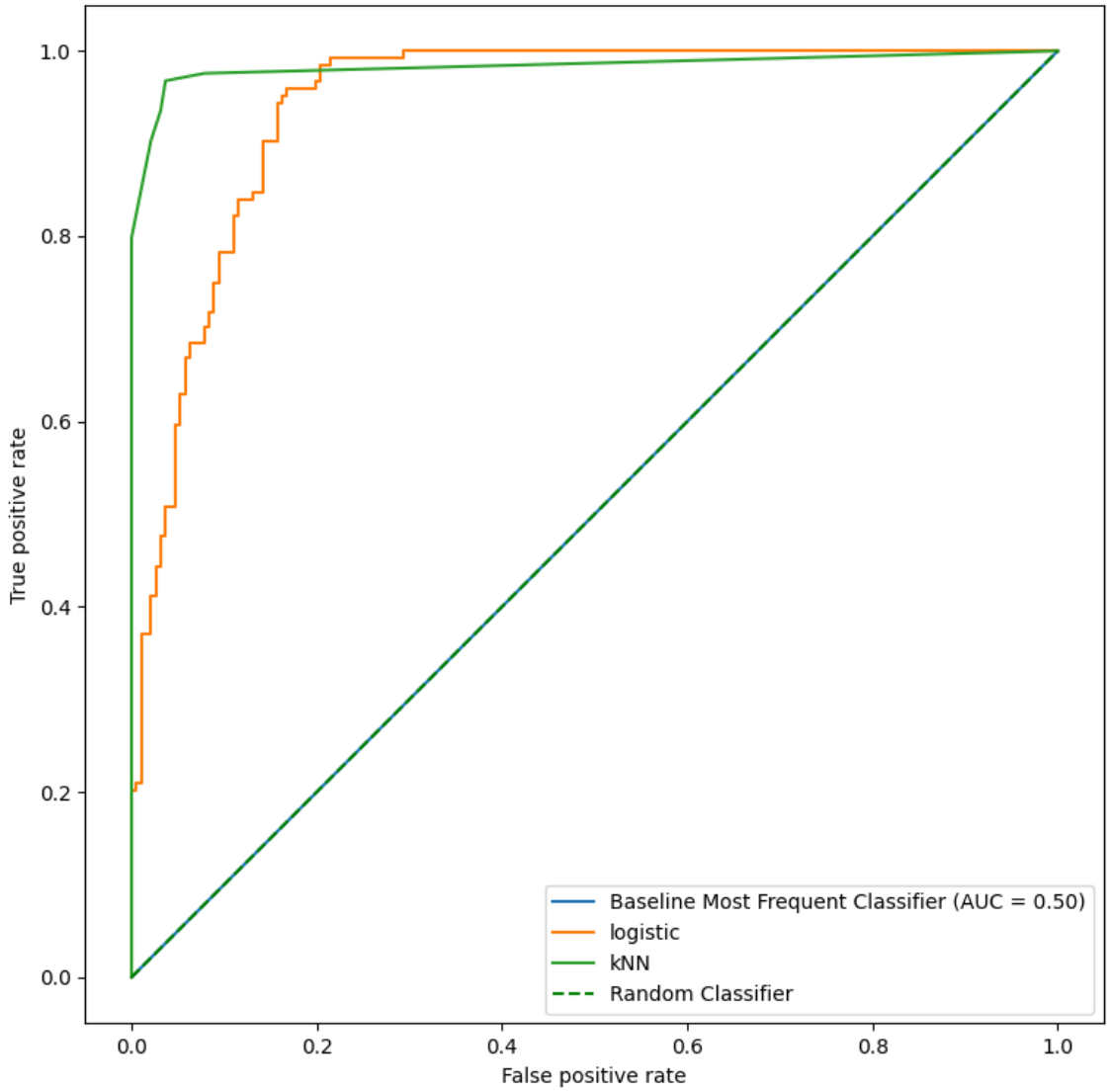
	PN	PP
TN	185	6
TP	8	116

Baseline most frequent classifier:

	PN	PP
TN	191	0
TP	124	0

We again get similar results, with our logistic regression model performing noticeably worse than our kNN model, and our baseline model being considerably worse than both.

(d)



We again get essentially the same graph as before.

(e) We again get practically the same results as before and so we could say that the same conclusion applies, we could also perhaps speculate that the two data sets came from the same data source, due to the very similar results, and the differences being so negligible for the size of the data sets that we were using.

Appendix:

```
# id: # id:14--14-14-0

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import PolynomialFeatures

dOne = []
dTwo = []

def extractData():
dataOne = {'X1': [], 'X2': [], 'Y': []}
dataTwo = {'X1': [], 'X2': [], 'Y': []}
data = dataOne
with open("./data.csv", 'r') as f:
dataset = 0
for line in f:
if '#' in line and dataset == 0:
dataset = 1
elif '#' in line and dataset == 1:
dataset = 2
data = dataTwo
else:
words = line.split(sep=',')
data['X1'].append(float(words[0]))
data['X2'].append(float(words[1]))
data['Y'].append(float(words[2].replace('\n','')))
return pd.DataFrame(dataOne), pd.DataFrame(dataTwo)

def choosePoly(X, Y, c):
model = LogisticRegression(penalty="l2", C=c)
kf = KFold(n_splits=5)
accuracy, mean_error, std_error = [], [], []
featureLength = range(1, 6)
```

```
for x in featureLength:
trans = PolynomialFeatures(degree=x)
data = trans.fit_transform(X)
acc = []
scores = []
for train, test in kf.split(data):
model.fit(data[train], Y[train])
ypred = model.predict(data[test])
acc.append(accuracy_score(Y[test], ypred))
score = cross_val_score(model, data[test], Y[test], cv=5, scoring='roc_auc')
scores.append(score)
accuracy.append(max(acc))
mean_error.append(np.array(scores).mean())
std_error.append(np.array(scores).std())

plt.errorbar(featureLength, mean_error, yerr=std_error, linewidth=3)
plt.xlabel('Polynomial'); plt.ylabel('AUC')
plt.title('AUC of different polynomials')
plt.show()

# After looking through the data, I have decided to use a polynomial of degree
2
return 2
def chooseC(X, Y, Ci_range, poly):
trans = PolynomialFeatures(degree=poly)
data = trans.fit_transform(X)
kf = KFold(n_splits=5)
accuracy, mean_error, std_error = [], [], []
for Ci in Ci_range:
acc = []
scores = []
model = LogisticRegression(penalty='l2', C=Ci)
for train, test in kf.split(data):
model.fit(data[train], Y[train])
ypred = model.predict(data[test])
acc.append(accuracy_score(Y[test], ypred))
score = cross_val_score(model, data[test], Y[test], scoring='roc_auc')
scores.append(score)
accuracy.append(max(acc))
mean_error.append(np.array(scores).mean())
std_error.append(np.array(scores).std())

plt.errorbar(Ci_range, mean_error, yerr=std_error, linewidth=3)
plt.xlabel('Ci'); plt.ylabel('AUC')
plt.title('AUC of different c values')
plt.show()
```

```
# By analysing the results, I've deemed 8 to be the best c value, returning it
like this so its a little more
# explicit as to how I get the value
return 8

def a(X, Y):
Ci_range = [0.1, 0.5, 1, 5, 10, 50]
c = Ci_range[2]
poly = choosePoly(X, Y, c)

c = chooseC(X, Y, Ci_range, poly)

return LogisticRegression(penalty='l2', C=c)
def b(X, Y):
kNeighbors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
kf = KFold(n_splits=5)
accuracy, mean_error, std_error = [], [], []
for k in kNeighbors:
acc, scores = [], []
for train, test in kf.split(X):
model = KNeighborsClassifier(n_neighbors=k, weights='uniform').fit(X[train],
Y[train])
ypred = model.predict(X[test])
acc.append(accuracy_score(Y[test], ypred))
score = cross_val_score(model, X[test], Y[test], scoring='roc_auc')
scores.append(score)
accuracy.append(max(acc))
mean_error.append(np.array(scores).mean())
std_error.append(np.array(scores).std())

plt.errorbar(kNeighbors, mean_error, yerr=std_error, linewidth=3)
plt.xlabel('k'); plt.ylabel('AUC')
plt.title('AUC of different k values')
plt.show()

return KNeighborsClassifier(n_neighbors=8, weights='uniform')

def c(X, Y, Xtest, Ytest, logistic, kNN, baseline):
# here we're gonna want to use the data that we preserved
model = logistic.fit(X, Y)
ypred = model.predict(Xtest)
cnf_matrix = confusion_matrix(Ytest, ypred)
print(cnf_matrix)

model = kNN.fit(X, Y)
ypred = model.predict(Xtest)
cnf_matrix = confusion_matrix(Ytest, ypred)
```

```
print(cnf_matrix)

model = baseline.fit(X, Y)
ypred = model.predict(Xtest)
cnf_matrix = confusion_matrix(Ytest, ypred)
print(cnf_matrix)

def d(X, Y, Xtest, Ytest, logistic, kNN, baseline):
model = baseline.fit(X, Y)
ypred = model.predict(Xtest)
RocCurveDisplay.from_predictions(Ytest, ypred, name='Baseline Most Frequent
Classifier')

model = logistic.fit(X, Y)
fpr, tpr, _ = roc_curve(Ytest, model.decision_function(Xtest))
plt.plot(fpr, tpr, label='logistic')

model = kNN.fit(X, Y)
y_scores = model.predict_proba(Xtest)
fpr, tpr, _ = roc_curve(Ytest, y_scores[:, 1])
plt.plot(fpr, tpr, label='kNN')

plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0,1], [0,1], color='green', linestyle='--', label='Random
Classifier')
plt.legend()
plt.show()

def part1(part=1):
if part == 1:
print("(i)\n")
X1 = dOne.iloc[:, 0]
X2 = dOne.iloc[:, 1]
X = np.column_stack((X1, X2))
Y = dOne.iloc[:, 2]
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.2)
X1 = np.array([x[0] for x in Xtrain])
X2 = np.array([x[1] for x in Xtrain])
X = np.column_stack((X1, X2))
Y = np.array(Ytrain)
else:
print("(ii)\n")
X1 = dOne.iloc[:, 0]
X2 = dOne.iloc[:, 1]
X = np.column_stack((X1, X2))
Y = dOne.iloc[:, 2]
```

```
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.2)
X1 = np.array([x[0] for x in Xtrain])
X2 = np.array([x[1] for x in Xtrain])
X = np.column_stack((X1, X2))
Y = np.array(Ytrain)

logistic = a(X, Y)
kNN = b(X, Y)
baseline = DummyClassifier(strategy='most_frequent')
c(X, Y, Xtest, Ytest, logistic, kNN, baseline)
d(X, Y, Xtest, Ytest, logistic, kNN, baseline)
# (e) is typing

dOne, dTwo = extractData()
part1()
part1(2)
```