Ritika Sharma sharmari@tcd.ie
MSc Computer Science – Intelligent Systems 21334470

WEEK 2 ASSIGNMENT

Dataset Used - # id:13-26--13-1

i) a) The first part requires to apply logistic regression to the dataset with different polynomial degrees and values of C and determine which combination gives the best result.

- First apply different values for degree of polynomial features. The range applied here is [1,6].

```
for q in range(1,6):
    poly = PolynomialFeatures(degree=q)
    X_poly = poly.fit_transform(X)
```

- Since we want to check the contribution of each feature in the model we will apply L2 penalty to the Logistic regression.

```
Ci_range = [0.01, 0.1, 1, 5, 10, 25, 50, 100]
model_logi = LogisticRegression(penalty='l2',C=Ci, verbose=0,max_iter=1500000)
```

- To split the dataset into training and testing, K-Fold Cross Validation technique is used. It will split the data into 80% training and 20% testing and repeat this process for 5 folds. Here, instead of applying the loop for k-fold, we use the function *cross_val_score* to determine the scores for cross validation. The scoring method used is f1 score, a combination of precision and recall.

```
scores = cross_val_score(model_logi, X_poly, y, cv=5, scoring='f1')
mean_error.append(np.array(scores).mean())
std_error.append(np.array(scores).std())
```

- To plot the error bar, we use the mean_error for all the values of C for corresponding degree. Below are the graphs for different degrees of polynomial feature.

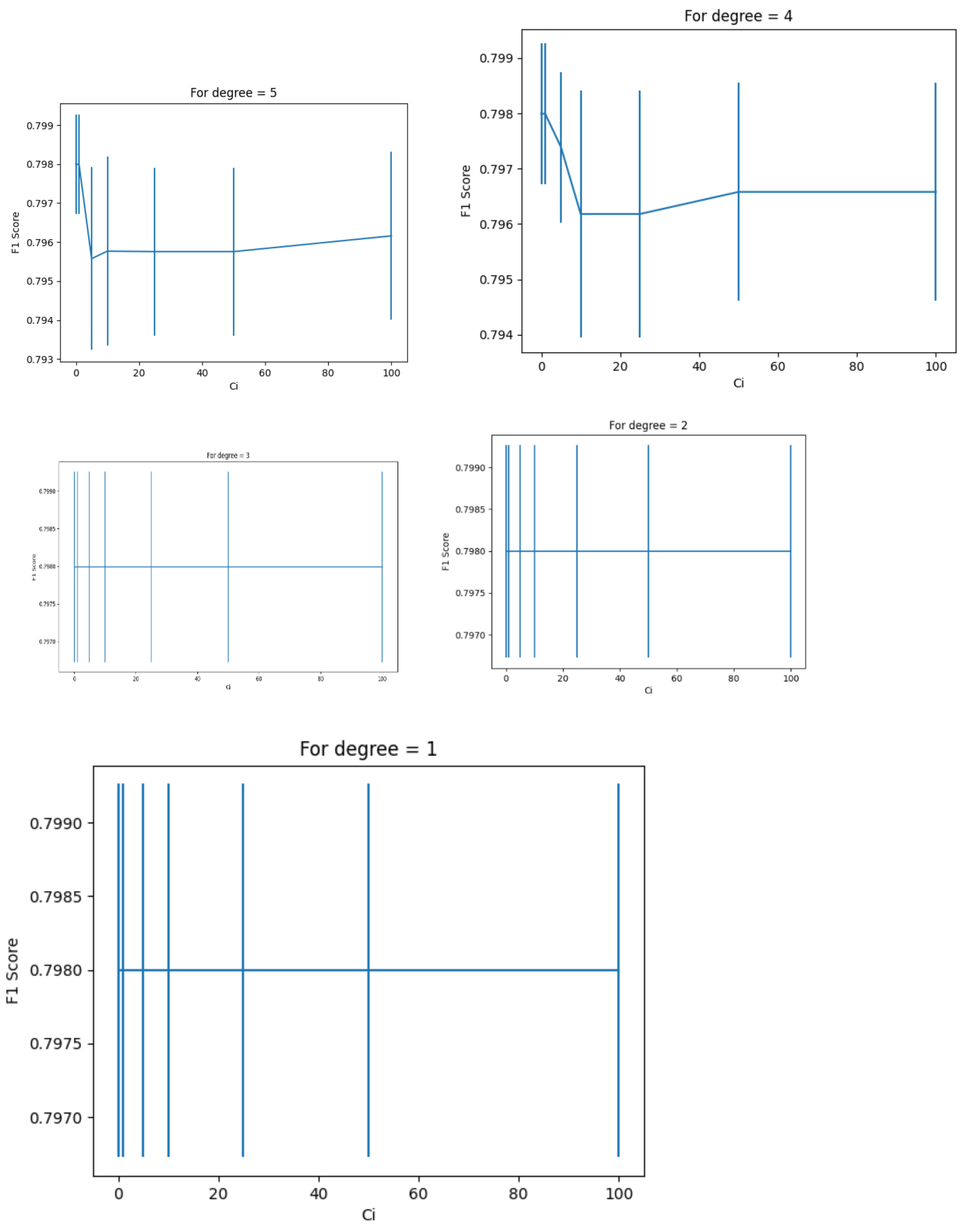


Fig 1. Logistic Regression for different values of C and degree

- From the above graphs we can observe that the data is noisy and for all the degrees we have almost same result. Even across values of C, the score is almost same giving us a straight line, except for degrees 4 and 5.
- Adding features to the already noisy data would make it even worse, so the wiser choice would be to go for lower degree values. From the graph we can see that the F1 score for degree 1 and C=1 is decent and almost the same for higher values as well. So we will choose degree = 1 and C=1 to train our logistic regression model.

i) b) In this part, we are required to train the model with kNN classifier with different values of k.

- To split the data, we will again be using K-Fold cross validation and this time we wont be introducing polynomial features, as kNN has no trouble capturing nonlinear decision boundaries.
- We are taking different values of k, to determine the best number of neighbours. k_range=[1,12]
- We repeat the same procedure as part i)a), calculate the scores for different set of training data using *cross_val_score* for different k.

```
knn = KNeighborsClassifier(n_neighbors=k)
scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')
mean_error.append(np.array(scores).mean())
std_error.append(np.array(scores).std())
```

- To plot the error bar, store the values of mean and standard deviation in a list and plot it against k_range using *plt.errorbar.*

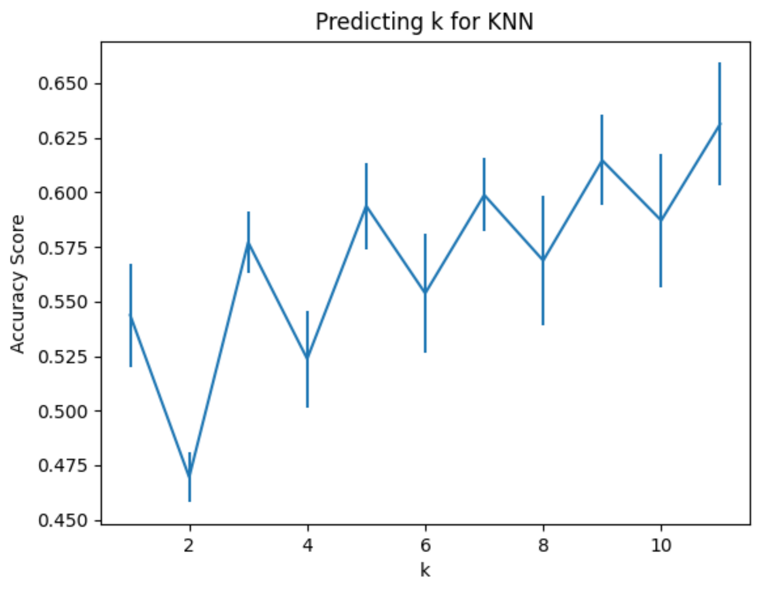


Fig 2. Accuracy Score for different values of k, kNN classifier

- In the above graph you can see that for k=7, we are getting decent accuracy score and also the std error is very less as compared to other values. So, it is safe to assume, k=7 can be a better choice than other values of k.

i) c) In the third part we are asked to calculate the confusion matrix for the above models.

- Confusion matrix is a measurement of the performance of a ML model, specially used for classification problems. It is a table of predicted and actual positive negative values. It is used for measuring precision, recall and roc curve.
- We calculate the confusion matrix for each type of model and baseline classifiers for kNN and Logistic regression model and compare their values.

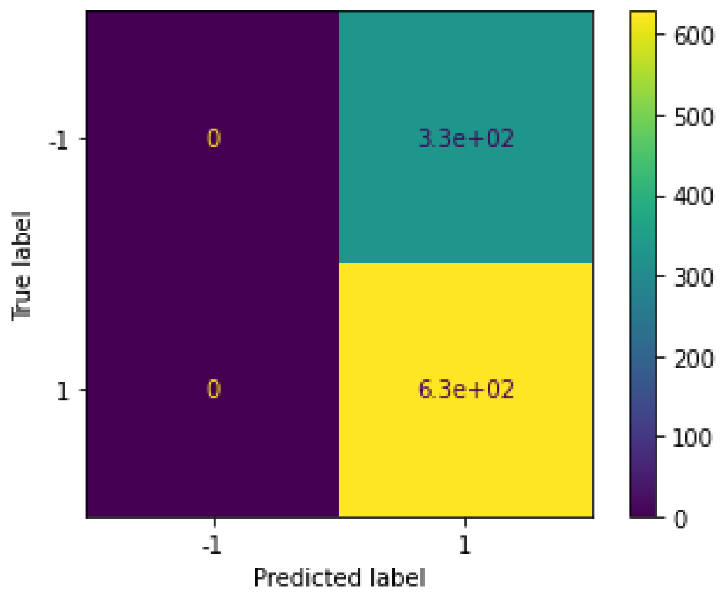


Fig 3 – Confusion matrix for logistic regression, C=1, degree=1

- From fig 3, we can observe that our logistic regression model doesn't predict any positive values. The predictions made by the model are same as that of baseline classifier and it doesn't outshine the baseline. So, this model shows no significant prediction.
- Confusion matrix for kNN with k=9 :–

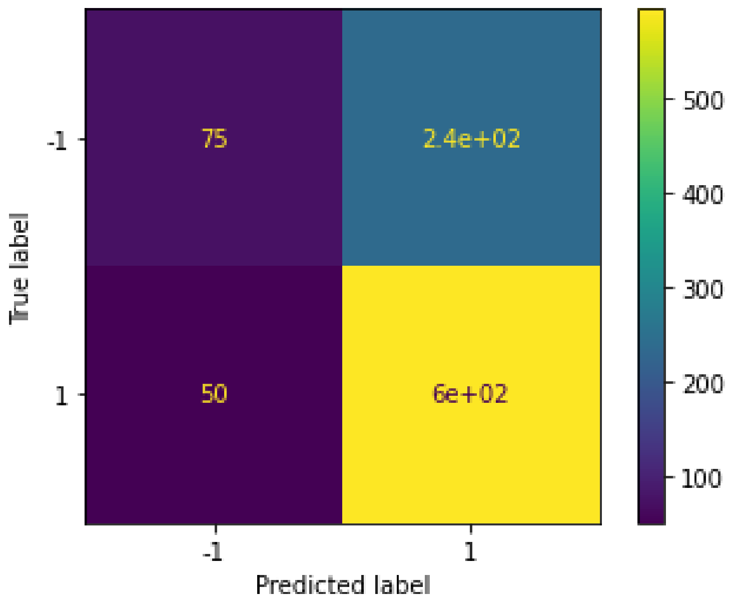


Fig 4 – Confusion matrix for kNN, k = 9

- The confusion matrix for kNN model shows that the model predicts true positives as well unlike logistic regression.
- The performance of kNN is better than the baseline classifiers that we created.

i) d) This part requires to plot roc curve involving all the models, logistic regression, kNN and the baseline classifiers.

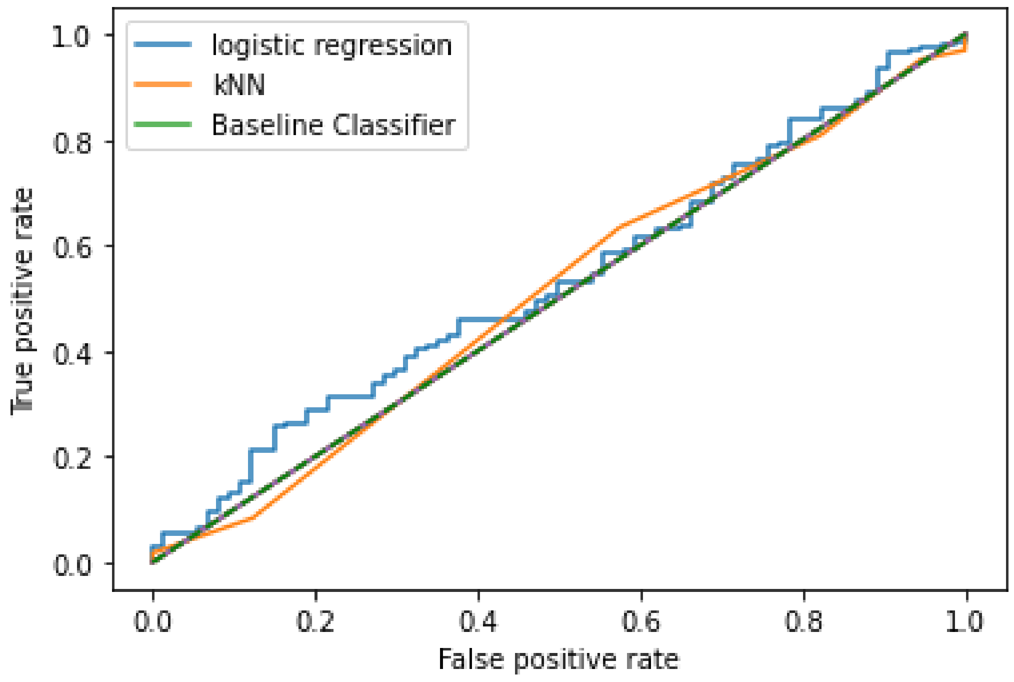


Fig 5 – ROC curve

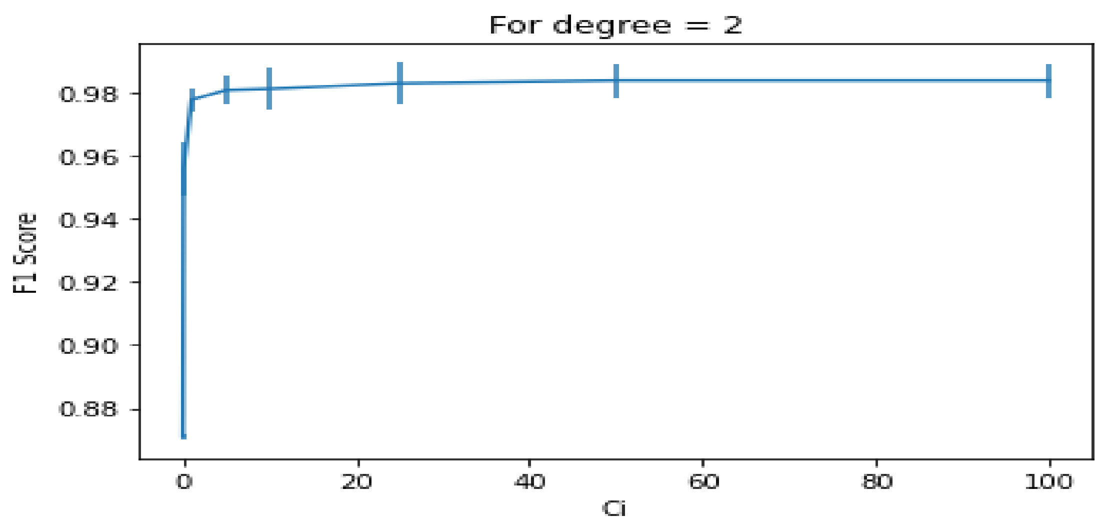
- This fig contains roc curves for all the models.
- The blue line contains the logistic regression classifier and we can clearly see it gives the same predictions as our baseline model.
- The orange line depicts kNN model and it is almost the same as the baseline classifier and doesn't give better results as well.

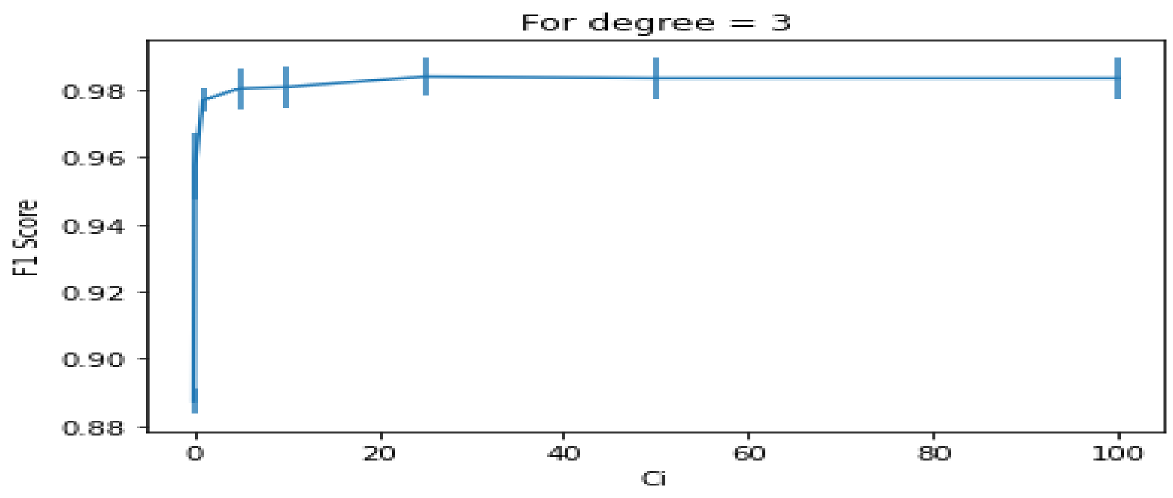
i) e) This part requires to describe which model is better.

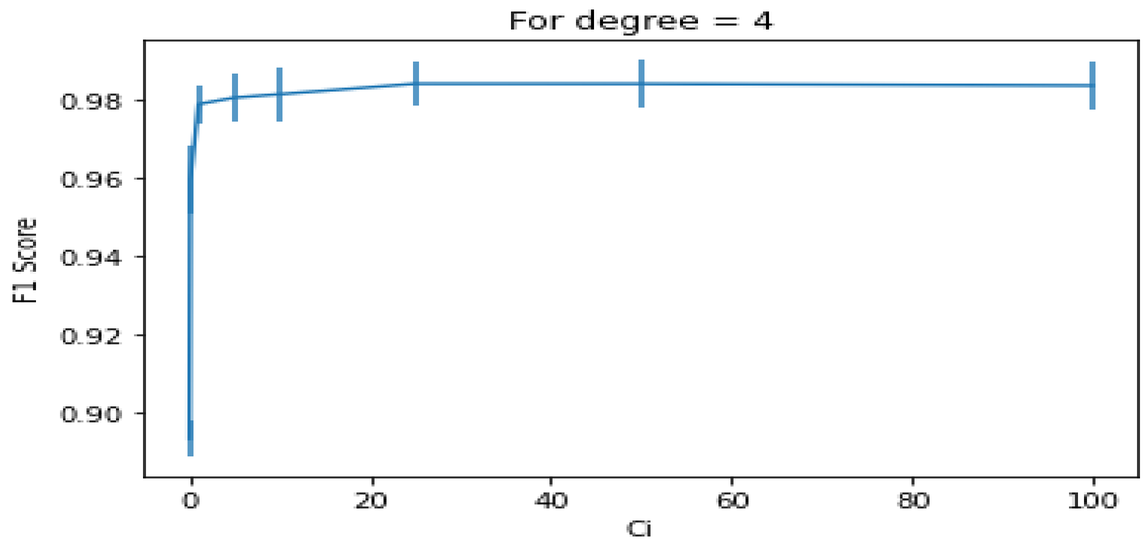
- With the help of fig 4, it can be easily concluded that kNN model is better than Logistic regression as the kNN model predicts some true positives as opposed to logistic regression which doesn't predict any positive values
- From fig 5, it can be concluded that even though kNN is better than logistic, it clearly is not the best solution for this dataset as its predictions are not better than the baseline classifiers.
- We would not recommend either models for solving real world problems as their performance is not upto the mark.

ii) For the second part, we have to repeat the above steps for a different dataset. This dataset is not as noisy as the previous one.

a) Create a Logistic regression model with different values of C and degree of polynomial features. For that we used the same code as before.


For degree = 2
F1 Score
0.98
0.96
0.94
0.92
0.90
0.88
0
20
40
60
80
100
Ci


For degree = 3
F1 Score
0.98
0.96
0.94
0.92
0.90
0.88
0
20
40
60
80
100
Ci


For degree = 4
F1 Score
0.98
0.96
0.94
0.92
0.90
0
20
40
60
80
100
Ci

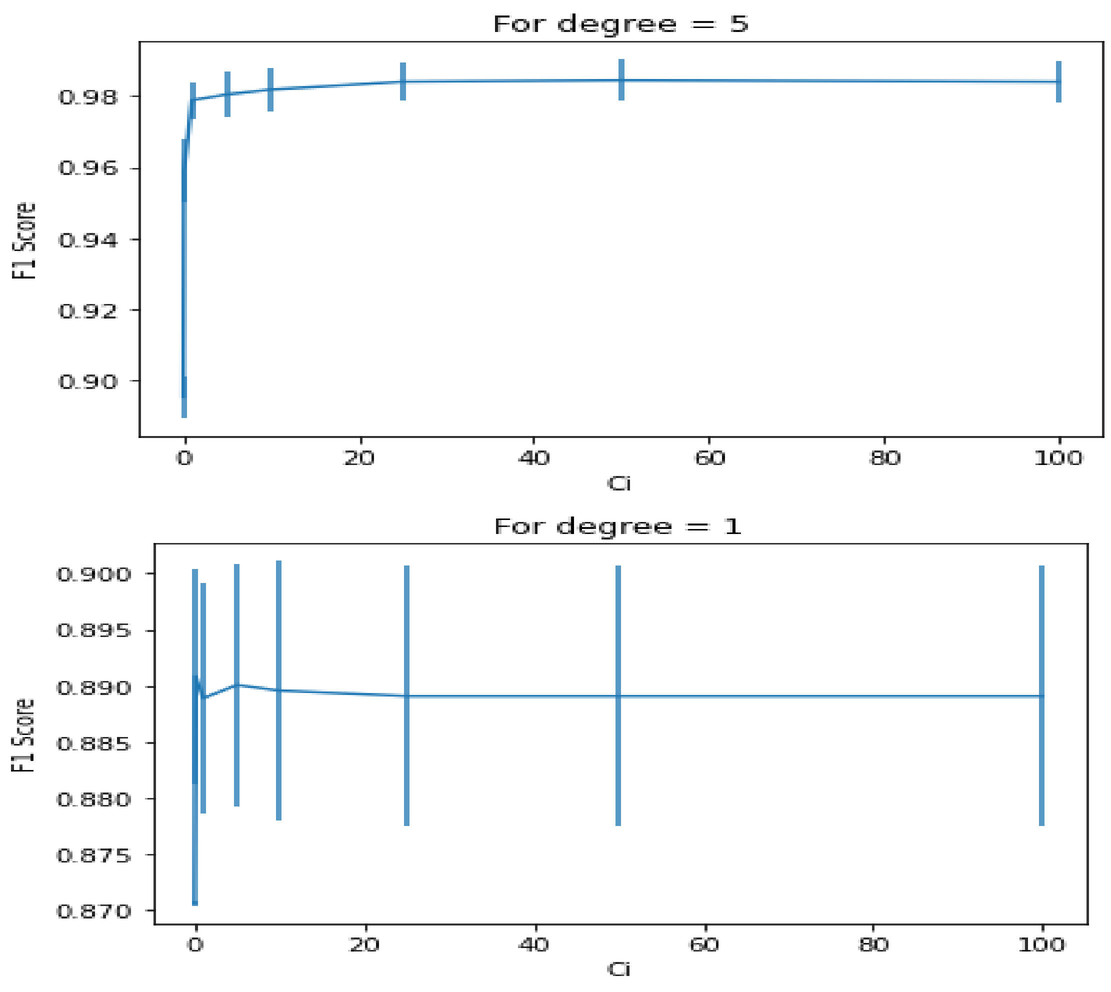


Fig 6 – Logistic regression, F1 score for different values of C

- For degree 1, the std error is high.
- For degree 2, we see that for C>=50 the F1 score is almost consistent and the f1 score is good.
- After degree 2, the plot remains consistent only, so we can conclude that degree=2 and C=50 gives us an optimal result for dataset 2.

b) Create a kNN model for the dataset(no polynomial feature) for different values of k, just like we did in the part i)

- The graph below shows the accuracy score for different values of k.

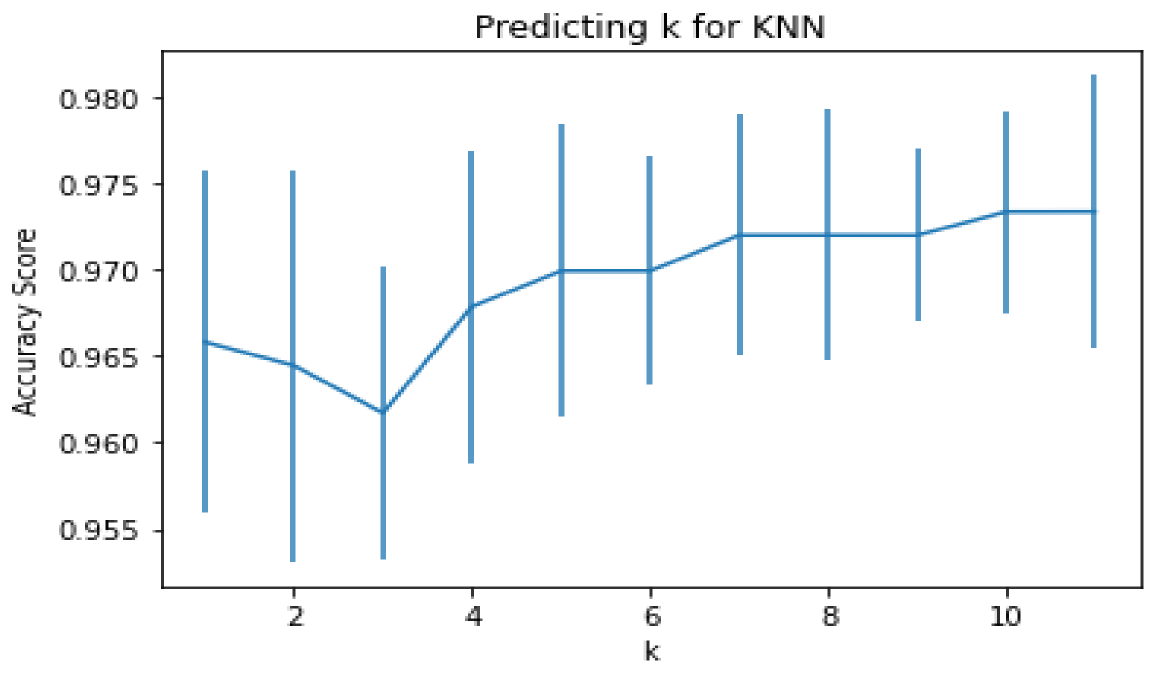


Fig 7. Accuracy score for different values of k, kNN model

- From the above graph we can observe that the best value of k is 9 as the std error is the minimum and we have a good value of accuracy score at that point.

c) Find confusion matrix for all the models.

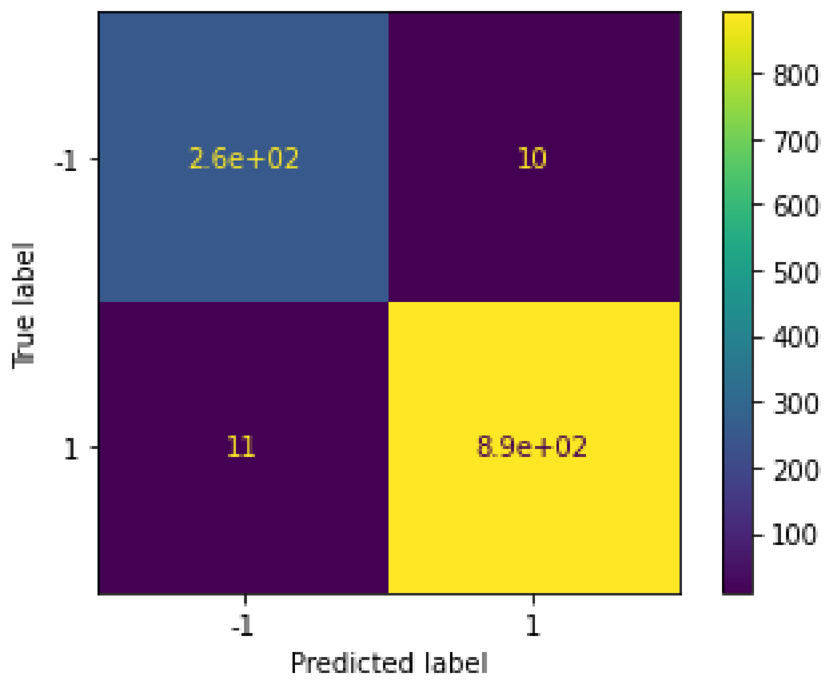


Fig 8 Confusion matrix for logistic regression, C=50, degree=2

- In the above confusion matrix plot we can see that the model does predict true positives and true negatives, unlike the model we had in part i).

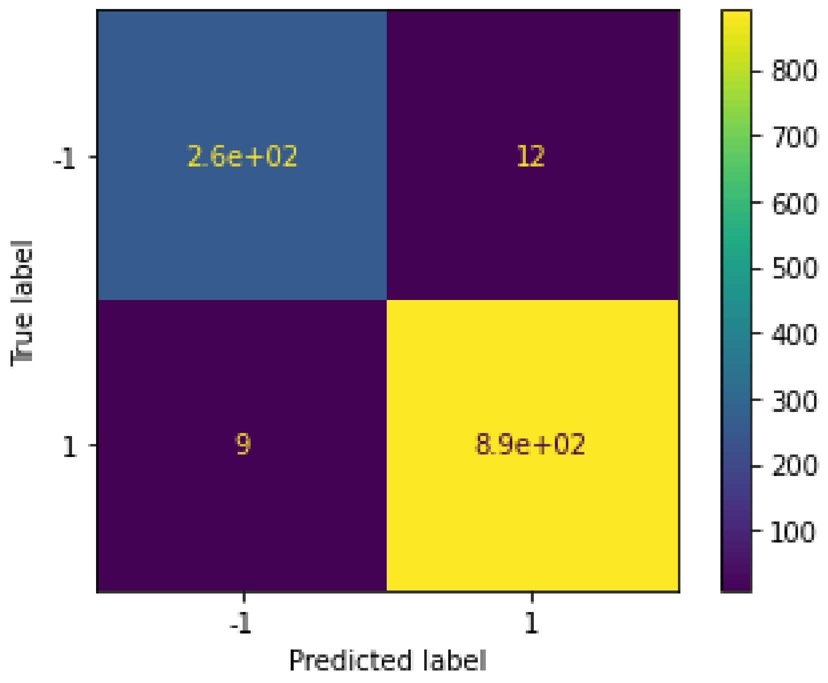


Fig 9 – Confusion matrix for kNN, k = 9

- For kNN model, we can observe the results are better than part i) and the model predicts true negatives and true positives.

d) Plot ROC curve for all models.

- In the Below graph we can see that both the models, logistic regression and kNN perform as expected and are optimal.
- The baseline classifier gives us an idea of what should the minimum values be.
- The models are close to true positive rate 1 and that is a sign of good model.

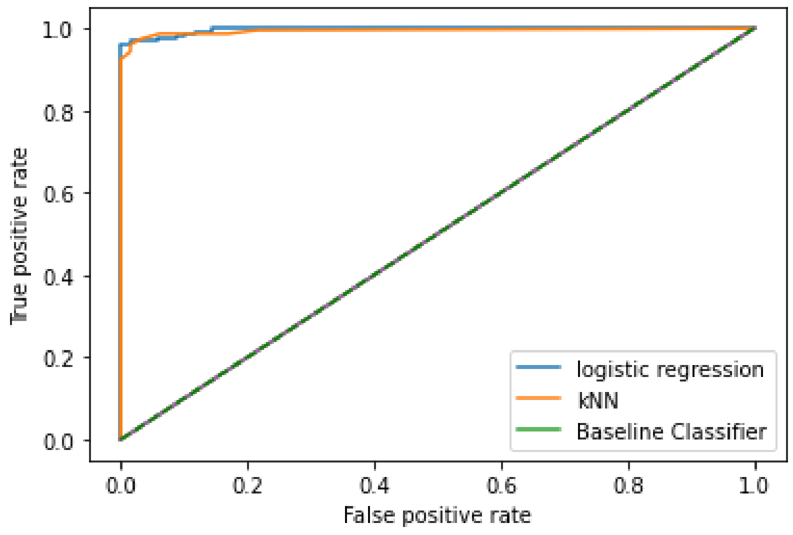


Fig 10 – ROC curve for all models.

e) Determine which model is better

- For this dataset, both models are good.

- Both perform the same way in ROC curve.
- Both their ROC curves are better than the baseline random classifier.
- Both can be used to predict this dataset.

Appendix

Part i)

```
# dataset id
# id:13-26--13-1

# import libraries
from matplotlib.pyplot import subplot
import pandas as pd
import numpy as np


# get data
df = pd.read_csv('week4_dataset1.csv')
# print(df.head())
X1=df.iloc[ : , 0 ]
X2 = df.iloc[:, 1]
y=df.iloc[ : , 2]
X = np.column_stack((X1,X2))


# i) a)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

for q in range(1,6):
    poly = PolynomialFeatures(degree=q)
    X_poly = poly.fit_transform(X)

    mean_error=[]
    std_error=[]
    Ci_range = [0.01, 0.1, 1, 5, 10, 25, 50, 100]

    for Ci in Ci_range:
        model_logi = LogisticRegression(penalty='l2', C=Ci, verbose=0,
max_iter=1500000)
        scores = cross_val_score(model_logi, X_poly, y, cv=5, scoring='f1')
        mean_error.append(np.array(scores).mean())
        std_error.append(np.array(scores).std())

    import matplotlib.pyplot as plt
    plt.errorbar(Ci_range,mean_error,yerr=std_error)
    plt.xlabel('Ci'); plt.ylabel('F1 Score')
    plt.title('For degree = '+str(q))
    plt.show()


# i) b)
from sklearn.neighbors import KNeighborsClassifier

# choose k between 1 to 12
k_scores = []
mean_error=[]
std_error=[]
k_range = [1,2,3,4,5,6,7,8,9,10,11]

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())

import matplotlib.pyplot as plt
plt.errorbar(k_range,mean_error,yerr=std_error)
plt.xlabel('k'); plt.ylabel('Accuracy Score')
plt.title('Predicting k for KNN')
plt.show()


# i) c)
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, plot_confusion_matrix

# confusion matrix for Logistic regression
poly = PolynomialFeatures(degree=1)
X_poly = poly.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.20)

model = LogisticRegression(penalty='l2', C=1)
model.fit(X_train, y_train)
y_pred_logi = model.predict(X_test)
y_pred_logi_prob = model.predict_proba(X_test)
plot_confusion_matrix(model, X_train, y_train)
print(confusion_matrix(y_test, y_pred_logi))

# baseline classifiers for Logistic Regression
# most frequent
from sklearn.dummy import DummyClassifier
dummy = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
y_dummy = dummy.predict(X_test)
```

```
y_dummy_prob1 = dummy.predict_proba(X_test)
print(confusion_matrix(y_test, y_dummy))
# random
dummy = DummyClassifier(strategy='uniform', random_state=3).fit(X_train,
y_train)
y_dummy = dummy.predict(X_test)
y_dummy_prob2 = dummy.predict_proba(X_test)
print(confusion_matrix(y_test, y_dummy))


# confusion matrix for kNN
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.20)
knn_9 = KNeighborsClassifier(n_neighbors=5, weights='uniform')
knn_9.fit(Xtrain, ytrain)
y_pred_knn = knn_9.predict(Xtest)
y_pred_knn_prob = knn_9.predict_proba(Xtest)
plot_confusion_matrix(knn_9, Xtrain, ytrain)
print(confusion_matrix(ytest, y_pred_knn))

# baseline classifiers for kNN
# most frequent
from sklearn.dummy import DummyClassifier
dummy = DummyClassifier(strategy='most_frequent').fit(Xtrain, ytrain)
y_dummy = dummy.predict(Xtest)
y_dummy_prob3 = dummy.predict_proba(Xtest)
print(confusion_matrix(ytest, y_dummy))
# random
dummy = DummyClassifier(strategy='uniform', random_state=3).fit(Xtrain,
ytrain)
y_dummy = dummy.predict(Xtest)
y_dummy_prob4 = dummy.predict_proba(Xtest)
print(confusion_matrix(ytest, y_dummy))


# i) d)
from sklearn.metrics import roc_curve
fpr, tpr, _ = roc_curve(y_test, y_pred_logi_prob[:,1])
plt.plot(fpr,tpr)
fpr, tpr, _ = roc_curve(ytest,y_pred_knn_prob[:,1])
plt.plot(fpr,tpr)
fpr, tpr, _ = roc_curve(ytest,y_dummy_prob4[:,1])
plt.plot(fpr,tpr)
fpr, tpr, _ = roc_curve(ytest,y_dummy_prob4[:,1])
plt.plot(fpr,tpr)
fpr, tpr, _ = roc_curve(y_test,y_dummy_prob2[:,1])
plt.plot(fpr,tpr)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='green',ls='--')
plt.show()
```

Part ii)

```
# dataset id
# id:13-26--13-1

# import libraries
from matplotlib.pyplot import subplot
import pandas as pd
import numpy as np


# get data
df = pd.read_csv('dataset2.csv')
# print(df.head())
X1=df.iloc[ : , 0 ]
X2 = df.iloc[:, 1]
y=df.iloc[ : , 2]
X = np.column_stack((X1,X2))


# ii) a)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

for q in range(1,6):
    poly = PolynomialFeatures(degree=q)
    X_poly = poly.fit_transform(X)

    mean_error=[]
    std_error=[]
    Ci_range = [0.01, 0.1, 1, 5, 10, 25, 50, 100]

    for Ci in Ci_range:
        model_logi = LogisticRegression(penalty='l2', C=Ci, verbose=0,
max_iter=1500000)
        scores = cross_val_score(model_logi, X_poly, y, cv=5, scoring='f1')
        mean_error.append(np.array(scores).mean())
        std_error.append(np.array(scores).std())

    import matplotlib.pyplot as plt
    plt.errorbar(Ci_range,mean_error,yerr=std_error)
    plt.xlabel('Ci'); plt.ylabel('F1 Score')
    plt.title('For degree = '+str(q))
    plt.show()
```

```
# ii) b)
from sklearn.neighbors import KNeighborsClassifier

# choose k between 1 to 12
k_scores = []
mean_error=[]
std_error=[]
k_range = [1,2,3,4,5,6,7,8,9,10,11]

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())

import matplotlib.pyplot as plt
plt.errorbar(k_range,mean_error,yerr=std_error)
plt.xlabel('k'); plt.ylabel('Accuracy Score')
plt.title('Predicting k for KNN')
plt.show()


# ii) c)
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, plot_confusion_matrix

# confusion matrix for Logistic regression
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.20)

model = LogisticRegression(penalty='l2', C=50)
model.fit(X_train, y_train)
y_pred_logi = model.predict(X_test)
y_pred_logi_prob = model.predict_proba(X_test)
plot_confusion_matrix(model, X_train, y_train)
print(confusion_matrix(y_test, y_pred_logi))

# baseline classifiers for Logistic Regression
# most frequent
from sklearn.dummy import DummyClassifier
dummy = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
y_dummy = dummy.predict(X_test)
y_dummy_prob1 = dummy.predict_proba(X_test)
print(confusion_matrix(y_test, y_dummy))
# random
dummy = DummyClassifier(strategy='uniform', random_state=3).fit(X_train,
y_train)
y_dummy = dummy.predict(X_test)
```

```
y_dummy_prob2 = dummy.predict_proba(X_test)
print(confusion_matrix(y_test, y_dummy))


# confusion matrix for kNN
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.20)
knn_9 = KNeighborsClassifier(n_neighbors=9, weights='uniform')
knn_9.fit(Xtrain, ytrain)
y_pred_knn = knn_9.predict(Xtest)
y_pred_knn_prob = knn_9.predict_proba(Xtest)
plot_confusion_matrix(knn_9, Xtrain, ytrain)
print(confusion_matrix(ytest, y_pred_knn))

# baseline classifiers for kNN
# most frequent
from sklearn.dummy import DummyClassifier
dummy = DummyClassifier(strategy='most_frequent').fit(Xtrain, ytrain)
y_dummy = dummy.predict(Xtest)
y_dummy_prob3 = dummy.predict_proba(Xtest)
print(confusion_matrix(ytest, y_dummy))
# random
dummy = DummyClassifier(strategy='uniform', random_state=3).fit(Xtrain,
ytrain)
y_dummy = dummy.predict(Xtest)
y_dummy_prob4 = dummy.predict_proba(Xtest)
print(confusion_matrix(ytest, y_dummy))


# ii) d)
from sklearn.metrics import roc_curve
fpr, tpr, _ = roc_curve(y_test, y_pred_logi_prob[:,1])
plt.plot(fpr,tpr)
fpr, tpr, _ = roc_curve(ytest,y_pred_knn_prob[:,1])
plt.plot(fpr,tpr)
fpr, tpr, _ = roc_curve(ytest,y_dummy_prob4[:,1])
plt.plot(fpr,tpr)
fpr, tpr, _ = roc_curve(ytest,y_dummy_prob4[:,1])
plt.plot(fpr,tpr)
fpr, tpr, _ = roc_curve(y_test,y_dummy_prob2[:,1])
plt.plot(fpr,tpr)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='green',ls='--')
```