Assignment 4: Evaluating and selecting models

Ciarán Donegan
16322253

31 October 2020

Note: I wrote up part (ii) of this report first, the second dataset was more useful and so these comments are more in-depth. Since the first dataset was too noisy to do anything useful with, not much interesting can be said in the discussion for it.

Dataset id:13–13–13-1

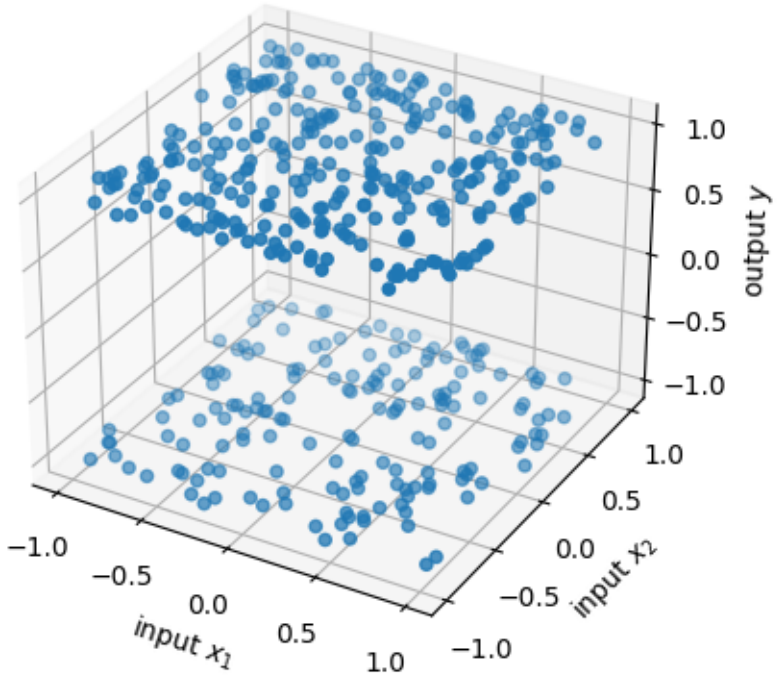
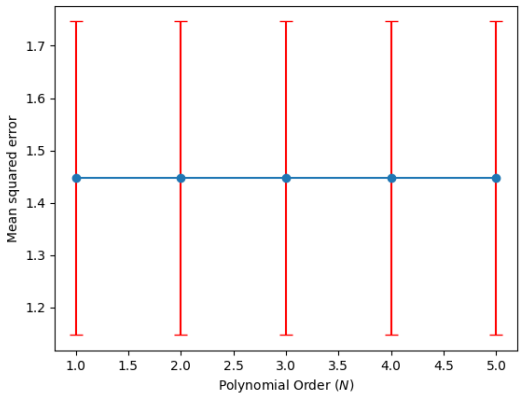


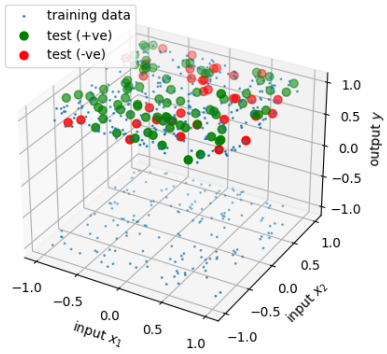
Figure 1: Dataset 1 training data

(i) (a) From figure 1, we can see that this dataset contains two classes but there does not appear to be an obvious pattern to the data. I can't tell *a priori* what order of polynomial features might perform best.

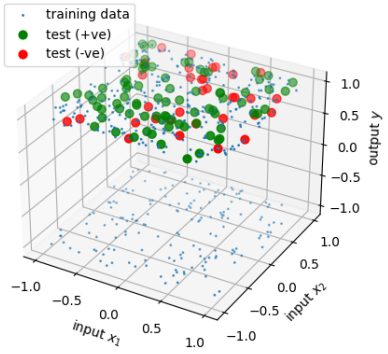
From the sub figures in figure 2, it appears as though no polynomial features for orders in $N =$



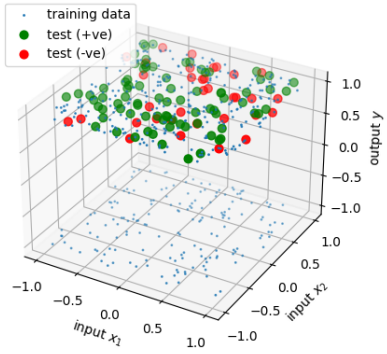
(a) MSE vs N for $N = \{1, 2, 3, 4, 5\}$.



(b) Training data and test predictions for $N = 1$.



(c) Training data and test predictions for $N = 2$.



(d) Training data and test predictions for $N = 3$.

Figure 2: Analysis of the effect of the order of the input features N on logistic regression MSE (using 10-fold cross validation and $C = 1$ for all models). Note: I plotted the training data in blue and reduced the marker size so that the more important test predictions can be more readily seen. The green points are positive examples and red points are negative examples.

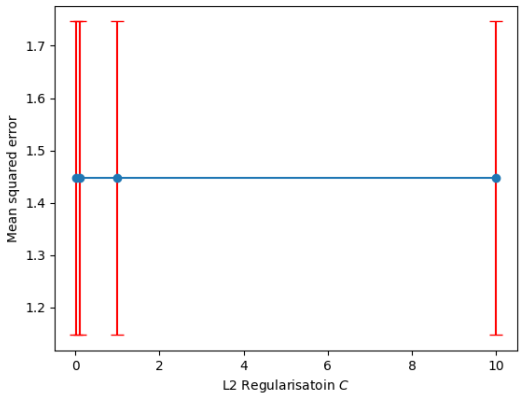
$\{1, 2, 3, 4, 5\}$ is outperforming the others and fitting the data well. (As a side note, I attempted to use even higher order but they were equally bad). In fact, when you look at the predictions for different orders, you see that in each case, the model always predicts positive.

It's hard to decide what order polynomial to use because they are all bad and in reality, I would probably give up on using logistic regression at this point. For the sake of simplicity I will go with $N = 1$, but this is no better than the others.

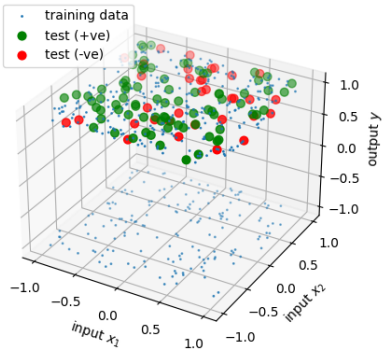
From figure 3, there is no obvious best choice of C. All models perform equally badly under all values of regularisation. For the sake of simplicity, I will use $C = 0.1$.

(b) From the cross-validation analysis in figure 4(a), it *appears* as though $k = 1$ is the best choice. However, what is really occurring with $k = 1$ is that the model is overfitting and learning the training data with little generalisation ability. It turns out that $k = 3$ is a better choice. But in fact, all kNN models are perfroming very badly as seen in the test predictions in figure 4(b-d).

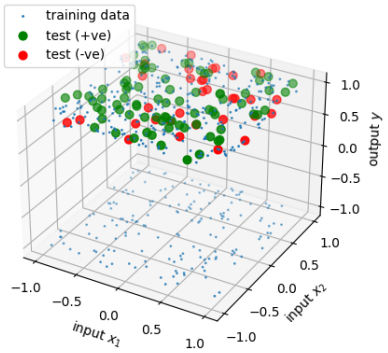
(c) Confusion Matrices are useful for examining your model more closely and give a more detailed overview than a simple accuracy metric. It shows the true positive (TP) and false positives (FP)



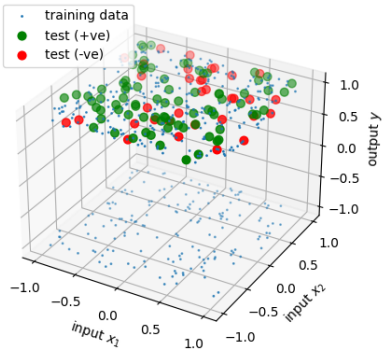
(a) MSE vs C for $C = \{0.01, 0.1, 1, 10\}$.



(b) Training data and test predictions for $C = 0.01$.

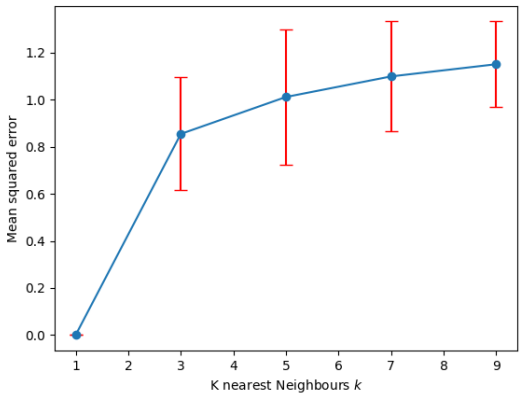


(c) Training data and test predictions for $C = 1$.

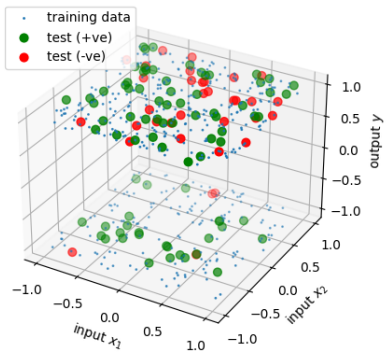


(d) Training data and test predictions for $C = 10$.

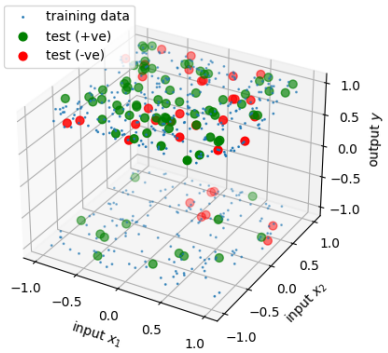
Figure 3: Analysis of the effect of the regularisation parameter C on logistic regression MSE (using 10-fold cross validation and $N = 1$ for all models).



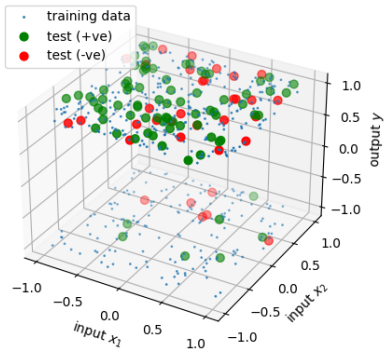
(a) MSE vs k for $k = \{1, 3, 5, 7, 9\}$.



(b) Training data and test predictions for $k = 1$.



(c) Training data and test predictions for $k = 3$.



(d) Training data and test predictions for $k = 5$.

Figure 4: Analysis of the effect of k in k-nearest neighbours on MSE (using 10-fold cross validation and $N = 1$ for all models). Odd values of k were chosen in all cases because choosing an even value of k leads to problems when half of the nearest neighbours belong to each class.

which indicates samples correctly and incorrectly classified as positive respectively. It also shows the false negatives (FN) and true negatives (TN) which indicate samples incorrectly and correctly classified as negative respectively. Ideally, you want a model with high TP and TN i.e. the diagonal values should be large and the others small.

$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$

Best logistic model. $N = 1$ and $C = 0.1$.

$$\begin{bmatrix} 0 & 34 \\ 0 & 81 \end{bmatrix}$$

Best kNN model. $k = 3$.

$$\begin{bmatrix} 7 & 27 \\ 14 & 67 \end{bmatrix}$$

Dummy model. Outputting the most common class (+ve) for all test points.

$$\begin{bmatrix} 0 & 34 \\ 0 & 81 \end{bmatrix}$$

I will discuss these confusion matrices fully in part (d).

(d) See figure 5. Receiver Operating Characteristic (ROC) curve is simply a plot of true positive rate vs false positive rate, where true positive and false positive rate are defined as follows:

$$\texttt{True positive rate} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FN}}$$

$$\texttt{False positive rate} = \frac{\texttt{FP}}{\texttt{TN} + \texttt{FP}}$$

(e) The baseline classifier and the logisitc classifier both perform the same (same confusion matrices and same ROC curve). The kNN model is *slightly* better than the other two, but still terrible.

I could not possibly recommend to use any of these models, they are all terrible. I would instead recommend tossing a coin to make predictions on this useless, noisy dataset.

(ii) (a) From figure 6, we can see that this dataset contains two classes and that there looks to be a strong quadratic relationship between the x_1 features and the output y. This tells me that I should likely be using polynomial features up to degree 2.

Examining what order polynomial features to use.

My guess from observing the training data is to use polynomial order $N = 2$, but I will look at $N = \{1, 2, 3, 4, 5\}$ to confirm or disprove my assumption.

From the sub figures in figure 7, it is clear that polynomial features of order > 1 provide better fits. Since there seems to be very little improvement when using $N = 3$ vs $N = 2$, I believe that $N = 2$ is the best polynomial order to use because it is the simplest while still fitting the data well.

From figure 8, it appears that $C = 10$ is the best choice of regularisation parameter to use. It seems to have the fewest false classifications on the test set (see figure 8 (c)). Furthermore, looking at the MSE vs C plot from k-fold cross validation, $C = 10$ has the lowest MSE, while still having an acceptable standard deviation. This choice of regularisation parameter makes sense to me. If it was very small e.g. $C = 0.01$, I would not trust it becasue there shouldn't be a very strong regularisation needed on such a simple model i.e. only using polynomial features up to $N = 2$.

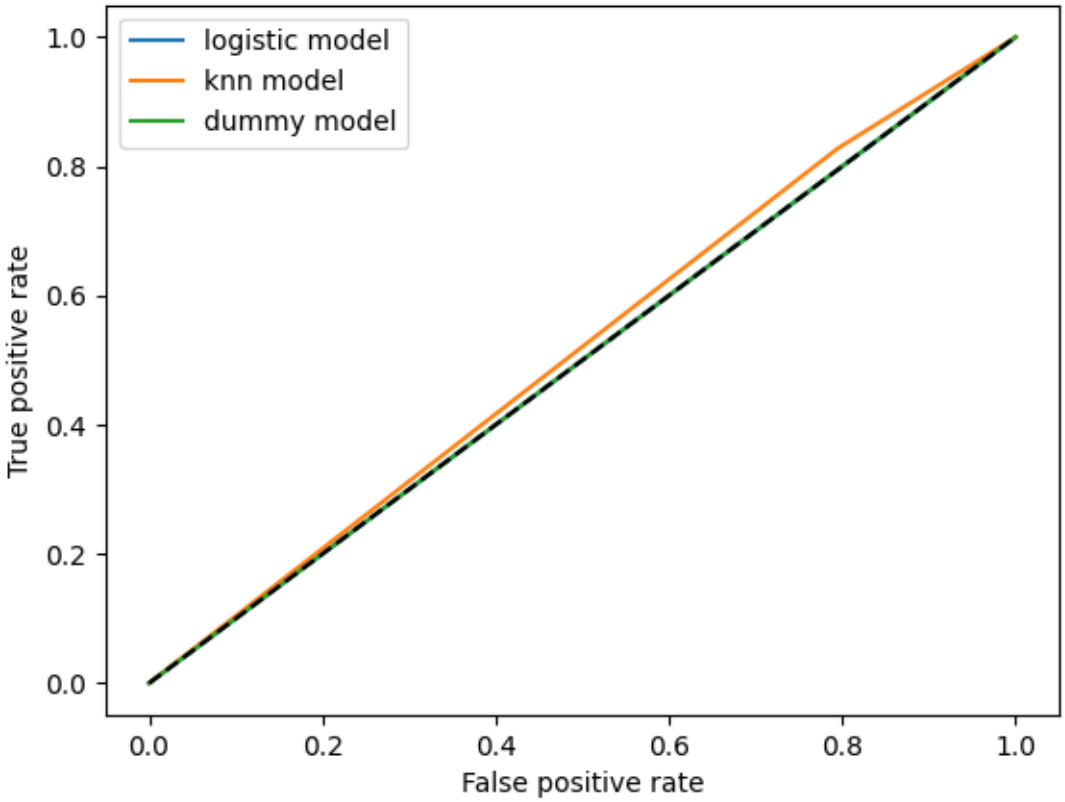


Figure 5: ROC curve for logistic regression model, kNN model and dummy classifier model

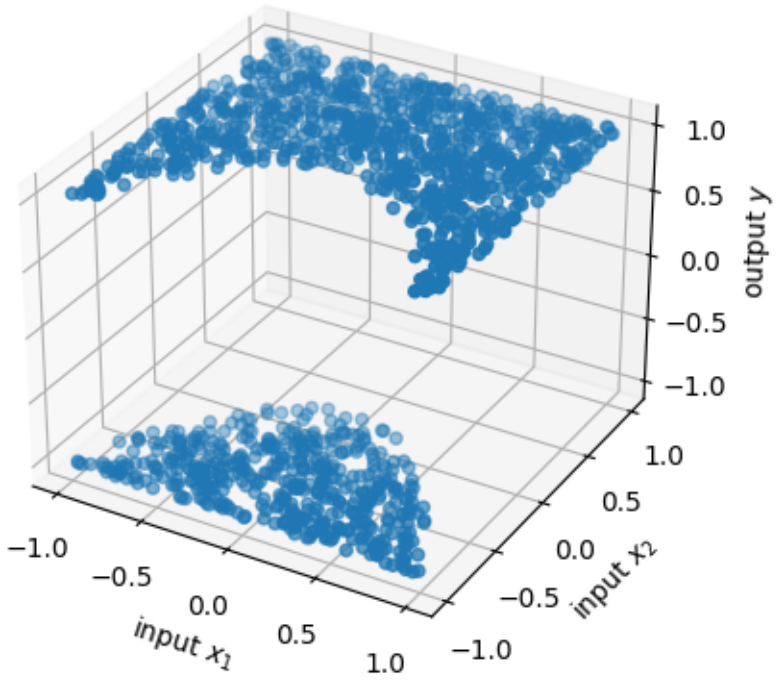


Figure 6: Dataset 2 training data

(b) From the cross-validation analysis in figure 9(a), it *appears* as though $k = 1$ is the best choice. However, what is really occurring with $k = 1$ is that the model is overfitting and learning the training data with little generalisation ability. It turns out that $k = 3$ is a better choice as will be confirmed from the confusion metric in part (c)

(c) Confusion Matrices are useful for examining your model more closely and give a more detailed overview than a simple accuracy metric. It shows the true positive (TP) and false positives (FP) which indicates samples correctly and incorrectly classified as positive respectively. It also shows the false negatives (FN) and true negatives (TN) which indicate samples incorrectly and correctly classified as negative respectively. Ideally, you want a model with high TP and TN i.e. the diagonal values should be large and the others small.

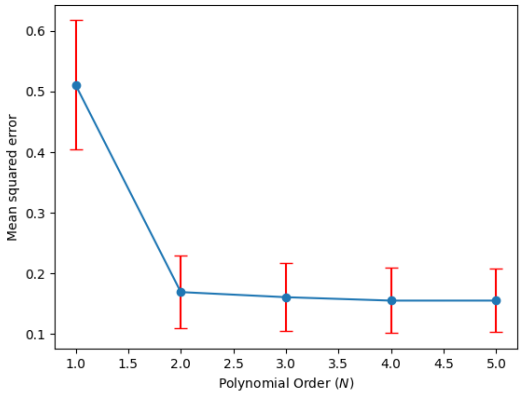
$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$

Best logistic model. $N = 2$ and $C = 10$.

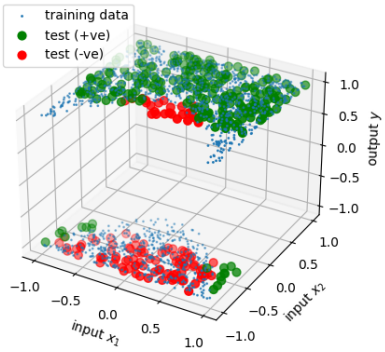
$$\begin{bmatrix} 111 & 7 \\ 5 & 232 \end{bmatrix}$$

Best kNN model. $k = 3$.

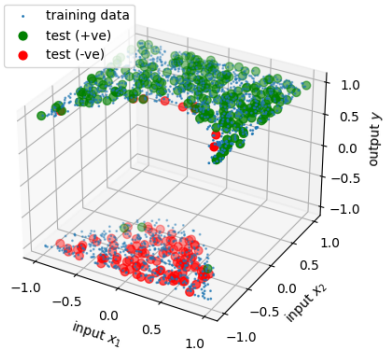
$$\begin{bmatrix} 110 & 8 \\ 9 & 228 \end{bmatrix}$$



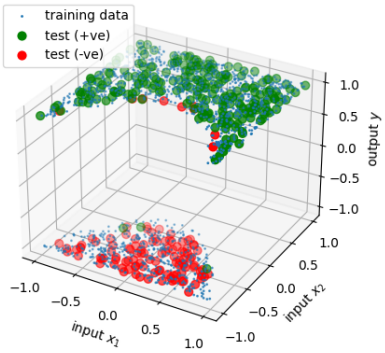
(a) MSE vs N for $N = \{1, 2, 3, 4, 5\}$.



(b) Training data and test predictions for $N = 1$.

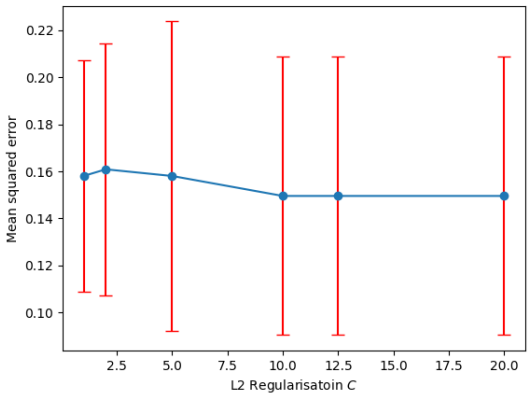


(c) Training data and test predictions for $N = 2$.

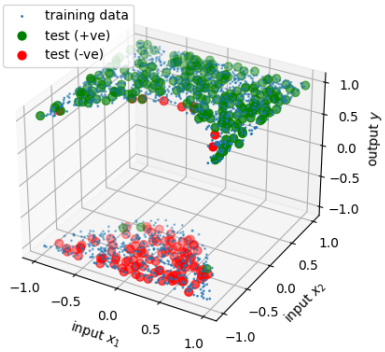


(d) Training data and test predictions for $N = 3$.

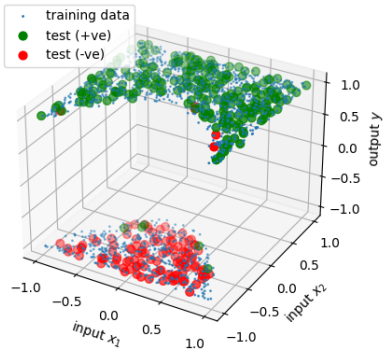
Figure 7: Analysis of the effect of the order of the input features N on logistic regression MSE (using 10-fold cross validation and $C = 1$ for all models). Note: I plotted the training data in blue and reduced the marker size so that the more important test predictions can be more readily seen. The green points are positive examples and red points are negative examples.



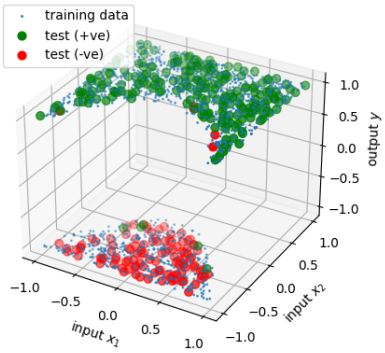
(a) MSE vs C for $C = \{1, 2, 5, 10, 12.5, 20\}$.



(b) Training data and test predictions for $C = 1$.

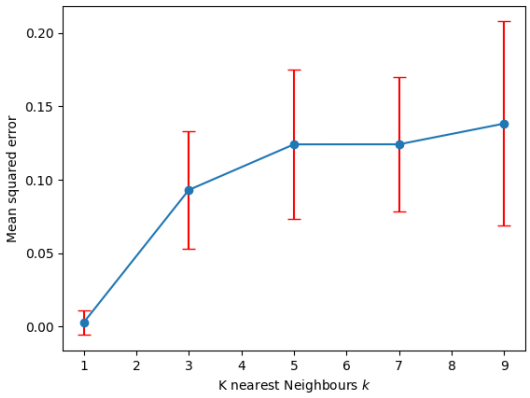


(c) Training data and test predictions for $C = 10$.

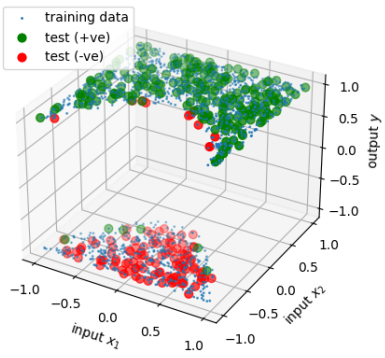


(d) Training data and test predictions for $C = 20$.

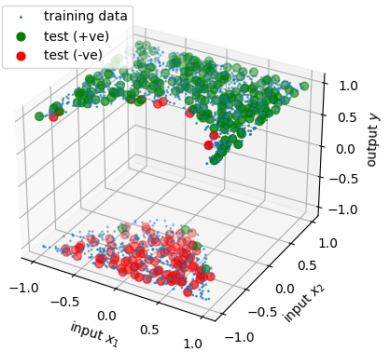
Figure 8: Analysis of the effect of the regularisation parameter C on logistic regression MSE (using 10-fold cross validation and $N = 2$ for all models).



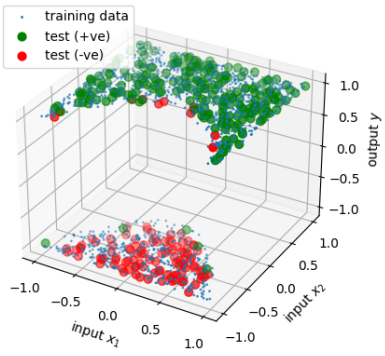
(a) MSE vs k for $k = \{1, 3, 5, 7, 9\}$.



(b) Training data and test predictions for $k = 1$.



(c) Training data and test predictions for $k = 3$.



(d) Training data and test predictions for $k = 5$.

Figure 9: Analysis of the effect of k in k-nearest neighbours on MSE (using 10-fold cross validation and $N = 1$ for all models). Odd values of k were chosen in all cases because choosing an even value of k leads to problems when half of the nearest neighbours belong to each class.

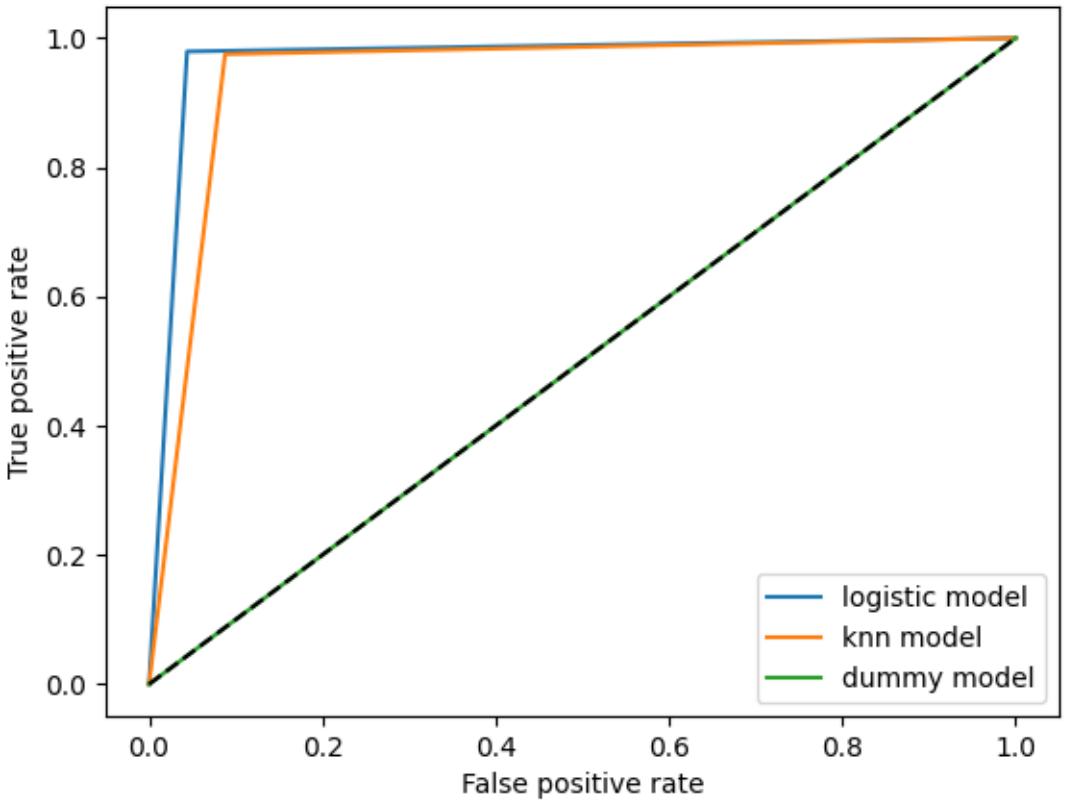


Figure 10: ROC curve for logistic regression model, kNN model and dummy classifier model

Dummy model. Outputting the most common class (+ve) for all test points.

$$\begin{bmatrix} 0 & 118 \\ 0 & 237 \end{bmatrix}$$

I will discuss these confusion matrices fully in part (d).

(d) See figure 10. Receiver Operating Characteristic (ROC) curve is simply a plot of true positive rate vs false positive rate, where true positive and false positive rate are defined as follows:

$$\texttt{True positive rate} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FN}}$$

$$\texttt{False positive rate} = \frac{\texttt{FP}}{\texttt{TN} + \texttt{FP}}$$

(e) The baseline classifier (which just outputs the most common class) is clearly the poorest classifier. From its confusion matrix, we see that it never outputs negative predictions and so has very high false positives. From the ROC curve, we see that its curve lies along the diagonal $TPR = FPR$ indicating that it is essentially just a random classifier.

The logistic model and k-nearest neighbours model both perform very well with the ROC curves very similar. The logistic model is marginally better as can be seen from the fact that its ROC

curve is slightly more to the upper left corner of the ROC plot. From the confusion matrices, it can be seen that the logisitc model has fewer FN and FP predictions than the kNN model.

I would therefore recommend that the logisitc model be used. Not only does it perfom better than the kNN model as described above, but it is also the simpler model and easier to interpret. From the plot of the original training data, it was clear that there was a quadratic relationship in the x_1 input feature. The logistic model can capture this relationship exactly and so will generalise to new, unseen data better than the kNN model.

A Code

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from sklearn.model_selection import KFold, train_test_split
from sklearn.metrics import mean_squared_error, confusion_matrix, classification_report,
                                                    roc_curve
from sklearn.neighbors import KNeighborsClassifier
from sklearn.dummy import DummyClassifier


def read_csv(filename):
    df = pd.read_csv(filename, comment="#", header=None)
    X1 = np.array(df.iloc[:,0])
    X2 = np.array(df.iloc[:,1])
    X = np.column_stack((X1, X2))
    y = np.array(df.iloc[:,2])
    return X, y


def plot_3d_scatter(x1, x2, y, out_dir):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    ax.scatter(x1, x2, y)
    ax.set_xlabel('input $x_1$')
    ax.set_ylabel('input $x_2$')
    ax.set_zlabel('output $y$')
    plt.savefig( out_dir + "/data.png")


def train_logistic_with_l2(X, y, C):
    clf = linear_model.LogisticRegression(C=C, random_state=0)
    clf.fit(X,y)
    # print("Coefficients\n", clf.coef_)
    # print("Intercept\n", clf.intercept_)
    return clf


def train_knn_model(X, y, k):
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X,y)
    return clf


def do_kfold(X, y,  method, order=None, C=None, knn=None, kfolds=10):
    valid_methods = ["logistic", "knn"]
    if method not in valid_methods:
        raise ValueError("Invalid type")

    kf = KFold(n_splits=kfolds)
    mean_error = []
    for train, test in kf.split(X):
        if method == "logistic":
            model = train_logistic_with_l2(X[train], y[train], C)
        else:
            model = train_knn_model(X, y, knn)
        y_pred = model.predict(X[test])
        mean_error.append(mean_squared_error(y[test], y_pred))
    mse = np.array(mean_error).mean()
    std = np.array(mean_error).std()
    # print("MSE = ", mse)
    # print("STD = ", std)
    return model, mse, std
```

```
def full_pipeline(filename, out_dir):
    X, y = read_csv(filename)

    # split into test and traing sets
    # the test set will NEVER be trained on
    # k folds cross validation is done on training set only for hyperparameter selction
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    plot_3d_scatter(X_train[:,0], X_train[:,1], y_train, out_dir)

    # Use cross validation to select polynomial order
    poly_orders = [1, 2, 3, 4, 5]
    mse_list = []
    std_list = []
    for order in poly_orders:
        # add polynomial features up to degree n
        poly = PolynomialFeatures(order)
        X_poly = poly.fit_transform(X_train)
        model, mse, std = do_kfold(X_poly, y_train, method="logistic", order=order, C=1)
        mse_list.append(mse)
        std_list.append(std)

        # evaluate on test set
        X_test_poly = poly.fit_transform(X_test)
        y_pred = model.predict(X_test_poly)
        fig = plt.figure()
        ax = fig.add_subplot(111, projection="3d")
        ax.scatter(X_poly[:,1], X_poly[:,2], y_train, marker="o", s=1, label="training data"
                                                    )
        pos = y_test==1
        neg = y_test==-1
        ax.scatter(X_test_poly[pos,1], X_test_poly[pos,2], y_pred[pos], marker="o", color="g
                                                    ", s=40, label="test (+ve)")
        ax.scatter(X_test_poly[neg,1], X_test_poly[neg,2], y_pred[neg], marker="o",color="r"
                                                    , s=40, label="test (-ve)")
        ax.set_xlabel('input $x_1$')
        ax.set_ylabel('input $x_2$')
        ax.set_zlabel('output $y$')
        plt.legend(loc="upper left")
        plt.savefig(out_dir +  "/logistic_N_" + str(order) + ".png")
    plt.figure()
    plt.xlabel("Polynomial Order ($N$)")
    plt.ylabel("Mean squared error")
    plt.errorbar(poly_orders, mse_list, yerr=std_list, fmt="-o", ecolor="r", capsize=5)
    plt.savefig(out_dir + "/logistic_mse_vs_N.png")


    # Use cross validation to select C regularisatoin in logistic regression
    c_vals = [0.01, 0.1, 1, 10]
    mse_list = []
    std_list = []
    for c in c_vals:
        # add polynomial features up to degree 2
        poly = PolynomialFeatures(2)
        X_poly = poly.fit_transform(X_train)
        model, mse, std = do_kfold(X_poly, y_train, method="logistic", order=2, C=c)
        mse_list.append(mse)
        std_list.append(std)

        # evaluate on test set
        X_test_poly = poly.fit_transform(X_test)
        y_pred = model.predict(X_test_poly)
        fig = plt.figure()
        ax = fig.add_subplot(111, projection="3d")
        ax.scatter(X_poly[:,1], X_poly[:,2], y_train, marker="o", s=1, label="training data"
                                                    )
        pos = y_test==1
        neg = y_test==-1
```

```
        ax.scatter(X_test_poly[pos,1], X_test_poly[pos,2], y_pred[pos], marker="o", color="g
                                                    ", s=40, label="test (+ve)")
        ax.scatter(X_test_poly[neg,1], X_test_poly[neg,2], y_pred[neg], marker="o",color="r"
                                                    , s=40, label="test (-ve)")
        ax.set_xlabel('input $x_1$')
        ax.set_ylabel('input $x_2$')
        ax.set_zlabel('output $y$')
        plt.legend(loc="upper left")
        plt.savefig(out_dir +  "/logistic_N_2" +  "_C_" + str(c) + ".png")
    plt.figure()
    plt.xlabel("L2 Regularisatoin $C$")
    plt.ylabel("Mean squared error")
    plt.errorbar(c_vals, mse_list, yerr=std_list, fmt="-o", ecolor="r", capsize=5)
    plt.savefig(out_dir + "/logistic_mse_vs_C.png")

    # Use cross validation to select k in knn
    knn_vals = [1, 3, 5, 7, 9]
    mse_list = []
    std_list = []
    for k in knn_vals:
        # # add polynomial features up to degree 1
        # # this has no effect on the features but is necessary to keep consitency
        # poly = PolynomialFeatures(1)
        # X_poly = poly.fit_transform(X_train)
        model, mse, std = do_kfold(X_train, y_train, method="knn", knn=k)
        mse_list.append(mse)
        std_list.append(std)

        # evaluate on test set
        y_pred = model.predict(X_test)
        fig = plt.figure()
        ax = fig.add_subplot(111, projection="3d")
        ax.scatter(X_train[:,0], X_train[:,1], y_train, marker="o", s=1, label="training
                                                    data")
        pos = y_test==1
        neg = y_test==-1
        ax.scatter(X_test[pos,0], X_test[pos,1], y_pred[pos], marker="o", color="g", s=40,
                                                    label="test (+ve)")
        ax.scatter(X_test[neg,0], X_test[neg,1], y_pred[neg], marker="o",color="r", s=40,
                                                    label="test (-ve)")
        ax.set_xlabel('input $x_1$')
        ax.set_ylabel('input $x_2$')
        ax.set_zlabel('output $y$')
        plt.legend(loc="upper left")
        plt.savefig(out_dir +  "/knn_k_" + str(k) + ".png")
    plt.figure()
    plt.xlabel("K nearest Neighbours $k$")
    plt.ylabel("Mean squared error")
    plt.errorbar(knn_vals, mse_list, yerr=std_list, fmt="-o", ecolor="r", capsize=5)
    plt.savefig(out_dir + "/knn_mse_vs_k.png")


    # best logisitc model
    poly = PolynomialFeatures(1)
    X_poly = poly.fit_transform(X_train)
    model = train_logistic_with_l2(X_poly, y_train, C=0.1)
    # evaluate on test set
    X_test_poly = poly.fit_transform(X_test)
    y_pred = model.predict(X_test_poly)
    log_fpr, log_tpr, _ = roc_curve(y_test, y_pred)
    print("Best Logistic Model")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))

    # best knn model
    model = train_knn_model(X_train, y_train, k=3)
    y_pred = model.predict(X_test)
    knn_fpr, knn_tpr, _ = roc_curve(y_test, y_pred)
```

```
    print("Best Knn Model")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))

    # Dummy classifier
    dummy = DummyClassifier(strategy="most_frequent").fit(X_train, y_train)
    y_dummy = dummy.predict(X_test)
    dummy_fpr, dummy_tpr, _ = roc_curve(y_test, y_dummy)
    print("Dummy Model")
    print(confusion_matrix(y_test, y_dummy))
    print(classification_report(y_test, y_dummy))

    # plot ROC curve
    plt.figure()
    plt.plot(log_fpr, log_tpr, label="logistic model")
    plt.plot(knn_fpr, knn_tpr, label="knn model")
    plt.plot(dummy_fpr, dummy_tpr, label="dummy model")
    plt.plot([0, 1], [0, 1], color='k',linestyle='--')
    plt.xlabel("False positive rate")
    plt.ylabel("True positive rate")
    plt.legend()
    plt.savefig(out_dir + "/ROC.png")


if __name__ == "__main__":

    # Just to keep stuff tidy
    # Output directory for each dataset
    out_path_1 = "dataset_1"
    out_path_2 = "dataset_2"
    if not os.path.exists(out_path_1):
        os.makedirs(out_path_1)
    if not os.path.exists(out_path_2):
        os.makedirs(out_path_2)

    full_pipeline("week4-b.csv", out_path_2 )
    full_pipeline("week4-a.csv", out_path_1 )
```