CT475 Assignment Two

Name: Michael Murphy
Student Number: 15300856
Class: 4BS2 – 4th Year Science (Maths & Computer Science)

1. Outline of assignment

For this assignment, we were required to build on the work done previously, when we researched machine learning packages, and chose two classification algorithms that we felt would appropriately deal with a given classification task. My choice of machine learning package was *scikit-learn*[1], and the classification algorithms I selected were the ***k-Nearest Neighbours***[2] and ***Support Vector Machines***[3] algorithms.

scikit-learn has built-in methods for constructing ROC curves when provided with a given set of probability scores for the 'positive' class, which I discuss in detail in section 2. Hence, I was able to obtain a number of ROC curves for both algorithms, which provided a new means of comparing them. These are included in section 3. I made three observations about the two curves, outlined in section 4.

2. Methodology and assumptions

scikit-learn's ability to deal with ROC curves comes in the form of the *roc_curve*[4] function. The parameters that were important for this dataset were *y_true, y_score,* and *pos_label.*

- y_true: the array of true binary labels. This array was simply a subset of the *autoimmune_target* array, which was obtained with the help of a *StratifiedKFold* (more below).
- y_score: the array of probability scores for the 'positive' label. This was obtained with the help of the *predict_proba* method (more below).
- pos_label: since the class labels in our dataset were not simple labels like {-1,1} or {0,1}, this had to specified so that *scikit-learn* knew what label corresponded to the positive class. In our case, this label was "positive". The function would then automatically assume that any labels other than "positive" would correspond to the negative class, since the function can only work with binary classification tasks.

The function returns the false positive and true positive rates as separate arrays, as well as a threshold array. As explained in the documentation, the value at *thresholds[i]* is used to calculate false positive and true positive rates, which are stored at *fpr[i]* and *tpr[i]* respectively. Representing the ROC curves visually was then a simple matter of plotting the values of *tpr* on the y-axis, and *fpr* on the x-axis.

As required, I plotted my curves using a held-out test set which was 1/3 of the overall dataset size. I used a *StratifiedKFold*[5] to achieve this. Specifically, I used the *split*[6] method, the purpose of which is to "generate indices to split data into training and test set". This works by splitting the data set into n folds and using 1/n of the dataset as a test set, while using the rest as a training set. Each sample gets used once for testing, and (n-1) times for training [See Figure 1].

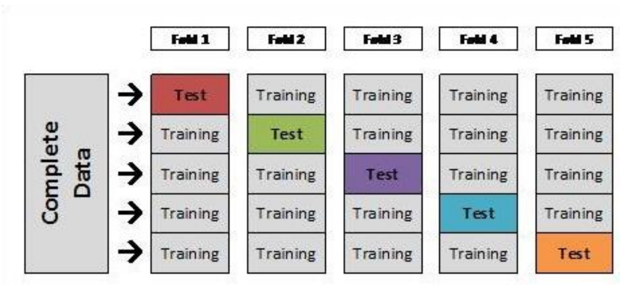


Figure 1[7]: Visual representation of k fold generation

CT475 Assignment Two

Name:	*Student Number:*	*Class:*
Michael Murphy	15300856	4BS2 – 4th Year Science (Maths & Computer Science)

2. *Methodology and assumptions (continued)*

So, by setting *n_splits* to 3, I generated 3 folds, each with a test set that was 1/3 of the overall dataset size. For each fold, I trained the model with the given training set by using the respective *fit* methods of kNN[8] and SVM[9], and then calculated the class probability scores of the samples in the training set using the *predict_proba* method, which again operates slightly differently for kNN[10] than for SVM[11]. Then, for each of the 3 folds, I was able to construct a different ROC curve.

Having multiple curves certainly offered a better idea of how the algorithm performed on average. Also, for each curve, I calculated its AUROC score, again with the help of one of the package functions; the *auc*[12] function. Each curve's AUROC score is reported in the legends of their graphs, shown in section 3.

The value of using a StratifiedKFold is the fact that for each fold, *the test set has approximately the same amount of positively labelled samples as every other test set.* For the *autoimmune* dataset, each fold's test set had either 39 or 40 positively labelled samples, while having 86 negatively labelled samples. This fact is also reported in the legends of the graphs.

This is useful because it ensures that each fold is as good a representative of the complete dataset as every other fold, which in theory should yield ROC curves that better reflect the future performance of the algorithm. Of course, this assumes that our current data is representative of any future data that needs to be classified; if it is not, the ROC curve is a somewhat redundant measure of future performance.

To get an even better visualisation of how the two algorithms compared to one another, I also obtained mean ROC curves. To do this, I created a fixed *mean_fpr* array, with 20 values between 0 and 1, evenly dispersed. Then, for each fold, I used the values in the *tpr* and *fpr* arrays to **interpolate** what the true positive rates would be for each value in the *mean_fpr* array**,** using Numpy's *interp*[13] function. I stored these values in separate arrays for each fold.

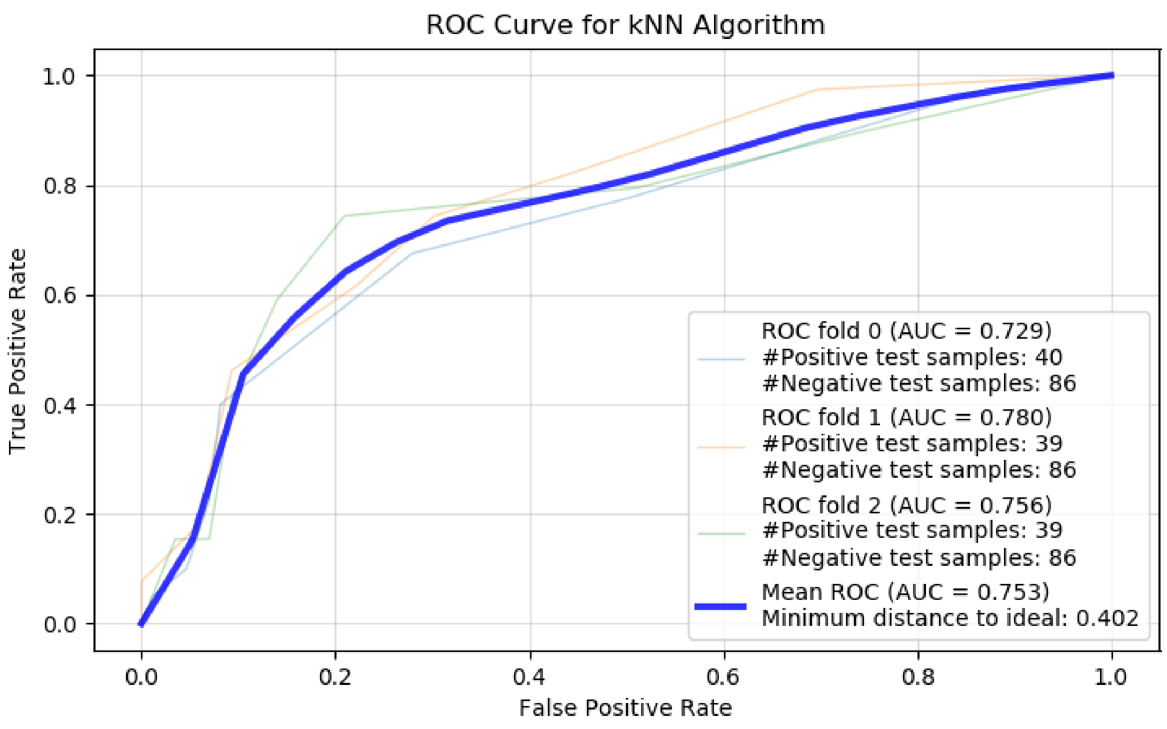
When all folds had been looped over, I created a *mean_tpr* array by taking the mean of the values at each index of the individual, interpolated *tpr* arrays, using the Numpy *mean*[14] function. Then I simply used these *mean_tpr* and *mean_fpr* arrays to plot a mean ROC curve for each algorithm, while also reporting the AUROC score for these curves, to have a single number comparison of the two methods.

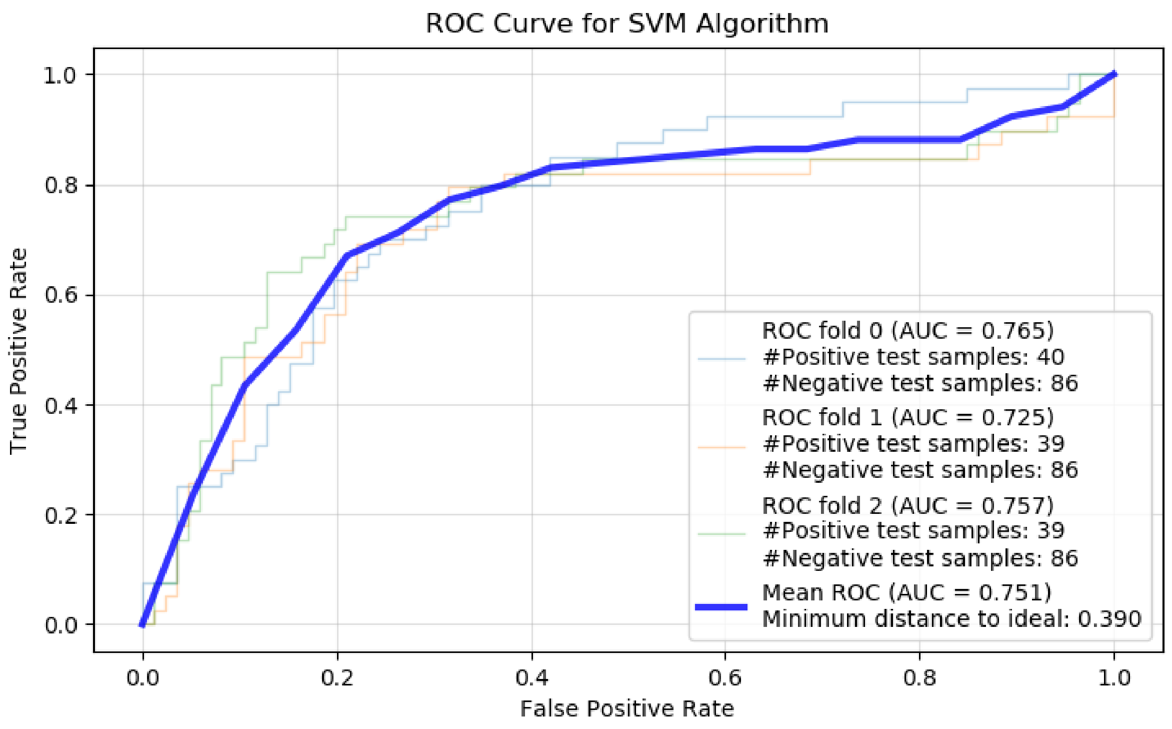
The final step in my procedure was carried out in order to determine how close each mean curve got to the ideal; that is, where false positive rate equals 0, and true positive rate equals 1, in the top-left corner of the graph. To do this, I simply calculated Euclidean distance from the point (0,1) to each point on the curve, and found the minimum distance across each point on the curve. I did this for both curves, and reported the minimum distance value in the legend of each graph.

CT475 Assignment Two

Name:	*Student Number:*	*Class:*
Michael Murphy	15300856	4BS2 – 4th Year Science (Maths & Computer Science)

3. ROC Curves





CT475 Assignment Two

Name:	*Student Number:*	*Class:*
Michael Murphy	15300856	4BS2 – 4th Year Science (Maths & Computer Science)

4. *Observations*

Observation 1: By examining the AUROC scores of the mean ROC curves for the two algorithms, we can see that kNN outperforms SVM by a very slight margin (0.753 > 0.751). This is in agreement with my observations from the first assignment, where, in their optimal setups, kNN and SVM obtained 10-fold cross validation scores of 0.7636 and 0.7424, respectively. As mentioned, this is only a marginal difference, and it would certainly be fair to say that, for this dataset, the two algorithms performed similarly. *As noted*[15] in the first assignment, for a dataset with which has neither a very large or small number of dimensions or samples, it is reasonable to expect that these two algorithms should perform similarly.

Observation 2: We expect that, the closer the curve is to the ideal (fpr = 0, tpr = 1), the higher the overall accuracy of the test[16]. Interestingly, the mean ROC curve for the SVM algorithm actually gets closer to the ideal than the kNN algorithm (0.39 < 0.402). This would seem to disagree with the earlier results, inferred from cross-validation and AUROC scores, that suggest that kNN is the more accurate test. However, when the margin between the two algorithms appears to be so small, it is perhaps unsurprising that there is slight disagreement between different performance measures.

Observation 3: Although the mean ROC curves of the two graphs are both smooth (since the points on the curve were interpolated), we can see that the curves of the individual kNN folds are also smooth, while those of the SVM folds actually have a stepped shape. This would indicate that, for the SVM algorithm, the values in the thresholds array returned by the *roc_curve* function are being selected in such a way that ***either** the true positive rate **or** the false positive rate* increases from the rates at the previous threshold value. Meanwhile, for the kNN algorithm, each sequential threshold value sees an increase in ***both** the true and false positive rates*. It's clear from the graphs that the *roc_curve* function generated far more threshold values for SVM than kNN, which is possibly the cause of this shape.

Bibliography

[1] – Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.
[2] – *Nearest Neighors*, scikit-learn documentation (v0.20.0)
[3] – *Support Vector Machines*, scikit-learn documentation (v0.20.0)
[4] – *roc_curve*, scikit-learn documentation (v0.20.0)
[5] – *StratifiedKFold*, scikit-learn documentation (v0.20.0)
[6] – *split*, scikit-learn documentation (v0.20.0)
[7] – *Cross-Validation for Genomic Prediction in SVS*, Golden Helix, Inc
[8] – *kNN.fit*, scikit-learn documentation (v0.20.0)
[9] – *SVM.fit*, scikit-learn documentation (v0.20.0)
[10] – *kNN.predict_proba*, scikit-learn documentation (v0.20.0)
[11] – *SVM.predict_proba*, scikit-learn documentation (v0.20.0)
[12] – *auc*, scikit-learn documentation (v0.20.0)
[13] – *interp*, NumPy documentation (v1.15)
[14] – *mean*, NumPy documentation (v1.15)
[15] - *What is better, k-nearest neighbors algorithm or Support Vector Machine classifier?*, quora.org
[16] – *Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine*, Zweig MH, Campbell G (1993), Clinical Chemistry 39:561-577

CT475 Assignment Two

Name:	*Student Number:*	*Class:*
Michael Murphy	15300856	4BS2 – 4th Year Science (Maths & Computer Science)

Code

```
import numpy as np
from scipy import interp
import matplotlib.pyplot as plt
from itertools import cycle
from sklearn import neighbors, svm
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import roc_curve, auc

#change into location of dataset, specify file name
fname = 'autoimmune_transpose.txt'

#use np.genfromtxt to read in training data, and target feature data
autoimmune_data = np.genfromtxt(fname, delimiter='\t', encoding=None,
usecols=np.arange(0,9))
autoimmune_target = np.genfromtxt(fname, delimiter='\t', dtype=None,
encoding=None, usecols=9)

#create models with kNN, checking the the 10-fold cross validation scores for
various values of k
k_scores=[]

for i in range(1,20):
    kNN = neighbors.KNeighborsClassifier(i)
    kNN_scores = cross_val_score(kNN, autoimmune_data, autoimmune_target,
cv=10)
    k_scores.append(kNN_scores.mean())

k_max = max(k_scores)
k_choose = k_scores.index(k_max)+1
print("The highest 10-fold cross validation scores were found for k = ",
k_choose, ", which had a mean score of %.4f." % k_max, sep='')

#create models with SVM, checking the the 10-fold cross validation scores
when gamma is set to 'scale' or 'auto'
s_scores=[]
s_options=['scale','auto']

for i in s_options:
    svc = svm.SVC(gamma=i)
    svc_scores = cross_val_score(svc, autoimmune_data, autoimmune_target,
cv=10)
    s_scores.append(svc_scores.mean())

s_max = max(s_scores)
s_choose = s_options[s_scores.index(s_max)]
print("The highest 10-fold cross validation scores were found when gamma was
set to '", s_choose, "', which had a mean score of %.4f." % s_max, sep='')
```

CT475 Assignment Two

Name:	*Student Number:*	*Class:*
Michael Murphy	15300856	4BS2 – 4th Year Science (Maths & Computer Science)

```
#report which algorithm has higher cross validation score
if k_max >= s_max:
    print("A higher mean cross validation score was obtained by the kNN
algorithm.")
else:
    print("A higher mean cross validation score was obtained by the SVM
algorithm.")

#keep the models that generated the highest cross validation scores for each
algorithm
kNN = neighbors.KNeighborsClassifier(n_neighbors=k_choose)
svc = svm.SVC(gamma=s_choose,probability=True)

#use a StratifiedKFold to generate training and testing sets
kf = StratifiedKFold(n_splits=3)

#we now use the split method of StratifiedKfold to split our dataset into
folds
#with 1/3 of the dataset to be put aside for testing in each fold

num_pos=0 #counts the number of positive samples in the test set of each fold
num_neg=0 #counts the number of negative samples in the test set of each fold
fold = 0 #keeps track of the current ROC fold for labelling the graph

#create arrays to hold the tpr arrays and the auc scores for each fold
ktprs, kaucs = [], []
stprs, saucs = [], []

#generate a linear space between 0 and 1 to use as fpr points; we will
generate tpr points using interpolation
mean_fpr = np.linspace(0, 1, 20)

#loop over each fold
for train, test in kf.split(autoimmune_data, autoimmune_target):
    #count number of positive and negative samples
    for k in test:
        if autoimmune_target[k] == "positive":
            num_pos+=1
        else:
            num_neg+=1

    #train kNN with the given training set, and predict the class probability
scores of the samples in the test set
    kprobas_ = kNN.fit(autoimmune_data[train],
autoimmune_target[train]).predict_proba(autoimmune_data[test])
```

```
    # Compute ROC curve for kNN, store true positive rates for this curve so
that mean tprs can be calculated later
    kfpr, ktpr, kthresholds = roc_curve(y_true=autoimmune_target[test],
y_score=kprobas_[:, 1], pos_label="positive")
    ktprs.append(interp(mean_fpr, kfpr, ktpr)) #interpolate some tpr values
at the values in mean_fpr so that mean tprs can be calculated later
    ktprs[-1][0] = 0.0 #manually set the tpr at (0,0) to zero, in case
interpolation hasn't done this already

    # Compute AUROC scores for kNN, then report this when plotting the kNN
ROC curve for this fold
    roc_auc = auc(kfpr, ktpr)
    kaucs.append(roc_auc) #store the AUROC score for this fold
    plt.figure(1) #figure 1 is kNN curves
    plt.plot(kfpr, ktpr, lw=1, alpha=0.3,
             label="ROC fold %d (AUC = %0.3f) \n#Positive test samples: %d
\n#Negative test samples: %d" % (fold, roc_auc, num_pos, num_neg))

    #repeat all these steps for SVM
    sprobas_ = svc.fit(autoimmune_data[train],
autoimmune_target[train]).predict_proba(autoimmune_data[test])
    sfpr, stpr, sthresholds = roc_curve(y_true=autoimmune_target[test],
y_score=sprobas_[:, 1], pos_label="positive")
    stprs.append(interp(mean_fpr, sfpr, stpr))
    stprs[-1][0] = 0.0
    roc_auc = auc(sfpr, stpr)
    saucs.append(roc_auc)
    plt.figure(2) #figure 2 is SVM curves
    plt.plot(sfpr, stpr, lw=1, alpha=0.3,
             label="ROC fold %d (AUC = %0.3f) \n#Positive test samples: %d
\n#Negative test samples: %d" % (fold, roc_auc, num_pos, num_neg))

    #next ROC fold, reset num_pos & num_neg, increment fold
    num_pos,num_neg=0,0
    fold += 1

#now that we have interpolated the same number of tpr values for each fold,
we can easily find the mean values at each index
#first for kNN
mean_tpr = np.mean(ktprs, axis=0)
mean_auc = auc(mean_fpr, mean_tpr) #also calculate AUROC score for these mean
tpr/fpr values

#find how close to the ideal it gets
dists_to_ideal = []
for i in range(len(mean_tpr)):
      dists_to_ideal.append(np.sqrt((1-mean_tpr[i])**2 + (0-mean_fpr[i])**2))

#plot mean curve
plt.figure(1)
plt.plot(mean_fpr, mean_tpr, color='b', #finally, plot the mean curve
         label="Mean ROC (AUC = %0.3f)" % (mean_auc),
         lw=3, alpha=.8)
```

CT475 Assignment Two

Name: Michael Murphy
Student Number: 15300856
Class: 4BS2 – 4th Year Science (Maths & Computer Science)

```
#then for SVM
mean_tpr = np.mean(stprs, axis=0)
mean_auc = auc(mean_fpr, mean_tpr)
dists_to_ideal = []
for i in range(len(mean_tpr)):
      dists_to_ideal.append(np.sqrt((1-mean_tpr[i])**2 + (0-mean_fpr[i])**2))
plt.figure(2)
plt.plot(mean_fpr, mean_tpr, color='b',
         label="Mean ROC (AUC = %0.3f)" % (mean_auc),
         lw=3, alpha=.8)


# Format the graphs appropriately, and then display them
plt.figure(1)
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for kNN Algorithm")
plt.legend(loc="lower right")
plt.grid(True, alpha=0.4)

plt.figure(2)
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for SVM Algorithm")
plt.legend(loc="lower right")
plt.grid(True, alpha=0.4)

plt.show()
```