Week 4 Assignment

CS7CS4/CSU44061 Machine Learning

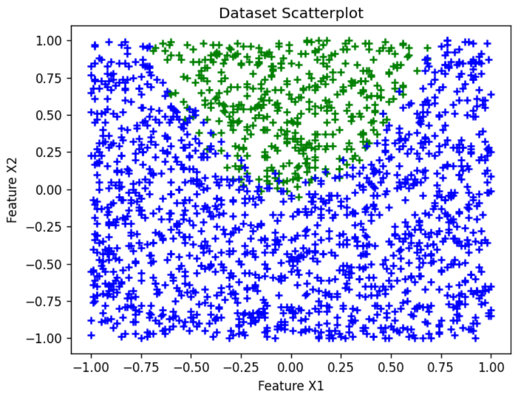
Cormac Madden – 18319983

Dataset ID: # id:6-12--6-0,

# id:6-6--6-0

i) a)

First, I plotted the dataset on a scatter plot to gain an understanding of the data.
I created a function to do this so it can be done for any new set of similarly structed data.

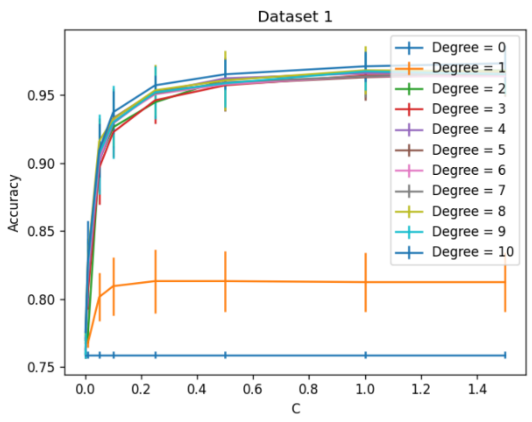


From observation the separation of the data between the two y features appears to follow a quadratic curve.

To augment the two features in the dataset, I created a function that iterates the degrees 0 through to 10 and used the Sklearn polynomial_features function to transform the values accordingly.

Then for each order of polynomial I created a logistic regression model, and used 7 weight values (C), ranging from 0.001 to 1.5, to adjust the penalty and find the appropriate cost function.

I decided to use “accuracy” as the measure of error for this plot. Accuracy is the ratio of correct predictions against total instances. It is flawed in many circumstances, but here it allows us to quickly see the effectiveness of each polynomial degree.



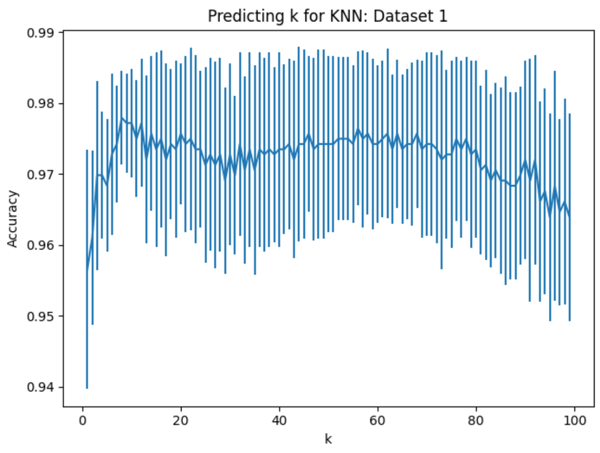
From the graph it is clear that any order higher than 1 fits the data well, but using the second order 2, would probably be most appropriate for this dataset to keep the model simple and avoid overfitting.

A weight value of C=1, seems like an appropriate value to give to the penalty of the cost function. It results in an accuracy of approximately 0.95 and keeps the model quite general.

b)

The following graph shoes a kNN classifier trained on the data. It plots accuracy against different values of k. K in k-NN represents the number of neighbours to each point considered when plotting the model. A higher k will tend to smooth out the function, while decreasing k will make the model track the training data points more closely, but potentially overfit the data. This is why a decline in accuracy can be seen at the end of the graph, as the model becomes more generalized.

I would recommend any value of k between 10 and 80. They all have a high accuracy, and most values have similar error. I would select a value of 60 as it has a slightly lower standard error. I would be weary of picking a k of 10 as it might be over-fitting the data, but it would be worth investigating.



c)

Below are the confusion matrices for each model. The C and k polynomial order values used in these models are those selected in section a and b. The input data is split into training data and testing data, with 25% of the data will be used for testing. Below each confusion matrix are the performance metrics for the model.

Logistic Regression Model

93 True Positives	7 False Positives
7 False Negatives	347 True Negatives

Accuracy = 0.969163

True Positive Rate (Recall) = 0.93

False Positive Rate (Specificity) = 0.02

Precision = 0.93

K-NN Model

91 True Positives	9 False Negatives
7 False Positives	347 True Negatives

Accuracy = 0.964758

True Positive Rate (Recall) = 0.91

False Positive Rate (Specificity) = 0.01977

Precision = 0.928571

Dummy Model

A baseline dummy model was implemented which predicts the majority every time.

In this case the majority is y = 1 (354 compared to 100).

0 True Positives	100 False Positives
0 False Negatives	354 True Negatives

Accuracy = 0.781182

True Positive Rate (Recall) = 0

False Positive Rate (Specificity) = 0.220264

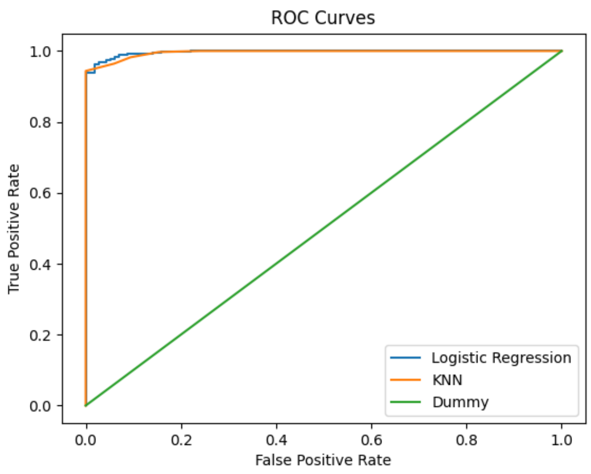
Precision = 0

The dummy model behaves as you would expect, and only predicts either false positives or true positives.

d)

The following graph contains the ROC curves for the trained models used in part c).
A ROC curve is a plot of true positive rate vs false positive rate. An Ideal classifier has 100% true positives, 0% false positives makes a point in the top-left corner of ROC plot.

They both have an area under the curve (AUC) value of approximately 0.96 which is significantly better than the dummy model AUC of 0.5.



e)

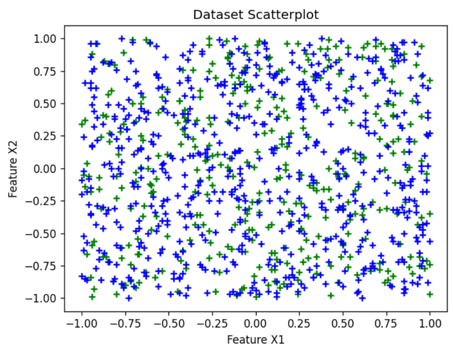
From the Confusion Matrices we can see that the logistic performs slightly better than the other two models with the highest accuracy, recall and precision. The difference here is minimal though and could be due to variations in the testing data. Interestingly the dummy model's accuracy and specificity metrics sum together to 1. This is because it only predicted the majority case which is +1.

Looking at the ROC graph it's clear both the models are quite effective models of the training data, with Logistic Regression looking slightly better. The Logistic regression model has a slightly higher AUC than the kNN model, but both are significantly better than the dummy model's AUC of 0.5.

I would recommend using the Logistic Regression model for this data, with a C value of 1 and with the data transformed to the second order polynomial. This also makes logical sense when compared to our hypothesis of the data in part a).

ii) a)

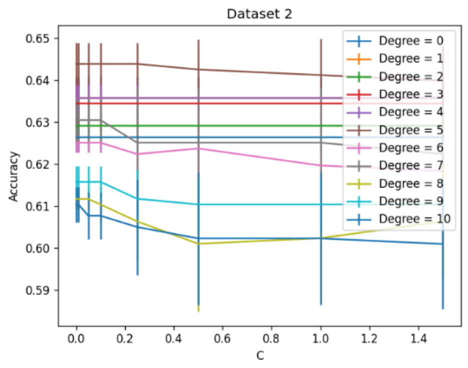
After plotting the second dataset it is clear that the two features will be difficult to categorize as there is a lot of overlap amongst the datapoints.



b) Following the same procedures as before, I plotted the accuracy of regression models of varying degrees against incremental C values.

The plot shows that the regression model results in quite low accuracy for all the degrees, and the C value seems to only really dictate the standard error.

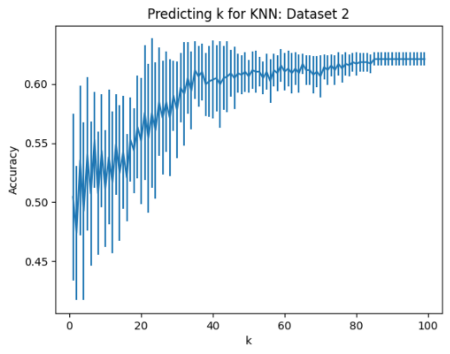
From this plot I would choose degree 5 as the highest polynomial order and let C=0.1 to keep the standard error low.



b)

Following the same procedures as before, I plotted the accuracy of k-NN models with k values from 1 to 100.

Similar to the Regression model, the accuracy of the kNN model is quite low across the board. Interestingly the model becomes slightly more accurate when it considers more neighbours. This means its more accurate when trying to generalize as opposed to fitting closely to the data, which is somewhat concerning.



c)

Below are the confusion matrices for each model, trained using the c, k and polynomial values chosen in the previous sections. Below each confusion matrix are the performance metrics of each model.

Logistic Regression Model

0 True Positives	92 False Positives
0 False Negatives	158 True Negatives

Accuracy = 0.632

True Positive Rate (Recall) = 0

False Positive Rate (Specificity) = 0.368

Precision = 0

K-NN Model

2 True Positives	83 False Positives
6 False Negatives	159 True Negatives

Accuracy = 0.644

True Positive Rate (Recall) = 0.0235

False Positive Rate (Specificity) = 0.3429

Precision = 0.0235

Dummy Model

A baseline dummy model was implemented which predicts the majority every time.

0 True Positives	85 False Positives
0 False Negatives	165 True Negatives

Accuracy = 0.66

True Positive Rate (Recall) = 0

False Positive Rate (Specificity) = 0.34

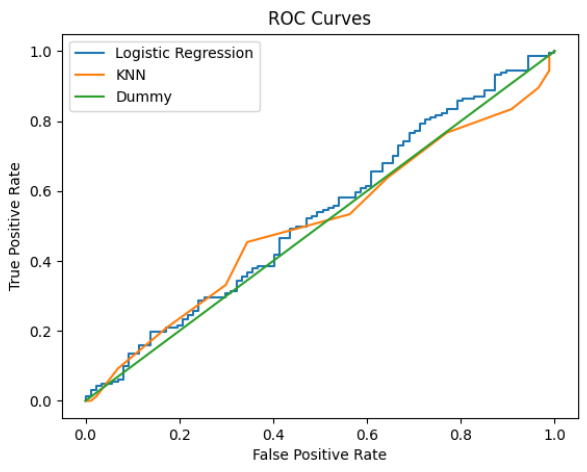
Precision = 0

Seeing how both models predict similarly to the dummy model and perform similarly on all metrics, it’s clear how ineffective these models are at predicting the data.

d)

The ROC curve again shows the similarity between the two models and the dummy model.

They all have an AUC (Area under the curve) of approximately 0.5 where a value of 1 is what is desired.



e)

In conclusion the two models perform quite poorly and give similar outcomes to that of a dummy model picking the majority every time.

Looking at the Confusion matrices and the performance metrics you can see that the dummy model has the highest accuracy, and the lowest false positive rate. Similar to the dummy case, the logistic regression will nearly always pick the majority case which is +1.

Because of this the variation in performance metrics more likely comes down to the variation in how the test data is split for each model. Each model uses a random 25% of the data as test data, but if the same data was used, the performance metrics would probably be even closer.

The similarities between the 3 models can be seen again in the ROC curve.

I wouldn't recommend using either model for non-trivial use cases. Perhaps even the Dummy classifier would be the best in this case as it is easy for humans to interpret and performs just as well as the other two more complicated models. It would also be slightly more efficient to run.

```python
# %%
%matplotlib widget
import numpy as np
import math
from matplotlib import cm
import pandas as pd
import scipy as sp
import seaborn as sn
import matplotlib.pyplot as plt
import matplotlib as matlib
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Ridge

from sklearn.model_selection import KFold
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
from matplotlib.colors import ListedColormap
from sklearn.metrics import roc_curve

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import LinearSVC
from sklearn.dummy import DummyRegressor
from sklearn.dummy import DummyClassifier
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.patches as mpatches

def read_data(filename):
    df = pd.read_csv(filename)
    df.columns = ["X1","X2","y"]
    X1=df.iloc[:,0]
    X2=df.iloc[:,1]
    y=df.iloc[:,2]
    X=np.column_stack((X1,X2))
    return X, y

# %%

def plot_data(X, y, title):
    fig = plt.figure()
    plt.rc('font', size=10)
    X_m1 = X[np.where(y == -1)]
    X_p1 = X[np.where(y == 1)]
    plt.scatter(X_m1[:, 0], X_m1[:, 1], c='g', marker='+',label="negative", s=30)
    plt.scatter(X_p1[:, 0], X_p1[:, 1], c='b', marker='+',label="positive", s=30)
    plt.xlabel('Feature X1')
    plt.ylabel('Feature X2')
    plt.legend(loc = 1, framealpha = 1.0)
    plt.title(title)
    plt.show()

X, y = read_data("week4.csv")
plot_data(X,y,"Dataset 1")

X, y = read_data("week4_1.csv")
plot_data(X,y,"Dataset 2")

# %%
#i) a)
def best_poly_and_c(X, y, title):
    fig = plt.figure()
    poly_test = [0,1,2,3,4,5,6,7,8,9,10]
    Cs = [0.001,0.01,0.05,0.1,0.25,0.5,1,1.5]
    for p in poly_test:
        avg_accuracy=[]
        std_err=[]
        poly_x = PolynomialFeatures(p).fit_transform(X)

        for c in Cs:
            log_reg = LogisticRegression(C=c, max_iter=1000000)
            scores = cross_val_score(log_reg, poly_x, y,cv=5, scoring="accuracy")
            avg_accuracy.append(scores.mean())
            std_err.append(scores.std())

        plt.errorbar(Cs, avg_accuracy,  label="Degree = {0}".format(p), yerr=std_err,)
        plt.xlabel('C')
        plt.ylabel('Accuracy')
        plt.legend()

    plt.title(title)
    plt.legend(loc = 1)
    plt.show()
    return

X, y = read_data("week4.csv")
best_poly_and_c(X,y, "Dataset 1")

X, y = read_data("week4_1.csv")
best_poly_and_c(X,y,"Dataset 2")


# %%
#i) b)
def knn(X,y,title):
    fig = plt.figure()
    x_train, x_test, y_train, y_test = train_test_split(X, y)
    avg_accuracy=[]
    std_err=[]
    k_test = np.array(range(1,100))
    for k in k_test:
        knn = KNeighborsClassifier(n_neighbors=k).fit(x_train,y_train)
        scores = cross_val_score(knn, x_train, y_train, cv=10, scoring="accuracy")
        avg_accuracy.append(scores.mean())
        std_err.append(scores.std())

    plt.errorbar(k_test, avg_accuracy, yerr=std_err)
    plt.xlabel('k')
    plt.ylabel('Accuracy')
    plt.title('Predicting k for KNN: ' + title)
    plt.show()

X, y = read_data("week4.csv")
knn(X,y, "Dataset 1")
X, y = read_data("week4_1.csv")
knn(X,y, "Dataset 2")

# %%
# i) c) & d)

def c_and_d(X, y, c, k, poly):
    fig = plt.figure()

    # c)

    y_pred_log_reg =[]; y_pred_knn = []
    x_poly = PolynomialFeatures(poly).fit_transform(X)
    x_poly_train, x_poly_test, y_poly_train, y_poly_test = train_test_split(x_poly, y)
    x_train, x_test, y_train, y_test = train_test_split(X, y)

    log_reg = LogisticRegression(C=c)
    knn = KNeighborsClassifier(n_neighbors=k)
    dummy = DummyClassifier(strategy="most_frequent")
    log_reg.fit(x_poly_train, y_poly_train)
    knn.fit(x_train, y_train)
    dummy.fit(x_train, y_train)

    y_pred_log_reg = log_reg.predict(x_poly_test)
    y_pred_knn = knn.predict(x_test)
    y_pred_dummy = dummy.predict(x_test)

    log_mat = confusion_matrix(y_poly_test, y_pred_log_reg)
    knn_mat = confusion_matrix(y_test, y_pred_knn)
    dummy_mat = confusion_matrix(y_test, y_pred_dummy)

    print("LogReg \n", log_mat)
    print("Knn \n", knn_mat)
    print("Dummy \n", dummy_mat)

    # d)

    scores = log_reg.predict_proba(x_poly_test)
    fpr, tpr, _ = roc_curve(y_poly_test, scores[:, 1])
    plt.plot(fpr, tpr)

    scores = knn.predict_proba(x_test)
    fpr, tpr, _ = roc_curve(y_test, scores[:, 1])
    plt.plot(fpr, tpr)

    scores = dummy.predict_proba(x_test)
    fpr, tpr, _ = roc_curve(y_test, scores[:, 1])
    plt.plot(fpr, tpr)

    plt.title('ROC Curves')
    plt.legend(['Logistic Regression', 'KNN', 'Dummy'])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.show()

X, y = read_data("week4.csv")
c_and_d(X,y,1,5,2)
X, y = read_data("week4_1.csv")
c_and_d(X,y,0.1,35,5)
```