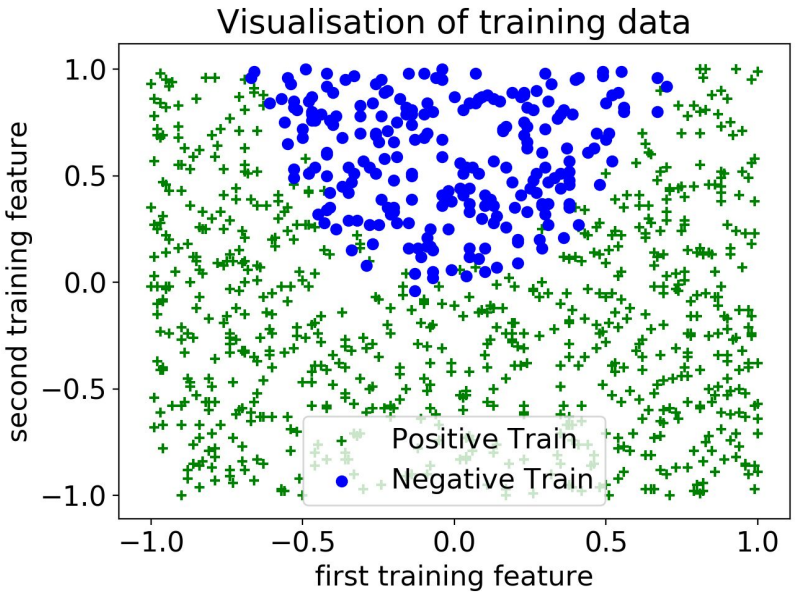
Dataset 1:
(a)
Graph of training data



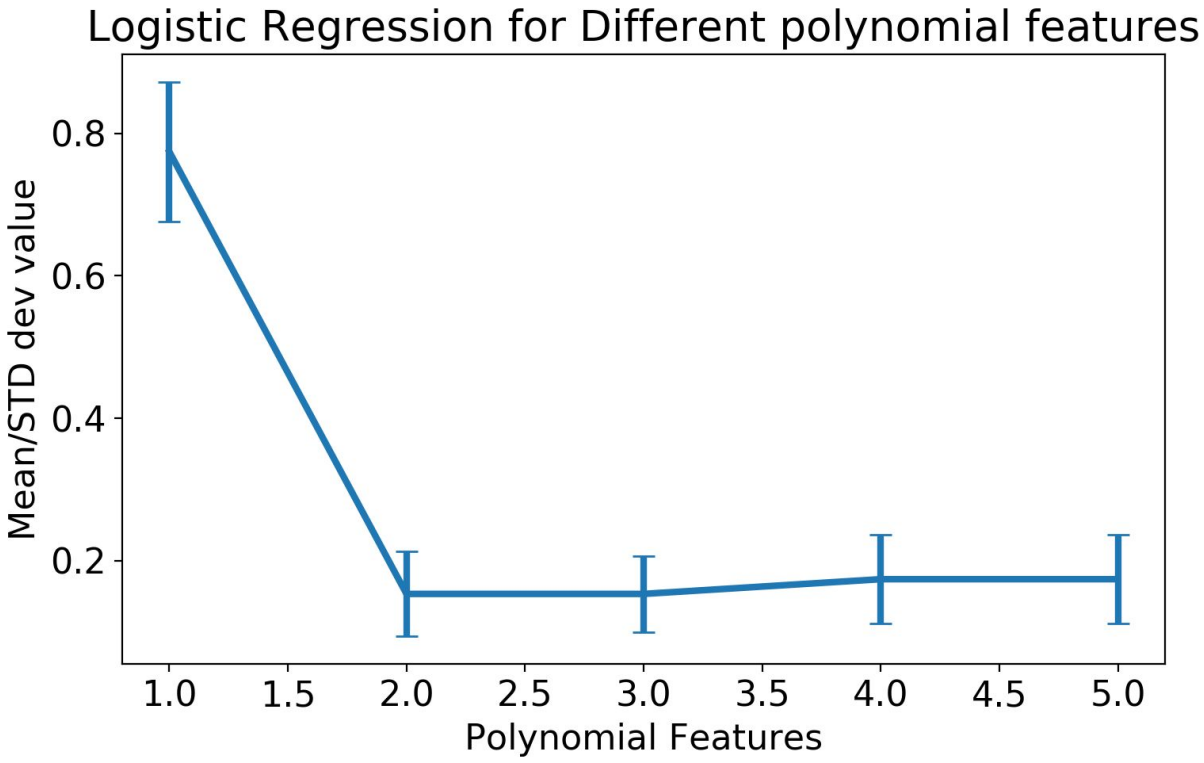
To visualize the data I first split the data into positive and negative data point arrays. I wrote a function called split to do this *(Appendix a.1.1)*. The split function iterates over the values and using the passed in result value (positive/negative) it then adds the features to a positive or negative feature array. These feature arrays are then combined to return two arrays, the positive and negative data point arrays. Using the two returned arrays I then plotted them, giving the positive data point pairs a marker of '+' and a color of green, and the negative point pairs a marker of 'o' and a color of blue *(Appendix a.1.2)*.

As we can see from the graphed data, the data represents a quadratic shape with negative data surrounded by positive data points. It is apparent that there is also some noise with the sleight overlap of some positive/negative area.

(i) Max Order Polynomial

Judging from the quadratic nature of the data initial observations would lead to the presumption of polynomial features being added to the second degree will improve the model (as it is quadratic data) and beyond that, any improvements will be marginal.

Cross-Validation

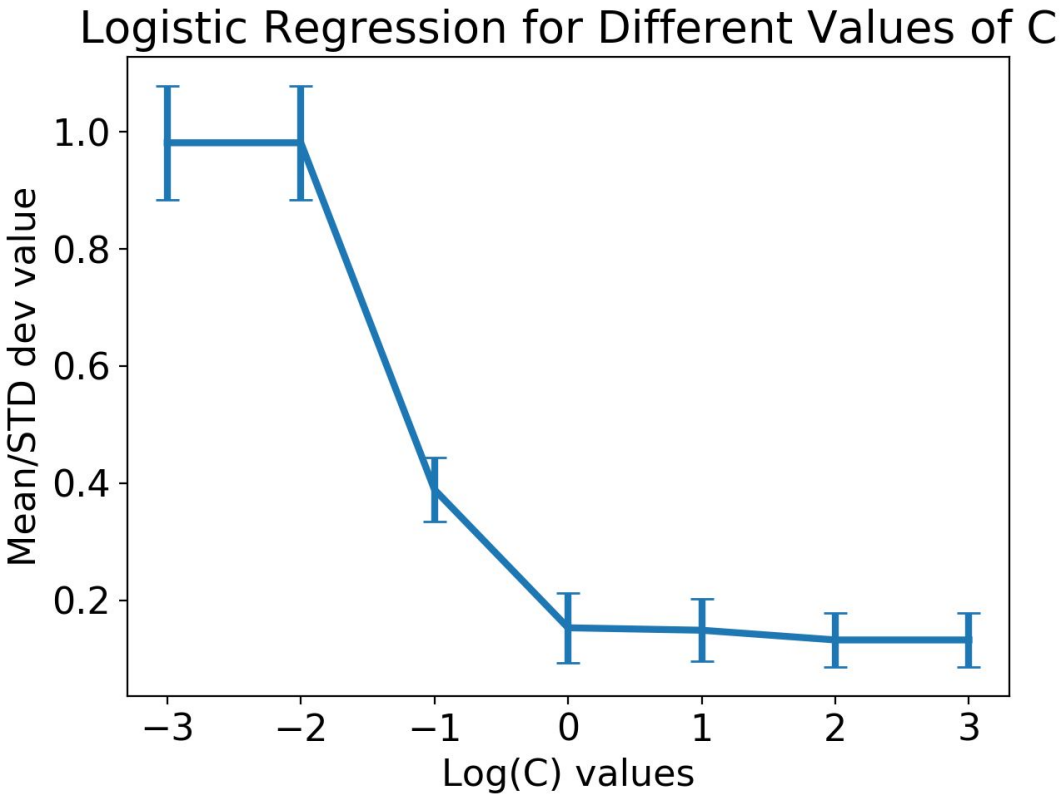


For cross-validation, I iterated over a number of polynomial features and trained a model for each of them. I calculated the mean of the mean squared error estimate for 5 folds for each degree of polynomial features *(Appendix a.1.3)* along with calculating the standard deviation and using it to plot the error bars *(Appendix a.1.4)*. As we can see in the above graph the polynomial feature being used is on the x-axis and the mean and standard deviation on the y-axis. All models were trained with C=1 to be consistent and only compare the degree of polynomial features. From this, we can conclude that beyond 2 features there is little to no improvement in our model showing that the 2 degrees we predicted are the best choice, as beyond this the mean MSE does not improve nor does the variance.

(ii) Weight Given to C

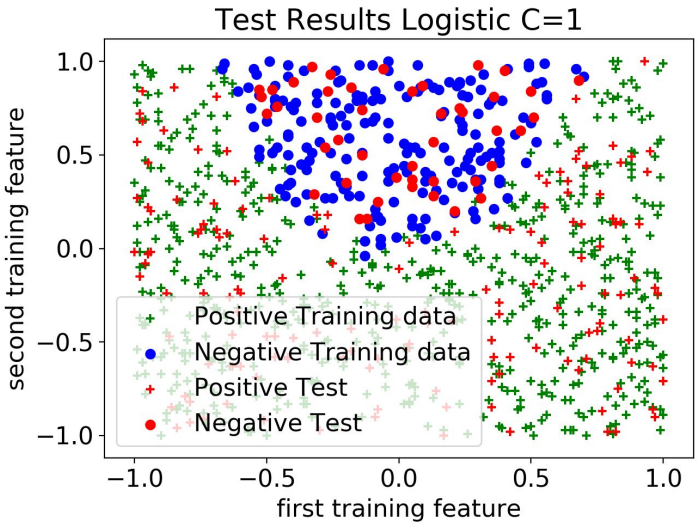
Cross-Validation

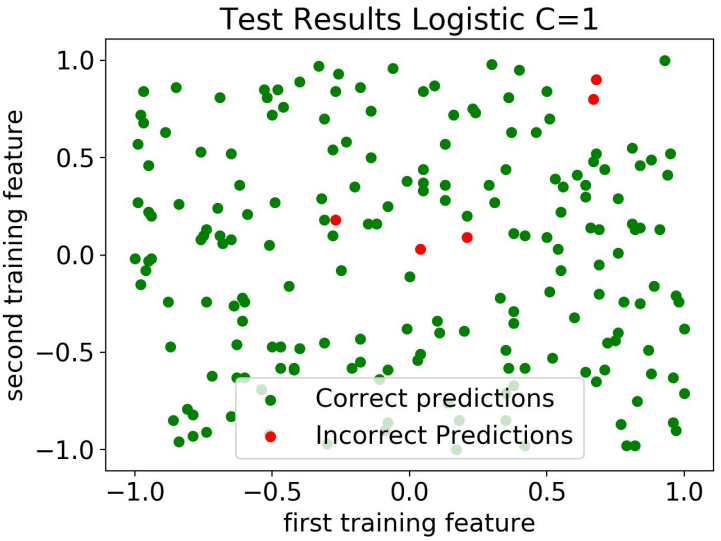
Using the found degree of 2, I repeated the previous process and iterated over different values for C *(Appendix a.1.5)* instead of different degrees of polynomial expansion. I started off with an extremely small penalty and increased by factors of 10 to find the best sort of range to work with. To display this clearly and ensure a readable graph I graphed the log of the C values against the mean and error bars *(Appendix a.1.6)*.



As we can see for the above graph before 0 for very small values there was a large mean error when calculating the mean squared error of the estimates, there was also a high variance, beyond C=1 (the value at 0) the MSE scores dropped dramatically. The score when C=1 performed essentially equally to the other models. I ran this numerous times a of splitting data and noise it would alternate between the values of C=1, C=10, C=100 and C=1000. Upong graphing against the test data, C=1 consistently slightly out performed the others so I choose that as the best model, however they were all fairly similar.

Best Logistic Model Results



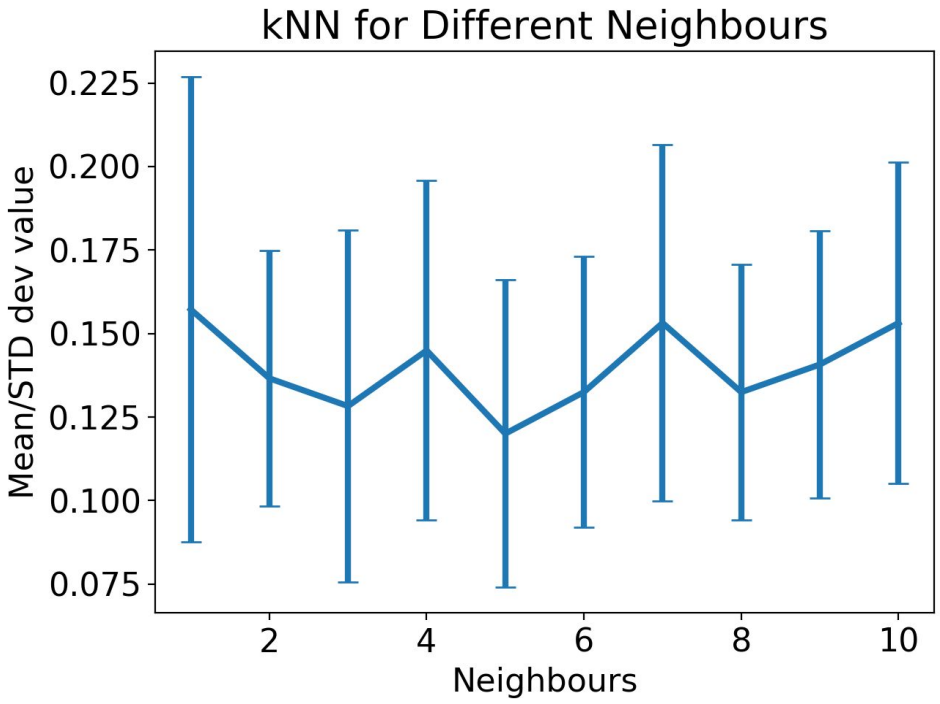


For C=1 with a 2nd-degree polynomial augmentation of the features, this is the result of the model graphed. The plotting function to produce the graph on the left takes in the positive training data, negative training data, positive test and negative test. It can be seen at *(Appendix a.1.7)* and it is used throughout the assignment and will be referenced as such. To get the graph on the right the data is split into correct and incorrect predictions by comparing predicted values to actual test values *(Appendix a.1.8)* and then graphed *(Appendix a.1.9)*.

As we can see from this, we have a relatively accurate model that was deduced from cross-validation to find penalty and how to optimally engineer the features with the polynomial additions to the second degree.

(b)

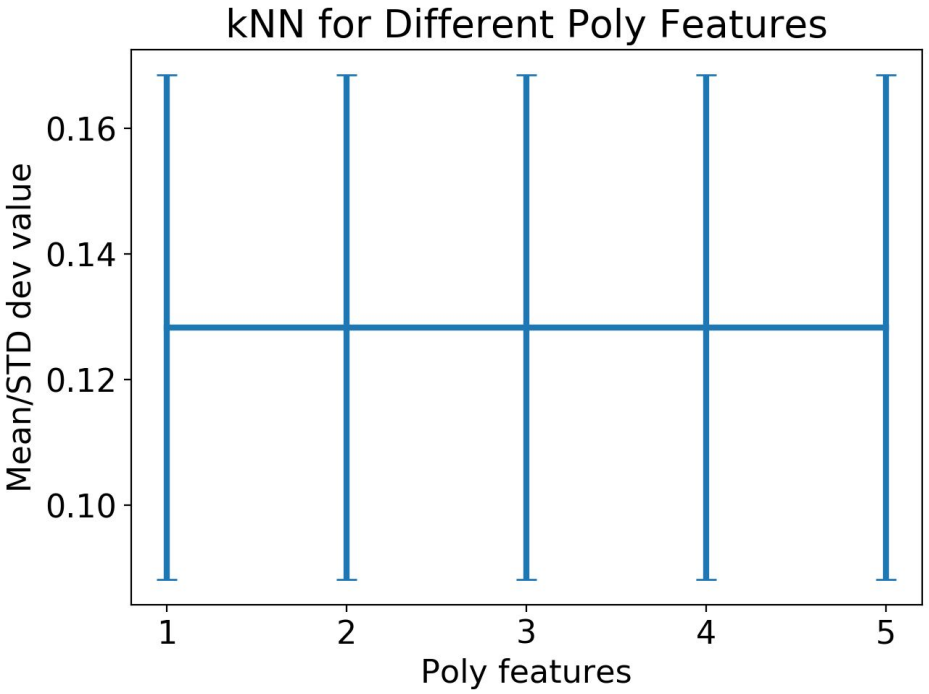
Cross-Validation to select k



I repeated a similar process as above for the kNN classifier. I first created an array of k values and iterated over it performing cross-validation for 5 folds. For each iteration again the mean of the mean squared error and the standard deviation was found *(Appendix b.1.1)*. These mean and standard deviations were then graphed against the number of neighbours used for k *(Appendix b.1.2)*.

As we can see from the graph, the mean of the mse estimates varies a lot based on which value is chosen for k. The lowest value would be the most accurate model that has the lowest variance also. In this case, it is k=5 as it has the lowest mean of the mean squared errors during the cross-validation along with similar enough variance to the other models.

Augmenting with Polynomial Features

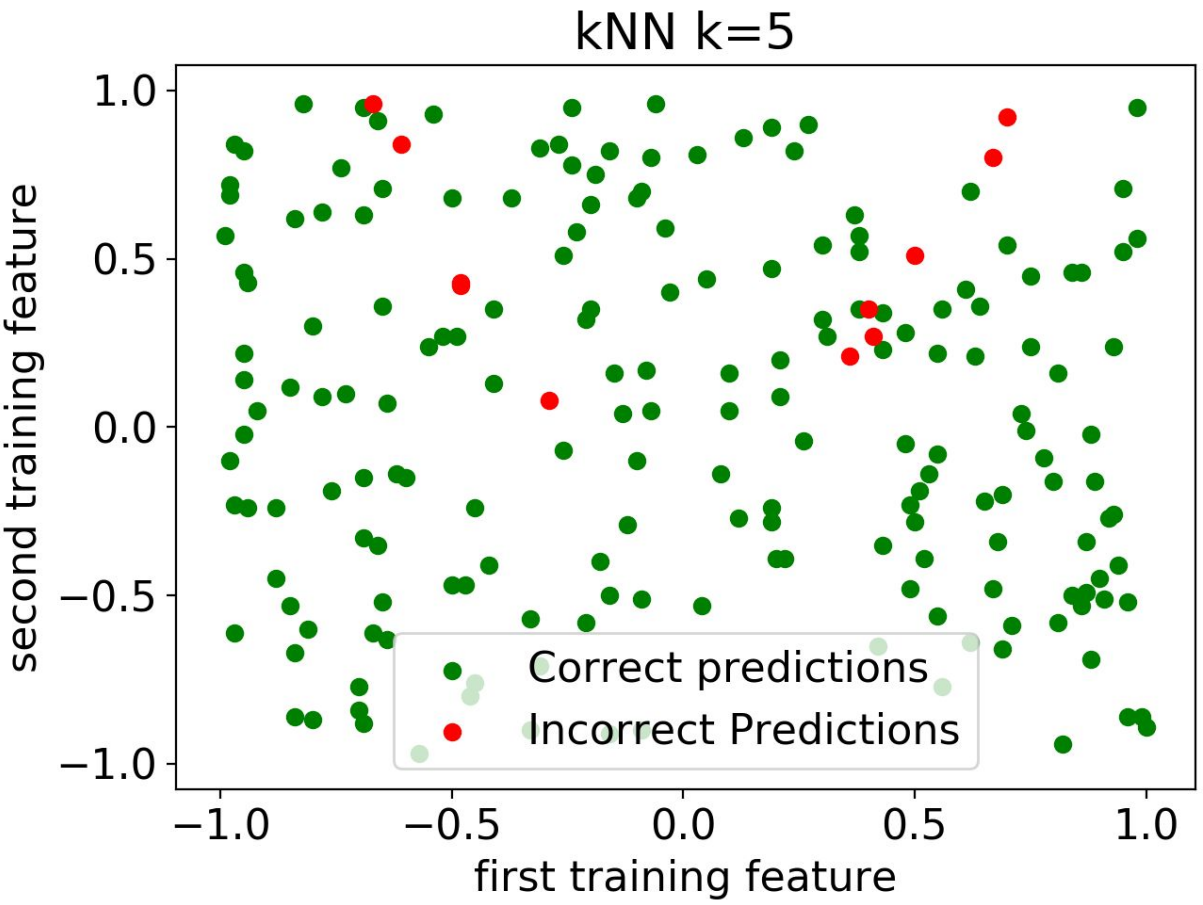


Using the best value for neighbours (k=5) from above, I then repeated the process iterating over different degrees of polynomials for 5 folds the same way I did for logistic regression. See explanation there for methodology and code *(Appendix b.1.3)*.

I then graphed the number of polynomial degree features added against the mean of the mean squared error/ standard deviation *(Appendix b.1.4)* similar to in the previous question.

As we can see from the above graph, the number of polynomial features added through feature engineering does not change the results of the kNN model significantly. I also tried more neighbours (up to 100) which also did not change the model significantly.

Best kNN Model Results
N = 5



As the previous results suggested 5 being the best due to the lowest mean MSE and standard deviation being relatively equal, I chose 5 and no polynomial features for my best model.

As we can see I graphed incorrect vs correct the same as i did for (a) logistic model.

We can see here the model performs extremely well, however around the boundaries of the quadratic shape of negative data there is some misclassification again. This could be due to noise in the data set.

(c)

Best kNN confusion matrix - (a)

k=5

True-negative: 37	False-positive: 3
False-negative: 1	True-positive: 153

Best logistic confusion matrix - (b)

C=1, Polynomial features=2

True-negative: 40	False-positive: 3
False-negative: 5	True-positive: 146

Random prediction confusion matrix using Dummy Classifier - (c)

TP 0.520618556701031
FP 0.47938144329896903

True-negative: 17	False-positive: 26
False-negative: 74	True-positive: 77

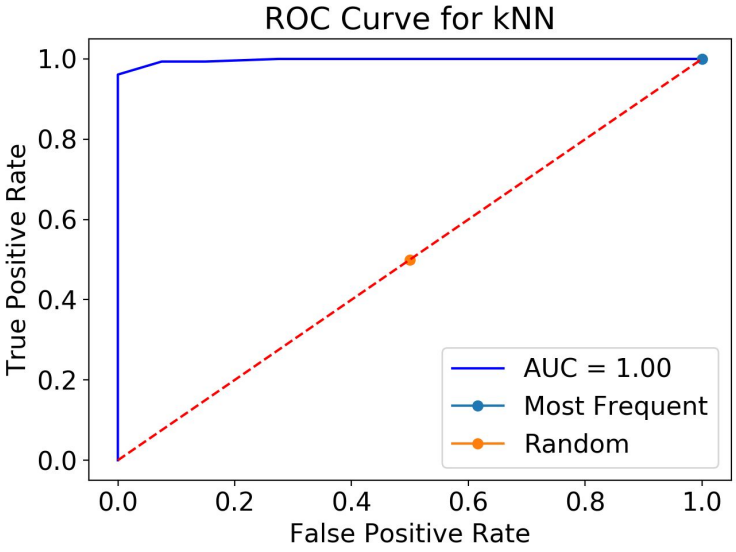
Most common class confusion matrix - (d)

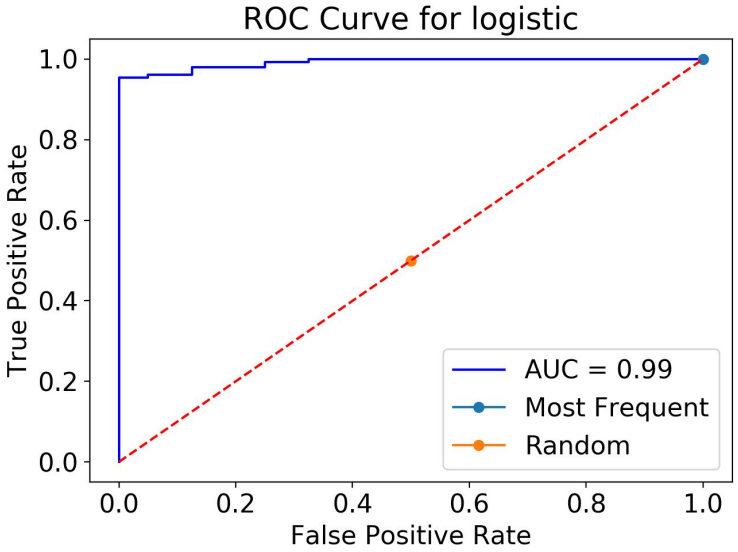
Most common case = 1

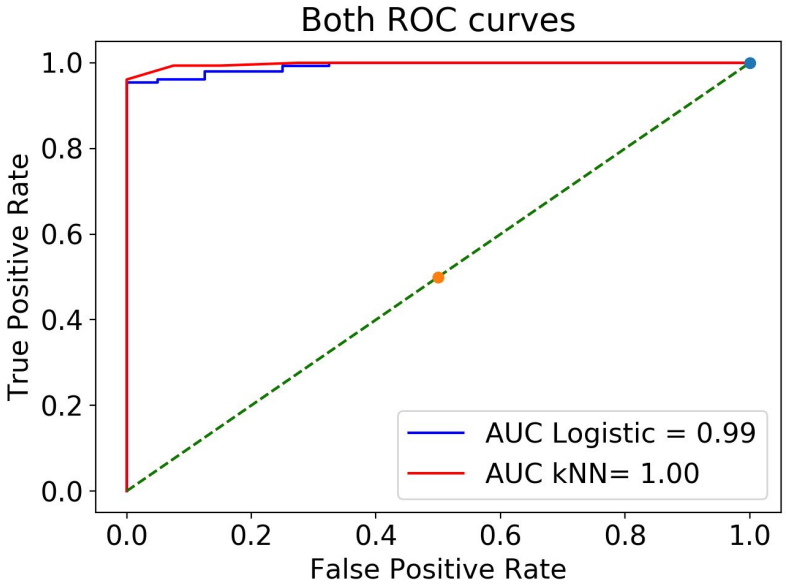
True-negative: 0	False-positive: 43
False-negative: 0	True-positive: 151

Our kNN model performed overall the best with only 3 false positives and 1 false negative, slightly better than our logistic model, however this could be due to noise in the data. Both models significantly outperformed the baseline predictors. The code for the confusion matrices can be seen at *(Appendix c.1.1)* and compares the predicted results to actual results to generate the matrix.

(d) ROC Curves







Upon plotting the ROC curve we see the balance between our true positive and false positive rate for each of the models. The point of the most frequent prediction (positive or 1) was also added and can be seen at the top right. I wrote a function to find the most common class by summing the actual values of the array and returning 1 as the most common class if the sum was greater than 0 and -1 if the sum of the array was less than 0, in the case of there being an even number it would return 0 indicating there is no common class and this baseline comparison would not work *(Appendix d.1.1)*. I also used a random prediction for the baseline. Here I graphed the theoretical random (0.5) rather than the results of my dummy classifier random prediction, as repeatedly simulating the model would lead to an outcome close to this. Upon comparisons we can see obviously predicting true all the time will give the most true positives but also the most false positives, while the random point lies

somewhere in the model of the graph on the 45 degree line as it is getting roughly half of the predicted positive values correctly.

Comparing the two models, they perform remarkably similarly. As the ideal classifier would have 100% true positives and 0% false positives, which would represent a point in the top left of the curve, our prediction models almost match this besides for a few small misclassifications which would be due to noise. To get an exact value we can use the area under the curve (auc), an ideal value here would be 1 representing all correctly classified values. As we can see the area under the curve for kNN = 1 and the area under the curve for the logistic model is 0.99. This would lead to the conclusion that the kNN is slightly better as it predicts correctly close to 100% of the time while the logistic model is closer to 99%. It is also worth noting again this could be due to noise and upon repeating the experiment multiple times for different random selections in the data sets results will vary slightly. Upon repeating it I got essentially the same results within 1-2% every time.

These values are found using sklearns roc_curve and roc_auc. Roc_curve returns two arrays which we can then graph representing the true and false positives while the auc function calculates the area under the curve. I then plotted these values separately *(Appendix d.1.2)* and together in one graph *(Appendix d.1.3)*

(e) Evaluate performance and pick the best

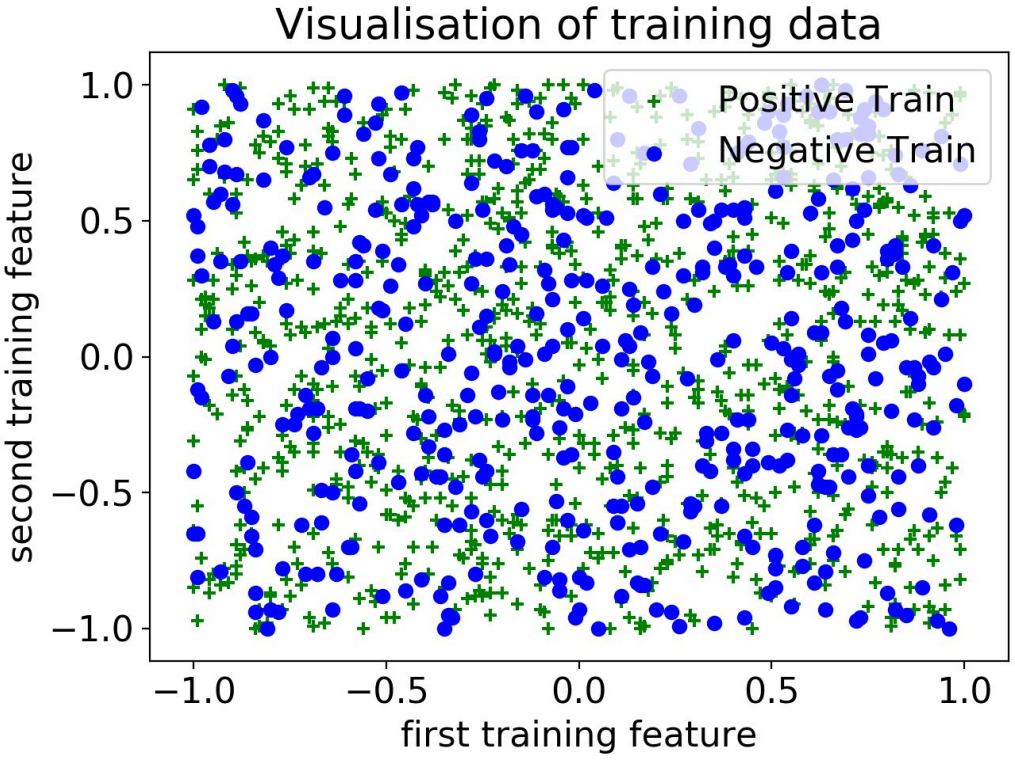
As mentioned above as the auc for the kNN classifier is slightly better than the logistic classifier, it outperforms the logistic classifier, although this may be just due to noise in the data and both perform very similarly. When models are this similarly positioned, trade offs may be made between performance of the model and memory usage, where logistic would slightly edge out kNN as it has to keep track of less points for memory and performs slightly faster, however results of the model are very similar.

For dataset 2 I won't repeat explanations of code - refer to section 1 for that. Results of the machine learning models and cross-validation etc. will be discussed.

Dataset 2:

(a)

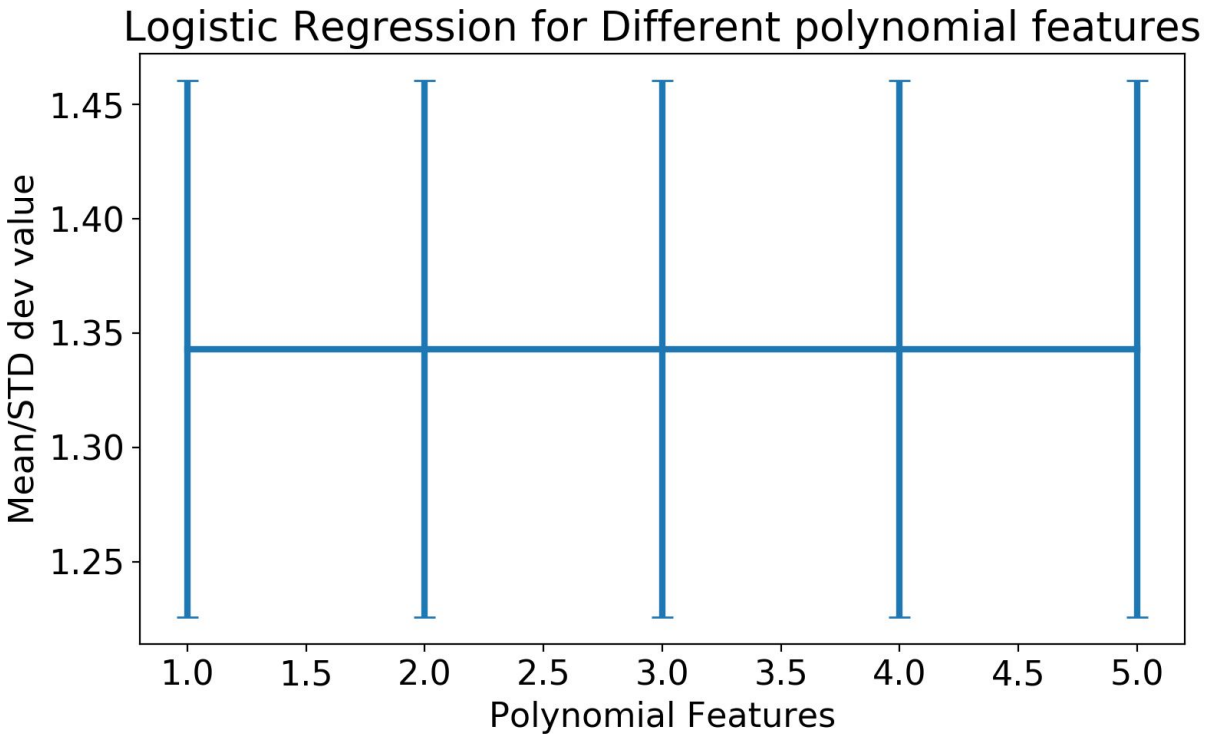
Graph of training data



As we can see from the data, it resembles pure noise so it is unlikely we will be able to model it accurately.

(i) Max Order Polynomial

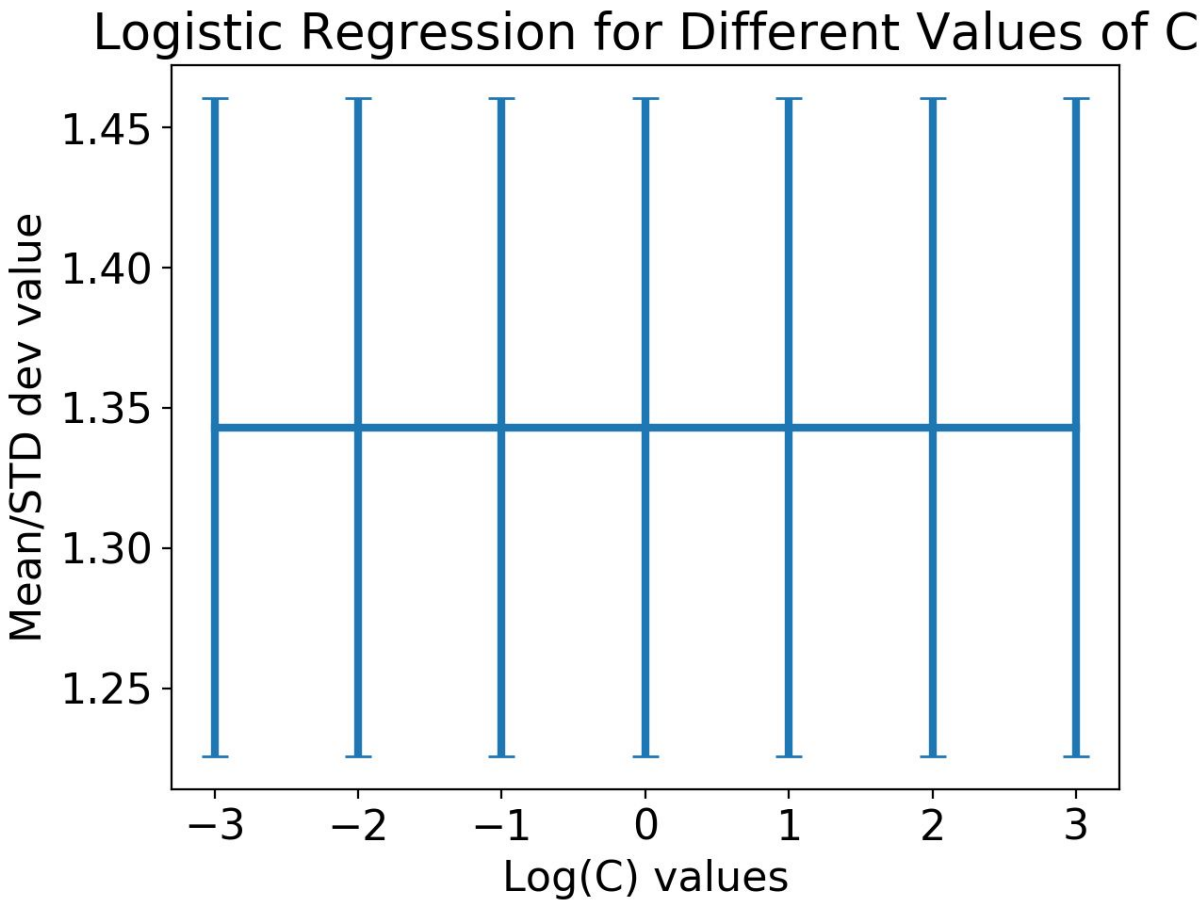
Cross-Validation



As we can see in the above graph, the mean of the mse and standard deviation do not change based on the number of polynomial features we add to the data. This is likely because it is just noise and cannot improve.

(ii) Weight Given to C

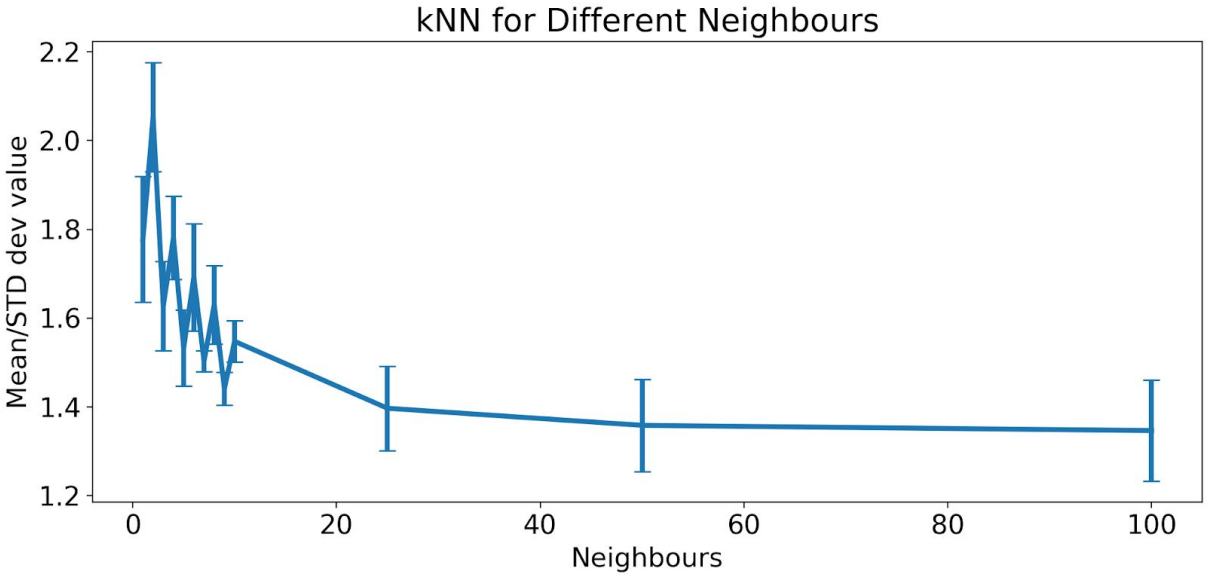
Cross-Validation



Similarly to above, our model does not improve with different penalty features as it cannot model the data as it is just noise.

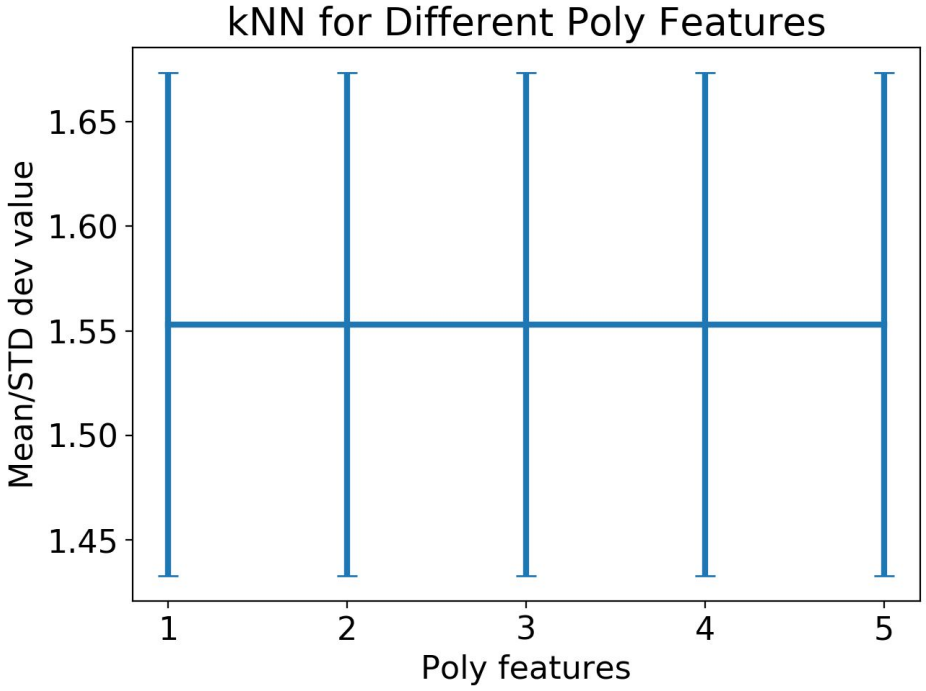
(b)

Cross-Validation to select k



As we can see here, the mean of the mean squared error actually reduces a lot for 20+ neighbours. I selected 50 for my best model as it had a low mean squared error and standard deviation however it is unlikely to improve model performance significantly as we are just modelling noise. It is likely here including so many neighbours we are just predicting the most common case.

Augmenting with Polynomial Features



Similarly to the previous kNN model, we do not see any drastic improvements adding polynomial features as kNN models a distance between points and is not significantly aided when the data appears to be just noise.

(c)

Best kNN confusion matrix

n=50

True-negative: 0	False-positive: 85
False-negative: 2	True-positive: 184

Best logistic confusion matrix C=1, features don't matter but used 2

True-negative: 0	False-positive: 85
False-negative: 0	True-positive: 186

Random prediction confusion matrix using Dummy Classifier

True-negative: 49	False-positive: 36
False-negative: 89	True-positive: 97

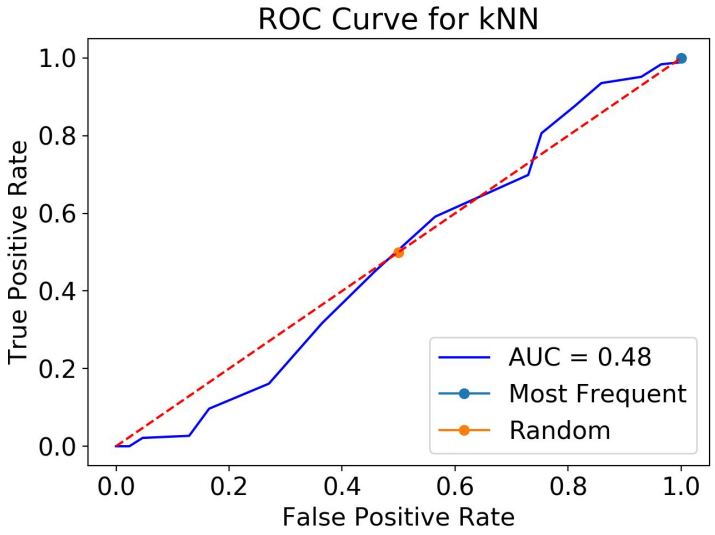
Most common class confusion matrix

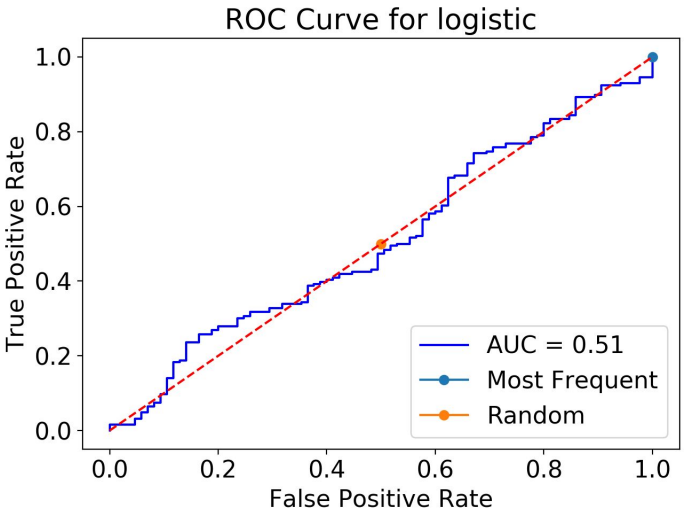
most common case : 1

True-negative: 0	False-positive: 85
False-negative: 0	True-positive: 186

As we can see, our trained logistic and kNN models simply predict (or to close to predict) the most common value in our dataset. This is due to kNN considering so many neighbours that it just picks the most common, and both models not being able to accurately fit the data as it is just noise. Neither out performs the baseline model of predicting the most common case.

(d) ROC Curves





ROC curves generated same as before, as we can see they perform roughly even to the baselines with the area under the curve being close to 0.5, meaning that the models are just predicting the most common value therefore the true positive rate and false positive rate will be equal.

(e) Evaluate performance and pick the best

Neither model fits the data well as they are trying to fit noise, as we can see in the roc curves the logistic performed slightly better but this is just due to noise in the data set. The models perform essentially the same as the baseline classifiers Therefore the best model would just be predicting the most common value.

Appendix

(a.1.1)

```
def split(X1,X2,y):
    """ Splits data of two features (X1,X2) into positive and negative data point arrays
        using result of data (y). Returns two arrays (positive and negative data point pairs)

    """

    X1positive = []
    X2positive = []

    X1negative = []
    X2negative = []

    # iterate over both
    for (x1i, x2i, yi) in zip(X1, X2, y):

        if(yi == 1):
            # if positive add to positive array
            X1positive.append(x1i)
            X2positive.append(x2i)
        else:
            # if negative add to negative array
            X1negative.append(x1i)
            X2negative.append(x2i)

    Xpositive = np.column_stack((X1positive,X2positive))
    Xnegative = np.column_stack((X1negative,X2negative))

    return Xpositive, Xnegative
```

(a.1.2)

```
def plot_splitted_data(positive,negative):
    plt.title('Visualisation of training data')
    plt.xlabel('first training feature')
    plt.ylabel('second training feature')
    plt.scatter(positive[:,0],positive[:,1], label="Positive Train",color='g',marker='+')
    plt.scatter(negative[:,0],negative[:,1], label="Negative Train",color='b')
    plt.legend()
    plt.show()
```

(a.1.3)

```
polynomial_features = [1,2,3,4,5]
std_devs = []
means = []
for degree in polynomial_features:
    kf = KFold(n_splits=5)
    c = 1
    mse_estimates = []
    for train, test in kf.split(X):
        poly_transform = PolynomialFeatures(degree=degree)
        model_logistic_c_1 = LogisticRegression(penalty="l2",C=c)
        model_logistic_c_1 .fit(poly_transform.fit_transform(X[train]),y[train])

        X_test_space = create_test_space(1)
        y_pred_i = model_logistic_c_1.predict(poly_transform.fit_transform(X[test]))
        mse_estimates.append(mean_squared_error(y_pred_i, y[test]))

    means.append(np.mean(mse_estimates))
    std_devs.append(np.std(mse_estimates))

print(means)
print(std_devs)
```

(a.1.4)

```
plt.errorbar(polynomial_features,means,yerr=std_devs,linewidth=3,capsize=5)
plt.title('Logistic Regression for Different polynomial features')
plt.xlabel('Polynomial Features');
plt.ylabel('Mean/STD dev value')
plt.show()
```

(a.1.5)

```
# K FOLDS TO GET STD/MEAN W/ C Values
c_values = [0.001,0.01,0.1,1,10,100,1000]
std_devs = []
means = []
for ci in c_values:
    kf = KFold(n_splits=5)
    mse_estimates = []
    for train, test in kf.split(X):
        poly_transform = PolynomialFeatures(degree=2)
        model_logistic_c_1 = LogisticRegression(penalty="l2",C=ci)
        model_logistic_c_1 .fit(poly_transform.fit_transform(X[train]),y[train])
        y_pred_i = model_logistic_c_1.predict(poly_transform.fit_transform(X[test]))
        mse_estimates.append(mean_squared_error(y_pred_i, y[test]))

    means.append(np.mean(mse_estimates))
    std_devs.append(np.std(mse_estimates))

print(means)
print(std_devs)
```

(a.1.6)

```
plt.errorbar(np.log10(c_values),means,yerr=std_devs,linewidth=3,capsize=5)
plt.title('Logistic Regression for Different Values of C')
plt.xlabel('Log(C) values');
plt.ylabel('Mean/STD dev value')
plt.show()
```

(a.1.7)

```
def plot_training_and_test_data_graph(positive_train,negative_train,positive_test,negative_test,title):
    plt.title(title)
    plt.xlabel('first training feature')
    plt.ylabel('second training feature')
    plt.scatter(positive_train[:,0],positive_train[:,1], label="Positive Training data",color='g',marker='+')
    plt.scatter(negative_train[:,0],negative_train[:,1], label="Negative Training data",color='b')
    plt.scatter(positive_test[:,0],positive_test[:,1], label="Positive Test",color='r',marker='+')
    plt.scatter(negative_test[:,0],negative_test[:,1], label="Negative Test",color='r',marker='o')
    plt.legend()
    plt.show()
```

(a.1.8)

```
def split_by_correct_prediction(X1,X2,y,y_pred):
    """ Splits data by correct / incorrect prediction

    """

    X1correct = []
    X2correct = []

    X1incorrect = []
    X2incorrect = []

    # iterate over both
    for (x1i, x2i, yi, yi_pred) in zip(X1, X2, y, y_pred):

        if(yi == yi_pred):
            # if positive add to positive array
            X1correct.append(x1i)
            X2correct.append(x2i)
        else:
            # if negative add to positive array
            X1incorrect.append(x1i)
            X2incorrect.append(x2i)

    Xcorrect = np.column_stack((X1correct,X2correct))
    Xincorrect = np.column_stack((X1incorrect,X2incorrect))

    return Xcorrect, Xincorrect
```

(a.1.9)

```
1 reference
def plot_correct_incorrect_graph(Xcorrect,Xincorrect,title):
    plt.title(title)
    plt.xlabel('first training feature')
    plt.ylabel('second training feature')
    plt.scatter(Xcorrect[:,0],Xcorrect[:,1], label="Correct predictions",color='g')
    plt.scatter(Xincorrect[:,0],Xincorrect[:,1], label="Incorrect Predictions",color='r')
    plt.legend()
    plt.show()
```

(b.1.1)

```
neighbours = [1,2,3,4,5,6,7,8,9,10]
std_devs = []
means = []
for n in neighbours:
    kf = KFold(n_splits=5)
    mse_estimates = []
    for train, test in kf.split(X):
        model_kNN_ni = KNeighborsClassifier(n_neighbors=n,weights="uniform").fit(X[train],y[train])
        y_pred_i = model_kNN_ni.predict(X[test])
        mse_estimates.append(mean_squared_error(y_pred_i, y[test]))

    means.append(np.mean(mse_estimates))
    std_devs.append(np.std(mse_estimates))

print(means)
print(std_devs)
```

(b.1.2)

```
plt.errorbar(neighbours,means,yerr=std_devs,linewidth=3,capsize=5)
plt.title('kNN for Different Neighbours')
plt.xlabel('Neighbours');
plt.ylabel('Mean/STD dev value')
plt.show()
```

(b.1.3)

```
polynomial_features = [1,2,3,4,5]
std_devs = []
means = []
for degree in polynomial_features:
    n=5
    kf = KFold(n_splits=5)
    mse_estimates = []
    for train, test in kf.split(X):
        model_kNN_ni = KNeighborsClassifier(n_neighbors=n,weights="uniform").fit(poly_transform.fit_transform(X[train]),y[train])
        y_pred_i = model_kNN_ni.predict(poly_transform.fit_transform(X[test]))
        mse_estimates.append(mean_squared_error(y_pred_i, y[test]))

    means.append(np.mean(mse_estimates))
    std_devs.append(np.std(mse_estimates))

print(means)
print(std_devs)
```

(b.1.4)

```
plt.errorbar(polynomial_features,means,yerr=std_devs,linewidth=3,capsize=5)
plt.title('kNN for Different Poly Features')
plt.xlabel('Poly features');
plt.ylabel('Mean/STD dev value')
plt.show()
```

(c.1.1)

```
model_kNN_5 = KNeighborsClassifier(n_neighbors=5,weights="uniform").fit(X_train,y_train)
y_pred_kNN = model_kNN_5.predict(X_test)
Xcorrect, Xincorrect = split_by_correct_prediction(X_test[:,0],X_test[:,1],y_test, y_pred_k
plot_correct_incorrect_graph(Xcorrect, Xincorrect, 'kNN k=5')
print('CONFUSION MATRIX FOR KNN N=5')
print(confusion_matrix(y_test, y_pred_kNN))

poly_transform = PolynomialFeatures(degree=2)
model_logistic_c_1 = LogisticRegression(penalty="l2",C=1)
model_logistic_c_1.fit(poly_transform.fit_transform(X_train),y_train)
y_pred_logistic = model_logistic_c_1.predict(poly_transform.fit_transform(X_test))
print('CONFUSION MATRIX FOR LOGISTIC C=1 W/ 2 FEATURES')
print(confusion_matrix(y_test, y_pred_logistic))

# Compare against baseline that always predicts most common case
most_common_case = get_most_common_case(y)

print('most common case : ', most_common_case)

# fill with most common case
baseline_prediction = np.ones(len(y_test))

# Show the Confusion Matrix
print(confusion_matrix(y_test, baseline_prediction))
```

(d.1.1)

```
def get_most_common_case(y):
    sum = np.sum(y)
    if sum > 0:
        return 1
    if sum < 0:
        return -1
    return 0
```

(d.1.2)

```
fpr,tpr, threshold = roc_curve(y_test,model_logistic_c_1.decision_function(poly_transform.fit_transform(X_test)))
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.plot(1,1, label='Most Frequent',marker='o')
plt.plot(0.5,0.5, label='Random',marker='o')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for logistic')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.show()

y_score = model_kNN_5.predict_proba(X_test)
fpr,tpr, threshold = roc_curve(y_test,y_score[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.plot(1,1, label='Most Frequent',marker='o')
plt.plot(0.5,0.5, label='Random',marker='o')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve for kNN')
plt.show()
```

(d.1.3)

```
fpr,tpr, threshold = roc_curve(y_test,model_logistic_c_1.decision_function(poly_transform.fit_transform(X_test)))
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC Logistic = %0.2f' % roc_auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for logistic')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')

y_score = model_kNN_5.predict_proba(X_test)
fpr,tpr, threshold = roc_curve(y_test,y_score[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'r', label = 'AUC kNN= %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.plot(1,1, label='Most Frequent',marker='o')
plt.plot(0.5,0.5, label='Random',marker='o')
plt.title('Both ROC curves')
plt.show()
```