**Machine Learning – Week 4 Assignment**

The dataset I am using has the following first line: # id:5--5--5-0

1. a)

Chart, scatter chart, qr code

Description automatically generated

The scatter graph for the first dataset I generated can be seen below. The green markers were for negative value of y (-1) and red for positive values of y (1). I used a logistic regression model passing it the ‘l2’ penalty parameter to specify the use of the L2 penalty.

First we must determine which P (Order Polynomial) value to use. I generated cross validation plots to do so, for polynomials 2 until 5 to see its effect on the accuracy. I chose values for P above 1 as we do not want a linear model. I also did not choose too high a degree to avoid overfitting. The different values I used to test for C were 0.1, 1, 10, 50, 100 in order to get a good spread. I used cross validation to obtain an a good estimate for the best P value to use. Scikit provides a helper function cross\_val\_score which allows me to perform cross validation on my model. This function uses the K-fold strategy and I used 5 splits passing 5 to the cv parameter. I computed F1 accuracy and precision scores from this function and graphed them against values of C for each polynomial factor. F1 scores takes into account the precision and recall scores. The graphs can be seen below.

Chart, line chart, box and whisker chart

Description automatically generated

Chart, line chart

Description automatically generated

The graphs for polynomial orders 3, 4 and 5 were all similar to these graphs and can all be seen if you run my code. We can see immediately that values for C higher than 10 began to overfit the data as the precision and F1 accuracy begins to lower slightly past this point. This helped me decide that a polynomial of order 2 would be suitable for this model. The use of higher degrees may yield more accurate result but there would be a risk of overfitting.

Next is to decide a value for C. The highest F1 accuracy and precision scores for C seemed to be between 1 and 10 , so I will graph some values between these to decide which is best. I used C even C values between 1 and 10. The graph generated can be seen below.

Chart, line chart

Description automatically generated

Here we can see that for the polynomial of order 2, a value for C between 4 and 6 seems to fit with the best in terms of F1 accuracy. In terms of precision a value between 2 and 4 would fit best. I will take the average of these two ranges and use C = 4 as the best fit.

So, in conclusion the best fit I found is using polynomial order 2 and a C value of 4. If we look at the training data plotted using the predictions from my model, we can see that a value of P = 1 is too low as it cannot capture the quadratic relationship between the Y values within the data. Chart, scatter chart, qr code

Description automatically generatedChart, scatter chart, qr code

Description automatically generated

Chart, scatter chart, qr code

Description automatically generated

We can also see that the result for P = 2 and P = 3 are very similar. This backs our decision to choose P = 2 as the best value as it fits the model appropriately without the risk of overfitting.

1. There is no need to use P with a KNN classifier so I removed the augmentation of my features for this method. I use cross validation to determine a good value to use for K. We can expect that if we use too small a value for K we can expect overfitting to occur as there are too little neighbours being used by each datapoint, and vice versa for choosing too high a value for K, resulting in underfitting as too many neighbours are being used which will reduce our precision. Like before I calculate a precision and F1 accuracy score for different values of K using 5-fold splits. The graph generated can be found below.

Chart, line chart

Description automatically generated

It seems like for even K values, the precision is increased. The F1 accuracy increases slowly with each increment of K, more so on odd values. This is helped by the split vote problem with choosing even numbers for K which causes some datapoints not to contribute to the model. The highest precision was obtained for K = 2, however the F1 accuracy is a lower than for higher values of K here. A good value which produces a high score for both metrics is K = 4. The training data graphed for K = 4 can be found below.

Chart, scatter chart, qr code

Description automatically generated

This is very similar to our graph from part a) using our optimal values for P = 2 and C = 4. This seems to fit the data well as we can see the quadratic relationship between the positive and negative outputs clearly.

1. It was decided that for my Logistic Regression model, the best fit was using a polynomial of order 2 and a value of 4 for C. I will use these parameters to obtain my model confusion matrix. The confusion matrix which I obtained using scikits ConfusionMatrixDisplay helper can be seen below.

Chart, treemap chart

Description automatically generated

This model shows my model predicted 784 true positive, 16 false positive, 22 false negative and 366 false positive outputs. This was an effective model as we can see only 38 predictions were incorrect.

Text

Description automatically generated

I used these formulas from the notes to calculate important statistics on my model.

Text

Description automatically generated

The accuracy, true positive and false positive rates, and precision calculated can be found to the left. They indicate this is a good model as the precision, accuracy and true positive rate is high and the false positive rate is low.

For my KNN model we decided a value of 4 was best suited for K. Using the same method as before I generated the confusion matrix which can be seen below for K = 4.

Chart, treemap chart

Description automatically generated

This model shows my model predicted 797 true positive, 3 false positive, 26 false negative and 362 false positive outputs. This was also an effective model as we can see only 29 predictions were incorrect.

Text

Description automatically generated

The accuracy, true positive and false positive rates, and precision calculated can be found to the left. Again, these indicate it is a well-trained model as the accuracy, precision and true positive rate is high, with a low false positive rate.

Chart, treemap chart

Description automatically generated

I then used sklearns DummyClassifier helper, passing ‘uniform’ as the strategy parameter. This generates predictions uniformly and at random. The confusion matrix generated can be found to the left.

Text

Description automatically generated

This classifier worked a lot worse than our KNN and Logistic regression models as we can see it generated over 500 incorrect predictions. This is expected with a random classifier. The accuracy, precision and rates can be found to the left.

These values are much lower than our previous models as expected.

1. Once again I chose to use the optimum values for P = 2 and C = 4 which I determined in part a) for the ROC curve I generate using the Logistic Regression model. I used the optimum value of K = 4 for the KNN model also. I generated the false and true positive rates for the ROC curves using scikits helper function roc\_curve before generating a graph using these values as the X and Y axes. We want our models to get as close to the point X = 0, Y = 1 in the graph as this indicates a well-trained model. If the ROC curve goes below the baseline classifier we can consider the model a failure. The graph generated can be found below.

Chart, line chart

Description automatically generated

We can see our Logistic Regression and KNN models both perform very well, with the K nearest neighbours model maybe edging the Logistic regression model slightly.

1. Using the values obtained from parts c) and d) the Logistic Regression and KNN classifiers are clearly more effective than the baseline classifier that I used (Random Classifier). We can take a look at the Accuracy, True Positive and True Negative rates and the precision calculated in part c) for each model. For the Logistic regression model, the accuracy was around 96.8%, true positive rate was around 94.3%, false positive rates was around 2% and precision was around 95.8%. This is good performance for a model, as stated before.

Our KNN model had an accuracy of around 97.5%, true positive rate of around 93.2%, false positive rate of around 0.37% and precision of around 99%. Here we can see that our KNN model edges the performance of the Logistic regression model slightly. It has a much lower false positive rate of 0.37% compared to 2%. It also has a higher precision and accuracy. It does have a lower true positive rate, but it is more accurate with negatives compared to the Logistic regression model.

As expected the random baseline classifier is outperformed immensely by both the Logistic regression and KNN models. Its accuracy was around 50%, true positive rate around 48%, false positive rate of around 48% and precision was around 32%. These figures compared to our trained models are laughable.

The ROC curves from part d) backup my choice. Both the Logistic Regression and KNN models are acceptable to use for this dataset, with KNN maybe being slightly more advantageous to use.

1. a)

Chart, scatter chart

Description automatically generated

The scatter graph for the second dataset I generated can be seen below. The green markers were for negative value of y (-1) and red for positive values of y (1). I used a logistic regression model passing it the ‘l2’ penalty parameter to specify the use of the L2 penalty.

First we must determine which P (Order Polynomial) value to use. I generated cross validation plots again to do so, for polynomials 2 until 5 to see its effect on the accuracy. I chose values for P above 1 as we do not want a linear model as it won’t fit the data appropriately. I also did not choose too high a degree to avoid overfitting. The different values I used to test for C were 0.1, 1, 10, 50, 100 in order to get a good spread. I used cross validation to obtain an a good estimate for the best P value to use. I used the same method for cross validation as before, using 5-fold splits and computing F1 accuracy and precision scores for each polynomial factor and against different values of C. The graphs can be seen below.

Graphical user interface, application

Description automatically generated

Graphical user interface, chart

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

As P is incremented, we can notice that the accuracy lowers slightly and the precision improves. The data seems largely random here as it is hard to spot a clear relationship within it. The graphs for polynomial orders 4 and 5 were all similar to these graphs and can all be seen if you run my code. As P seems to not have a great effect as it increments, I will choose P = 2 as the optimum as it seems to hold the highest F1 accuracy and precision together. However, this choice is tough to make as the effect of P on these graphs seems minimal.

We do not want to overfit the data so I will not choose a high value for C. I will looks at the plot for values of C between 1 and 10 as before. The use of higher degrees may yield more accurate result but there would be a risk of overfitting. The graph for P = 2 below shows that the precison and F1 score are at their peak when C = 1. They trend downwards until C = 2 then seem to continue in a straight line, unaffected by C changing.

Chart

Description automatically generated

The highest F1 accuracy and precision scores for C seemed when C = 1, but again this is hard to determine as the relationship is not clear to see. It seems varying C does not have a huge effect on the model. The dataset seems inappropriate for our model. So, in conclusion the best fit I found is using polynomial order 2 and a C value of 1.

Chart, scatter chart, qr code

Description automatically generated

If we look at a graph of the predictions we can see that the model is only predicting positive outputs.

If we look at the training data plotted using the predictions from my model, we can see that a negative values only begin to be predicted when P is 4 or greater.

Chart, scatter chart

Description automatically generated Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated Chart, scatter chart

Description automatically generated

Something that is noticable here is that once the model began predicting negative outputs, if we increased C it would increase the number of negative outputs as can be seen above for Polynomial: 5.

Chart, scatter chart

Description automatically generated Chart, scatter chart

Description automatically generated

Chart

Description automatically generated

The problem with this is that the accuracy and precision isn’t altered by the newly added negative outputs as can be seen in the cross validation graph below. This means that it is hard to say that using a higher value for P will result in a better fit for our model as both have their flaws.

To conclude, this dataset seems too noisy to create a logistic regression model which fits the data well as we have conflicting and vague results.

1. Again, there is no need to use P with a KNN classifier so I removed the augmentation of my features for this method. I use the same cross validation method to determine a good value to use for K. Like before I calculate a precision and F1 accuracy score for different values of K using 5-fold splits. The graph generated can be found below.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

If we use too high a value for K we can see the gap between the precision and F1 accuracy increases to a level we do not want. We want to strike a balance between both performance metrics.

Again we can see that for even K values, precision is slightly increased. However the F1 accuracy fluctuates conversely, increasing for odd values of K. This again is helped by the split vote problem as mentioned before. The accuracy compared to the model in part a) is a lot lower, with the precision around the same mark.

Chart, scatter chart

Description automatically generated

The highest precision was obtained for K = 4, however the F1 accuracy is a lower than for higher values of K here. A good value which produces a high score for both metrics is K = 5. The training data graphed for K = 5 can be found below.

This is very different to our graph from part a) using logistic regression. This seems to fit the data better as we can see the negative predicted inputs immediately.

1. It was difficult to determine for my Logistic Regression model, which parameter values yielded the best fit. So I will experiment with a few values here also. I obtained the confusion matrix using the same method as before.

Chart, treemap chart

Description automatically generated Text

Description automatically generated

Chart, treemap chart

Description automatically generated Text

Description automatically generated

As we increase P we can begin to see the false positive rate drop, as negative outputs begin to be predicted. At the same time however the true positive rate drops. Accuracy and precision all increase slightly. This high true positive rate is only due to the fact that it is mostly positive outputs predicted with a very small amount of negatives predicted, as can be seen with the high false positive rate. The model is still ineffective with 334 incorrect predictions for P = 8.

For my KNN model we decided a value of 5 was best suited for K. Using the same method as before I generated the confusion matrix which can be seen below for K = 5.

Chart, treemap chart

Description automatically generatedText

Description automatically generated

This graph shows my model predicted 257 true positive, 86 false positive, 165 false negative and 483 false positive outputs. The model is more effective than the Logistic regression model with 271 incorrect predictions for K = 4. This is however still a large amount of inaccurate predictions.

The accuracy, true positive and precision for the KNN model is however a lot better than the Logistic regression. There is a high false positive rate but a lot lower than the logistic regression’s rate.

I used the same method as before to generate the baseline random classifier. The confusion matrix generated can be found below.

Chart, treemap chart

Description automatically generatedText

Description automatically generated

This classifier did not perform as badly as with the first dataset, compared to KNN and Logistic Regression. We can see it generated 489 incorrect predictions which is a sharp improvement compared to the first dataset. It is expected with a random classifier to have lots of incorrect predictions. The accuracy, precision and rates can be found below.

As it is randomly predicting between 2 values (-1 or 1) it is expected to be around 50% accurate as can be seen from the values above. The precision of this classifier is very close to the logistic regressions precision which indicates the logistic regression does not give a good fit for this dataset.

1. As the accuracy, precision, true positive rate began to increase as I used higher values for P in my confusion matrix results in part c), I will use a higher value for P to generate this ROC curve. I used the optimum value of K = 5 for the KNN model also. I used the same method to generate the ROC curve as before. Once again, we want our models to get as close to the point X = 0, Y = 1 in the graph as this indicates a well-trained model. If the ROC curve goes below the baseline classifier we can consider the model a failure. The graph generated can be found below.

Chart, line chart

Description automatically generated

We can see our Logistic Regression model does not perform well with this dataset. It stays close to the baseline classifier which is random. The KNN model performs slightly better but not being a great fit for the data although outperforming the other models. The baseline random classifier performs as expected.

1. Using the information we obtained from part c) and d) It is clear that the best performing model with this dataset was the KNN model. The baseline classifier was similar performance wise as the Logistic Regression model which indicates that the Logistic Regression model is not suitable to use for this dataset. Again we can compare the Accuracy, True Positive and True Negative rates and the precision calculated in part c) for each model.

For the logistic regression model the best values we obtained were around 66% accuracy, 98% true positive rate, 93% false positive rate and 66% precision. Again I must mention these are not impressive results for the Logistic classifier. Our KNN model had an accuracy of 75%, true positive rate of 74%, a false negative rate of 25% and a precision rate of around 85%. This is much better performance for a model compared to the Logistic regression however we cannot say it is a great fit for the data. It has a higher accuracy by around 10% and a higher precision by around 20%. The false positive rate is around 20% lower but the true positive rate is 24% lower. However, this is only because the Logistic regression model rarely predicts negative outputs. The random baseline classifier isn’t as outperformed with this dataset as it was with the first dataset. It had an accuracy as expected of around 50%, true positive rate of around 50%, false positive rate of around 50% and precision around 65%. Its precision is as good as the logistic regression model.

Based on these results its hard to say that any of these models would be a suitable fit for this dataset as they all produce lots of inaccuracies. KNN is the best fit out of the choices given but it does not yield good results. This is mainly because the data is too noisy. We would need a model which predicts which a much higher accuracy and precision for it to be a good fit.

**Appendix:**

import numpy as np# id:5--5--5-0

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import cross\_val\_score

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.dummy import DummyClassifier

from sklearn.metrics import roc\_curve

dfi = pd.read\_csv("week4i.csv")

X1i = dfi.iloc[:,0]

X2i = dfi.iloc[:,1]

Xi = np.column\_stack((X1i,X2i))

yi = dfi.iloc[:,2]

dfii = pd.read\_csv("week4ii.csv")

X1ii = dfii.iloc[:,0]

X2ii = dfii.iloc[:,1]

Xii = np.column\_stack((X1ii,X2ii))

yii = dfii.iloc[:,2]

def trainingDataScatterPlot(features, output):

plt.figure()

plt.scatter(features[output > 0, 0], features[output > 0, 1], color="red")

plt.scatter(features[output < 0, 0], features[output < 0, 1], color="green")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend(["Positive Output","Negative Output"])

plt.show()

def predictionsScatterPlot(X, yPredictions, title):

plt.figure()

plt.scatter(X[yPredictions > 0, 0], X[yPredictions > 0, 1], color="red")

plt.scatter(X[yPredictions < 0, 0], X[yPredictions < 0, 1], color="green")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.title(title)

plt.legend(["Positive Output","Negative Output"])

plt.show()

def crossValPlot(CValues, prec\_mean\_err, prec\_std\_err, f1\_mean\_err, f1\_std\_err, title):

plt.figure()

plt.errorbar(CValues, prec\_mean\_err, label='Precision Score', ecolor='red', color='blue', yerr=prec\_std\_err, linewidth=2)

plt.errorbar(CValues, f1\_mean\_err, label='F1 Score', ecolor='g', color='orange', yerr=f1\_std\_err, linewidth=2)

plt.xlabel("C")

plt.xlim(0,1000)

# plt.xlim(0,100)

# plt.xlim(0,10)

plt.ylabel("F1 Accuracy & Precision")

plt.legend()

plt.title(title)

plt.show()

def KNNPlot(KValues, prec\_mean\_err, prec\_std\_err, f1\_mean\_err, f1\_std\_err, title):

plt.figure()

plt.errorbar(KValues, prec\_mean\_err, label='Precision Score', ecolor='red', color='blue', yerr=prec\_std\_err, linewidth=2)

plt.errorbar(KValues, f1\_mean\_err, label='F1 Score', ecolor='g', color='orange', yerr=f1\_std\_err, linewidth=2)

plt.xlabel("K Value")

plt.xlim(1,10)

plt.ylabel("F1 Accuracy & Precision")

plt.legend()

plt.title(title)

plt.show()

def crossValLReg(features, output):

# Ci = [1, 2, 4, 6, 8, 10]

Ci = [1, 10, 50, 100, 1000]

# Ci = [100]

for polyFeatures in [2, 3, 4, 5, 6]:

prec\_mean\_err = []; prec\_std\_err = []

f1\_mean\_err = []; f1\_std\_err = []

poly = PolynomialFeatures(polyFeatures)

result = poly.fit\_transform(features)

for C in Ci:

model = LogisticRegression(max\_iter=1000, C=C, penalty="l2")

model.fit(result, output)

predictions = model.predict(result)

title = "Polynomial: " + str(polyFeatures) + ", C: " + str(C)

predictionsScatterPlot(features, predictions, title)

# Cross Validation K-Fold

f1Scores = cross\_val\_score(model, result, output, scoring='f1', cv=5) # f1 scores

precScores = cross\_val\_score(model, result, output, scoring='precision', cv=5) # precision scores

prec\_std\_err.append(np.array(precScores).std())

prec\_mean\_err.append(np.array(precScores).mean())

f1\_std\_err.append(np.array(f1Scores).std())

f1\_mean\_err.append(np.array(f1Scores).mean())

tn, fp, fn, tp = confusion\_matrix(output, predictions).ravel()

accuracy = (tn + tp)/(tn + tp + fn + fp)

truePosRate = (tp)/(tp + fn)

falsePosRate = (fp)/(tn + fp)

precision = (tp)/(tp + fp)

print("Accuracy: \n", accuracy)

print("True Positive Rate: \n", truePosRate)

print("False Positive Rate: \n", falsePosRate)

print("Precision: \n", precision)

ConfusionMatrixDisplay.from\_predictions(output, predictions)

plt.title("Polynomial: " + str(polyFeatures) + ", C: " + str(C) + " Confusion Matrix")

plt.show()

crossValPlot(Ci, prec\_mean\_err, prec\_std\_err, f1\_mean\_err, f1\_std\_err, "Cross Validation Precision & F1 Accuracy: P=" + str(polyFeatures))

def crossValKNN(features, output):

Ki = [3, 4, 5, 6, 7, 8, 9, 10]

prec\_mean\_err = []; prec\_std\_err = []

f1\_mean\_err = []; f1\_std\_err = []

for K in Ki:

model = KNeighborsClassifier(n\_neighbors=K, weights='uniform')

model.fit(features, output)

predictions = model.predict(features)

title = "KNN Predictions, K: " + str(K)

predictionsScatterPlot(features, predictions, title)

f1Scores = cross\_val\_score(model, features, output, scoring='f1', cv=5) # f1 scores

precScores = cross\_val\_score(model, features, output, scoring='precision', cv=5) # precision scores

prec\_std\_err.append(np.array(precScores).std())

prec\_mean\_err.append(np.array(precScores).mean())

f1\_std\_err.append(np.array(f1Scores).std())

f1\_mean\_err.append(np.array(f1Scores).mean())

tn, fp, fn, tp = confusion\_matrix(output, predictions).ravel()

accuracy = (tn + tp)/(tn + tp + fn + fp)

truePosRate = (tp)/(tp + fn)

falsePosRate = (fp)/(tn + fp)

precision = (tp)/(tp + fp)

print("Accuracy: \n", accuracy)

print("True Positive Rate: \n", truePosRate)

print("False Positive Rate: \n", falsePosRate)

print("Precision: \n", precision)

ConfusionMatrixDisplay.from\_predictions(output, predictions)

plt.title("K: " + str(K) + " Confusion Matrix")

plt.show()

KNNPlot(Ki, prec\_mean\_err, prec\_std\_err, f1\_mean\_err, f1\_std\_err, "KNN Scores")

def randClassifier(features, output):

dummy\_clf = DummyClassifier(strategy="uniform")

dummy\_clf.fit(features, output)

predictions = dummy\_clf.predict(features)

tn, fp, fn, tp = confusion\_matrix(output, predictions).ravel()

accuracy = (tn + tp)/(tn + tp + fn + fp)

truePosRate = (tp)/(tp + fn)

falsePosRate = (fp)/(tn + fp)

precision = (tp)/(tp + fp)

print("Accuracy: \n", accuracy)

print("True Positive Rate: \n", truePosRate)

print("False Positive Rate: \n", falsePosRate)

print("Precision: \n", precision)

ConfusionMatrixDisplay.from\_predictions(output, predictions)

plt.title("Random Classifier Confusion Matrix")

plt.show()

def ROCPlot(features, output):

# poly = PolynomialFeatures(2) # P = 2 optimum

poly = PolynomialFeatures(8) # P = 8 optimum

result = poly.fit\_transform(features)

model = LogisticRegression(max\_iter=1000, C=4, penalty="l2") # C = 4 optimum

model.fit(result, output)

LRegPredictions = model.predict(result)

fprLReg, tprLReg, \_LReg = roc\_curve(output, LRegPredictions)

# model = KNeighborsClassifier(n\_neighbors=4, weights='uniform') # K = 4 optimum

model = KNeighborsClassifier(n\_neighbors=5, weights='uniform') # K = 5 optimum

model.fit(features, output)

KnnPredictions = model.predict(features)

fprKnn, tprKnn, \_Knn = roc\_curve(output, KnnPredictions)

dummy\_clf = DummyClassifier(strategy="uniform")

dummy\_clf.fit(features, output)

randPredictions = dummy\_clf.predict(features)

fprRand, tprRand, \_Rand = roc\_curve(output, randPredictions)

plt.plot(fprLReg, tprLReg, color='g')

plt.plot(fprKnn, tprKnn, color='b')

plt.plot(fprRand, tprRand, color='orange', linestyle='--')

plt.xlabel("False positive Rate")

plt.ylabel("True positive Rate")

plt.title("ROC curves")

plt.legend(["Logistic Regression", "K - Nearest Neighbors", "Random Classifier"])

plt.show()

# trainingDataScatterPlot(Xi, yi) # training data scatter

crossValLReg(Xi, yi)

# crossValKNN(Xi, yi)

# randClassifier(Xi, yi)

# ROCPlot(Xi, yi)

# trainingDataScatterPlot(Xii, yii) # training data scatter

# crossValLReg(Xii, yii)

# crossValKNN(Xii, yii)

# randClassifier(Xii, yii)

# ROCPlot(Xii, yii)