Mini project 1

IEE 520

Submitted by:

Ting Yan Fok
Ali Sarabi
Kiarsh Ghasemzadeh
Sanket Bhale
Indraneel Phirke

Question 1. Use a Naïve Bayes classifier. Complete the following calculations without a Naïve Bayes software package so you understand the steps.

a) Fill the predicted class for each instance and record your prediction in the right-most column in the table

The predicted class labels are shown in the most right column.

Instance	x1	x2	x3	y	Posterior (class 0)	Posterior (class 1)	Posterior (class 2)	max value	Predicted Class (Yhat)
1	1	Active	High	0	0.00556	0.00419	0.00338	0.00556	0
2	1	Sedentary	Normal	2	0.00556	0.00419	0.01013	0.01013	2
3	6	Sedentary	Normal	0	0.00732	0.00698	0.01491	0.01491	2
4	2.5	Active	Normal	1	0.01792	0.01194	0.01374	0.01792	0
5	3	Sedentary	Normal	1	0.00975	0.00647	0.01467	0.01467	2
6	7	Active	High	2	0.00493	0.00615	0.00433	0.00615	1
7	4.6	Active	Normal	0	0.01981	0.01468	0.01604	0.01981	0
8	1.8	Active	Normal	2	0.01496	0.01035	0.01216	0.01496	0
9	9	Active	High	1	0.00142	0.00379	0.00265	0.00379	1
10	8.5	Sedentary	Normal	2	0.00205	0.00440	0.00922	0.00922	2

In order to calculate the values in the table above (posteriors), we used the Bayes formula as follows:

$$P(Y = y| X = x) = \frac{P(X=x| Y=y).\, P(Y=y)}{P(X=x)}$$

And since the values for denominators are equal for all classes, we can just compare the nominators with each other and find the Posterior for every class. The predicted class label then will be found by finding which class has the most value in the posterior value.
In order to calculate each posterior value, since we have conditional interdependencies, we can rewrite the formulas as follows:

$$P(X = x| Y = y).\, P(Y = y) = P(X = x_1| Y = y).P(X = x_2| Y = y).P(X = x_3| Y = y).\, P(Y = y)$$

Since the values for x_1 is continuous, we need to estimate the underlying probability distribution function. The $P(X = x_1| Y = y)$ is assumed to be normally distributed with a mean of and standard deviation.

Sample Mean and Standard Deviation		
Y	mean (x1)	sd (x1)
0	3.866666667	2.579405616
1	4.833333333	3.617089069
2	4.575	3.731286641

And the conditional probabilities for x_1 are calculated as follows:

$P(X = x_1 | Y = y) \simeq \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$

P(X1|Y=y)	0	1	2
(x1i- sample mean)^2	P(X1 | Y=0)	P(X1 | Y=1)	P(X1 | Y=2)
8.218	0.083	0.063	0.068
12.781	0.083	0.063	0.068
4.551	0.110	0.105	0.099
5.444	0.134	0.090	0.092
3.361	0.146	0.097	0.098
5.881	0.074	0.092	0.087
0.538	0.149	0.110	0.107
7.701	0.112	0.078	0.081
17.361	0.021	0.057	0.053
15.406	0.031	0.066	0.061

And the conditional probabilities for x_2 and x_3 are calculated as follows:

$P(X = x_2 | Y = y) = \frac{P(X=x_2, Y=y)}{P(Y=y)}$

P(X2|Y=y)	0	1	2
Active	0.667	0.667	0.500
Sedentary	0.333	0.333	0.500

$P(X = x_3 | Y = y) = \frac{P(X=x_3, Y=y)}{P(Y=y)}$

P(X3|Y=y)	0	1	2
High	0.333	0.333	0.250
Normal	0.667	0.667	0.750

And finally the probabilities for actual labels are calculated with $P(Y = y) = \frac{N(Y=y)}{N}$ where N is total instances.
Sample calculation for Y=0 is $P(Y = 0) = 3/10 = 0.3$. Similarly, the results are tabulated in the following table:

Label	P(Y=y)
0	0.3
1	0.3
2	0.4

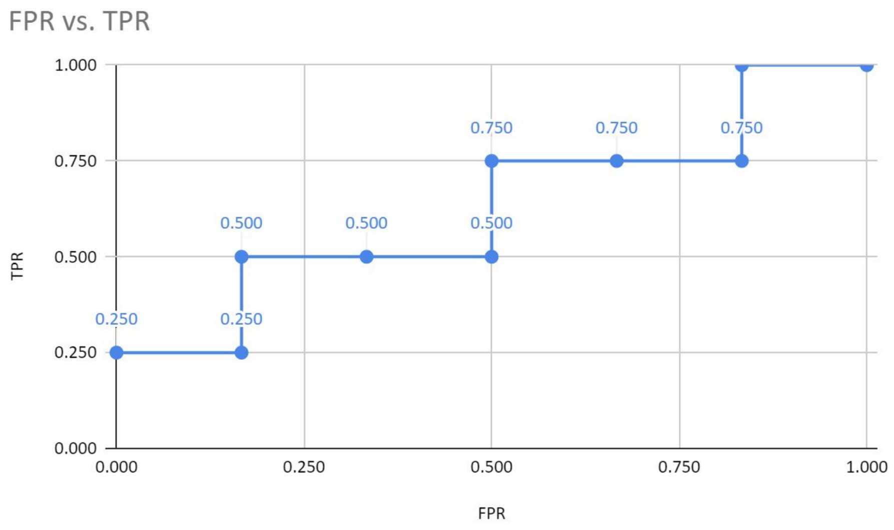
b) Create the confusion matrix for all three classes.

		Predicted Y		
		0	1	2
Actual Y	0	2	0	1
	1	1	1	1
	2	1	1	2

c) Assume that class 2 is the positive class. Calculate and plot a ROC curve for this model.

Instance	Predicted Class	Class 2 probability estimate	Coded Actual Class	TPR	FPR
10	2	0.5884	1	0.250	0.000
3	2	0.5104	-1	0.250	0.167
2	2	0.5096	1	0.500	0.167
5	2	0.4751	-1	0.500	0.333
9	1	0.3368	-1	0.500	0.500
8	0	0.3246	1	0.750	0.500
7	0	0.3175	-1	0.750	0.667
4	0	0.3151	-1	0.750	0.833
6	1	0.2810	1	1.000	0.833
1	0	0.2573	-1	1.000	1.000

Assuming that class 2 is a positive class, instances in the table above are sorted by probability estimate value in order to plot the ROC curve. The area under the ROC curve below is 0.625 which is greater than 0.5 showing that the classifier works fine but still is not a good classifier.



Question 2: Select a dataset of your own choice of sufficient size, at least 500 rows, (prefer over a thousand rows), and at least 10 predictor attributes (prefer at least 20), and a target attribute. The target should be categorical.
Build a naïve Bayes classifier for your data in Python. Evaluate the generalization error of the classifier in three ways:
o From the training data
o From a test set of 20% of the data
o From cross-validation with 5 folds

a) Provide the code and the results of your analysis. Comment on any differences in these estimates of generalization error.

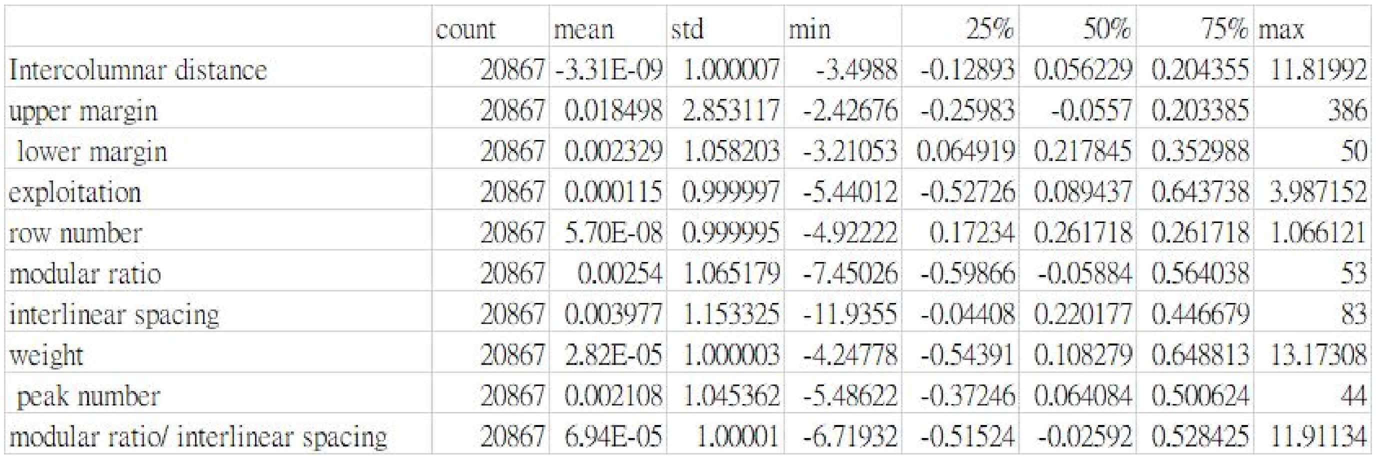
About the Dataset

We found our data from the Center for Machine Learning and Intelligent Systems (machine learning repository with address https://archive.ics.uci.edu/ml/datasets). Our data set has been extracted from 800 images of the "Avila Bible", a 12th-century giant latin copy of the Bible. The prediction task consists in associating each pattern to a copyist. Therefore, we are trying to classify the data into different letters (A,B,C,D,E,F,G,H,I,W,X,Y). The data set consisted of the training set and test set, in which we merged and combined so we have 20867 instances. Also, we have 10 attributes and all of the attribute characteristics are real numbers. We are trying to train our model to be able to classify the data using the attributes given by implementing the Gaussian Naive Bayes Classifier.

Attributes Information:

F1: intercolumnar distance
F2: upper margin
F3: lower margin
F4: exploitation
F5: row number
F6: modular ratio
F7: interlinear spacing
F8: weight
F9: peak number
F10: modular ratio/ interlinear spacing
Target Value Classes: A, B, C, D, E, F, G, H, I, W, X, Y

Summary of Data:



	count	mean	std	min	25%	50%	75%	max
Intercolumnar distance	20867	-3.31E-09	1.000007	-3.4988	-0.12893	0.056229	0.204355	11.81992
upper margin	20867	0.018498	2.853117	-2.42676	-0.25983	-0.0557	0.203385	386
lower margin	20867	0.002329	1.058203	-3.21053	0.064919	0.217845	0.352988	50
exploitation	20867	0.000115	0.999997	-5.44012	-0.52726	0.089437	0.643738	3.987152
row number	20867	5.70E-08	0.999995	-4.92222	0.17234	0.261718	0.261718	1.066121
modular ratio	20867	0.00254	1.065179	-7.45026	-0.59866	-0.05884	0.564038	53
interlinear spacing	20867	0.003977	1.153325	-11.9355	-0.04408	0.220177	0.446679	83
weight	20867	2.82E-05	1.000003	-4.24778	-0.54391	0.108279	0.648813	13.17308
peak number	20867	0.002108	1.045362	-5.48622	-0.37246	0.064084	0.500624	44
modular ratio/ interlinear spacing	20867	6.94E-05	1.00001	-6.71932	-0.51524	-0.02592	0.528425	11.91134

Before fitting the data to model and train, we looked into summary statistics of it by using the command data.describe(). It can be observed that most of the features have a similar range except F2 upper margin.

Class Label Frequency Count:
Note that the target variable of this dataset is categorical and they are also letters. So, we encoded the letter labels before fitting the data to the model.
Besides, a frequency count of each class label shows us that they are imbalanced, especially class B has only ten records in the dataset. This is a call for normalized Confusion Matrix when analyzing the results.

A	8572
B	10
C	206
D	705
E	2190
F	3923
G	893
H	1039
I	1663
W	89
X	1044
Y	533

Name: Class: A, B, C, D, E, F, G, H, I, W, X, Y, dtype: int64

Model training:

As stated by the problem description, we first split our data into training and test sets which compose 80% and 20% of all the data, then we used 5-fold cross-validation to further split the training and validation sets.

Python code:

Naive Bayes

```
from google.colab import drive
drive.mount('/content/drive')
```

```
# For compatibility with Python 2
from __future__ import print_function

# To load datasets
from sklearn import datasets

# To import the classifier (Naive Bayes)
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier

# To split data to train and test
from sklearn.model_selection import train_test_split

# To measure accuracy
from sklearn import metrics

# To support plots
import matplotlib.pyplot as plt

# To display all the plots inline
%matplotlib inline

# Import Numpy library
import numpy as np

# To import for Cross-Validation Set
from sklearn import model_selection

# To increase quality of figures
plt.rcParams["figure.figsize"] = (20, 10)
```

Load the data

```
from google.colab import drive
drive.mount('/content/drive')
#import data
import pandas as pd
path = "/content/drive/Shared drives/IEE520/Avila-Data-Edited_Final.csv"
data = pd.read_csv(path)
#print(data.shape)
print(data.describe())
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
data_transform = data.apply(le.fit_transform)


# Need to convert as yhat returned by the code is numpy
# Otherwise, we input one Dataframe and one numpy array to the confusion matrix
# and this will cause calculation error

data_transform = data_transform.to_numpy()

X = data_transform[:, 0:-1]
y = data_transform[:, -1]

# Count of each target values in the dataset
tgt_values, unique_counts = np.unique(y,return_counts=True)
print(list(zip(tgt_values, unique_counts)))

```

Split to train and test

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Train the model and predict

```
model = GaussianNB()
model.fit(X_train, y_train)
yhat_train = model.predict(X_train) #model.predict always returns numpy array
yhat_test = model.predict(X_test)
```

Model evaluation

Accuracy:

```
print(metrics.accuracy_score(y_train, yhat_train))
print(metrics.mean_squared_error(y_train, yhat_train))
```

Classification report:

```
print(metrics.accuracy_score(y_test, yhat_test))
print(metrics.mean_squared_error(y_test, yhat_test))
print(metrics.classification_report(y_train, yhat_train))

# You need to install pandas_ml in order to use that!
 !pip install pandas_ml
```

Confusion matrix

Confusion matrix:

```
from pandas_ml import ConfusionMatrix
```

Stats:

```
cm = ConfusionMatrix(y_train, yhat_train)
print(cm)
```

Normalized Confusion Matrix (as we have imbalanced target values)[1]

```
import numpy as np
import matplotlib.pyplot as plt


from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels

# import some data to play with
class_names = np.sort(np.unique(y))


# Run classifier, using a model that is too regularized (C too low) to see
# the impact on the results
#classifier = svm.SVC(kernel='linear', C=0.01)
#y_pred = classifier.fit(X_train, y_train).predict(X_test)


def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
```

[1] We modified the code from the scikit-learn website: https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py

```
●               title = 'Normalized confusion matrix'
●           else:
●               title = 'Confusion matrix, without normalization'
●
●       # Compute confusion matrix
●       cm = confusion_matrix(y_true, y_pred)
●       print(cm)
●
●
●       # Only use the labels that appear in the data
●       classes = classes[unique_labels(y_true, y_pred)]
●       if normalize:
●           cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
●           print("Normalized confusion matrix")
●       else:
●           print('Confusion matrix, without normalization')
●
●       print(cm)
●
●       fig, ax = plt.subplots()
●       im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
●       ax.figure.colorbar(im, ax=ax)
●       # We want to show all ticks...
●       ax.set(xticks=np.arange(cm.shape[1]),
●              yticks=np.arange(cm.shape[0]),
●              # ... and label them with the respective list entries
●              xticklabels=classes, yticklabels=classes,
●              title=title,
●              ylabel='True label',
●              xlabel='Predicted label')
●
●       # Rotate the tick labels and set their alignment.
●       plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
●                rotation_mode="anchor")
●
●       # Loop over data dimensions and create text annotations.
●       fmt = '.2f' if normalize else 'd'
●       thresh = cm.max() / 2.
●       for i in range(cm.shape[0]):
●           for j in range(cm.shape[1]):
●               ax.text(j, i, format(cm[i, j], fmt),
●                       ha="center", va="center",
●                       color="white" if cm[i, j] > thresh else "black")
●       fig.tight_layout()
●       return ax
●
●
●   np.set_printoptions(precision=2)
```

```
●   # Plot normalized confusion matrix
●   plot_confusion_matrix(y_train, yhat_train, classes=class_names, normalize=True,
                        title='Normalized confusion matrix')
plt.show()
```

Cross-validation

```
seed = 2357
np.random.seed(seed)
actuals = []
probs = []
hats = []

SCALE = False

kfold = model_selection.KFold(n_splits=5, shuffle=True, random_state=seed)
# Cross-validation
for train, test in kfold.split(X, y):
    # print('train: %s, test: %s' % (train, test))
    # Train classifier on training data, predict test data
    if SCALE:
        scaler.fit(X[train]) # Learn scaling parameters on training data
        Xtrain = scaler.transform(X[train])
        Xtest = scaler.transform(X[test]) # Apply transform to test data
    else:
        Xtrain = X[train]
        Xtest = X[test]

 temp = pd.DataFrame(Xtest.tolist(), columns = data.columns[0:-1])
    print('Cross validation set' + str(ind))
    ind += 1
    display(temp.describe())

    model.fit(Xtrain, y[train])
    foldhats = model.predict(Xtest)
    foldprobs = model.predict_proba(Xtest)[:,1] # Class probability estimates for ROC
curve
    actuals = np.append(actuals, y[test]) # Combine targets, then probs, and then
predictions from each fold
    probs = np.append(probs, foldprobs)
    hats = np.append(hats, foldhats)
```

Model evaluation on Cross Validation Set

Accuracy:

```
print(metrics.accuracy_score(actuals, hats))
print(metrics.mean_squared_error(actuals, hats))
```

Classification report:

```
print(metrics.classification_report(actuals, hats))
```

Test (cross-validation) confusion matrix

```
cm = ConfusionMatrix(actuals, hats)
print(cm)
```

```
cm.print_stats()
ax = cm.plot(backend='seaborn', annot=True, fmt='g')
ax.set_title('Test Confusion Matrix')
plt.show()
```

Normalized Confusion Matrix for Cross Validation

```
# Plot normalized confusion matrix
plot_confusion_matrix(actuals.astype(int), hats.astype(int), classes=class_names, 
normalize=True,
                      title='Normalized confusion matrix')

plt.plot(backend='seaborn', annot=False, fmt='g')
plt.show()
```

ROC curve[2]

```
from itertools import cycle # for plotting
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from scipy import interp

# Binarize the output
y = label_binarize(y, classes=class_names)
n_classes = y.shape[1]

# Learn to predict each class against the other
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.5,
                                                    random_state=0)
classifier = OneVsRestClassifier(GaussianNB())
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)


# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

```
# Compute macro-average ROC curve and ROC area
lw = 2 #line width for plotting
```

[2]**We modified the code from the scikit-learn website:**
https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

```python

# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors =
cycle(['darkgoldenrod','darkgray','darkgreen','darkkhaki','darkmagenta','darkolivegreen'
,'darkorange','darkorchid','darkred',
'darksalmon','darkseagreen','darkslateblue','darkslategray','darkturquoise','darkviolet'
])


for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for multi-class GaussanNB')
plt.legend(loc="lower right")
plt.show()
```

After executing the code and fitting data, the value of the Generalization Error obtained by the three methods are as follows:

Generalization Errors of the Classifier

Mean Squared Error (Training Data)	Mean Squared Error (Test Data)	Mean Squared Error (Crossvalidation with 5 folds)
11.5762	**12.2245**	**11.7818**

As we can see the mean squared error (MSE) from the Training Data, Test Data and 5folds are very similar and the reason behind that is we had so many instances to train our data on and we had very few attributes in comparison. Also the classes number were not that many making the model very accurate. As we got much higher mean squared error for test data versus training data in another data set that had 40 attributes for 500 instances, and the model needed a prior analysis to find the more important attributes and then try to train our model. But since this was out of the scope of the project we changed our data set.

Accuracies of the Classifier

Accuracy score (Training Data)	Accuracy score (Test Data)	Accuracy score (Crossvalidation with 5 folds)
0.53364	**0.53234**	**0.53228**

As we can see the Accuracy scores from the Training Data, Test Data and 5folds are very similar and it shows that that the measurements are close to the actual or expected value. The maximum value for accuracy scores is one. The score of 0.53 in accuracy is a good value and represents the relative accuracy of the classifier.

b) Provide the code and a confusion matrix, summary statistics, and a ROC curve calculated from the cross-validation only

For getting Summary statistics of cross-validation set in each fold we have added the following code:

```
•   temp = pd.DataFrame(Xtest.tolist(), columns = data.columns[0:-1])
•      print('Cross validation set' + str(ind))
•      ind += 1
•      display(temp.describe())
```

The summary statistics for each fold are given as follows:

Cross validation set 1

	Intercolumnar distance	upper margin	lower margin	exploitation	row number	modular ratio	interlinear spacing	weight	peak number	modular ratio/ interlinear spacing
count	4174	4174	4174	4174	4174	4174	4174	4174	4174	4174
mean	65.82	93.94	107.26	383.11	35.44	124.17	179.70	9837.70	172.07	9665.56
std	26.57	42.13	55.15	213.38	8.79	32.13	28.16	5607.84	41.17	5529.35
min	0	0	0	0	0	0	0	5	3	0
25%	52	64	69	201	37	103	176	5024.75	155.25	4937.75
50%	67	90	111	383.5	38	121	186	9908.5	175	9658
75%	79	123	149	565	38	145	195	14667.75	195	14471.75
max	143	207	231	750	47	261	280	19610	287	19289

Cross validation set 2

	Intercolumnar distance	upper margin	lower margin	exploitation	row number	modular ratio	interlinear spacing	weight	peak number	modular ratio/ interlinear spacing
count	4174	4174	4174	4174	4174	4174	4174	4174	4174	4174
mean	66.6	94.2	106.7	384.7	35.7	123.9	178.9	9858.9	171.6	9593.7
std	25.9	41.4	55.8	211.5	8.3	32.8	28.8	5628.6	40.6	5571.2
min	0	0	0	0	0	0	0	3	2	0
25%	52	63	67	205	37	102	176	5076.5	154	4715.25
50%	68	90	111	386.5	38	123	184	9938.5	175	9534
75%	79	123	149	564.75	38	143	195	14700.5	195	14351
max	143	205	229	750	47	259	269	19598	282	19292

Cross validation set 3

	Intercolumnar distance	upper margin	lower margin	exploitation	row number	modular ratio	interlinear spacing	weight	peak number	modular ratio/ interlinear spacing
count	4173	4173	4173	4173	4173	4173	4173	4173	4173	4173
mean	66.7	93.9	107.0	384.9	35.7	123.4	179.3	9867.4	172.6	9558.5
std	26.4	41.5	55.0	213.8	8.3	32.1	29.1	5578.6	40.1	5459.3
min	0	0	0	0	0	0	0	4	1	0
25%	52	64	67	197	37	103	176	5138	155	4906
50%	68	89	110	393	38	121	186	9901	175	9535
75%	81	123	149	570	38	142	194	14708	195	14220
max	143	205	230	750	47	256	279	19606	284	19290

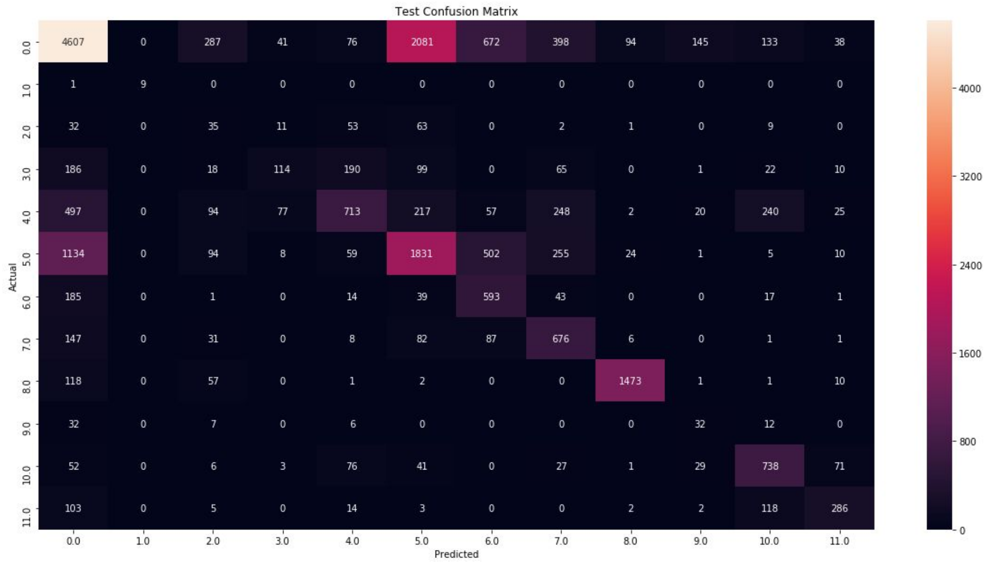
Cross validation set 4

	Intercolumnar distance	upper margin	lower margin	exploitation	row number	modular ratio	interlinear spacing	weight	peak number	modular ratio/ interlinear spacing
count	4173	4173	4173	4173	4173	4173	4173	4173	4173	4173
mean	67.0	94.3	107.4	382.6	35.6	124.1	179.7	9815.7	173.3	9574.1
std	25.9	41.4	54.8	210.8	8.3	31.9	28.5	5539.9	39.3	5497.4
min	0	0	0	0	0	0	0	0	3	0
25%	52	64	68	201	37	103	176	5074	156	4893
50%	68	90	112	384	38	121	186	9784	175	9503
75%	81	123	149	566	38	143	195	14543	195	14242
max	143	204	230	750	47	248	276	19600	285	19284

Cross validation set 5

	Intercolumnar distance	upper margin	lower margin	exploitation	row number	modular ratio	interlinear spacing	weight	peak number	modular ratio/ interlinear spacing
count	4173	4173	4173	4173	4173	4173	4173	4173	4173	4173
mean	65.94393	93.5	107.7	383.3	35.5	123.7	179.2	9826.1	171.6	9490.7
std	27.1	41.7	55.6	212.9	8.8	32.2	28.8	5605.2	42.0	5548.3
min	0	0	0	0	0	0	0	12	0	0
25%	52	63	68	201	37	102	176	4993	155	4654
50%	67	89	111	385	38	121	186	9841	175	9371
75%	80	123	150	568	38	142	195	14639	195	17251
max	143	206	230	750	47	260	268	19611	286	19293

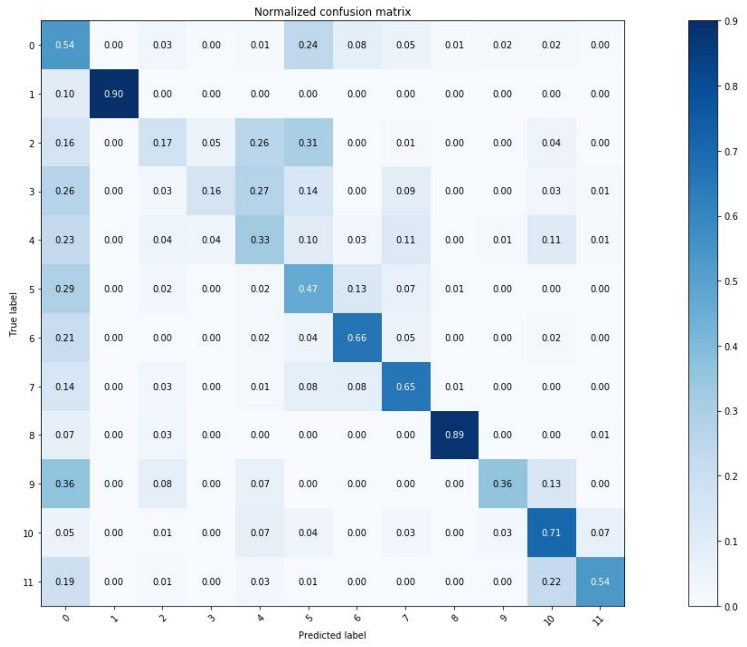
Confusion matrix for cross-validation method is provided as follows:



Actual \ Predicted	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0
0.0	4607	0	287	41	76	2081	672	398	94	145	133	38
1.0	1	9	0	0	0	0	0	0	0	0	0	0
2.0	32	0	35	11	53	63	0	2	1	0	9	0
3.0	186	0	18	114	190	99	0	65	0	1	22	10
4.0	497	0	94	77	713	217	57	248	2	20	240	25
5.0	1134	0	94	8	59	1831	502	255	24	1	5	10
6.0	185	0	1	0	14	39	593	43	0	0	17	1
7.0	147	0	31	0	8	82	87	676	6	0	1	1
8.0	118	0	57	0	1	2	0	0	1473	1	1	10
9.0	32	0	7	0	6	0	0	0	0	32	12	0
10.0	52	0	6	3	76	41	0	27	1	29	738	71
11.0	103	0	5	0	14	3	0	0	2	2	118	286

Confusion Matrix of cross validation (Not Normalized)

As we mentioned earlier, a frequency count of each class label shows us that they are imbalanced. Plus, it is very hard to tell from the confusion matrix without normalization the performance of classifier is. Cell with higher number of classified labels have brighter color above, but this is unfair to compare as some of the class labels have very few instances. Therefore normalized Confusion Matrix is provided to further analyze the results.

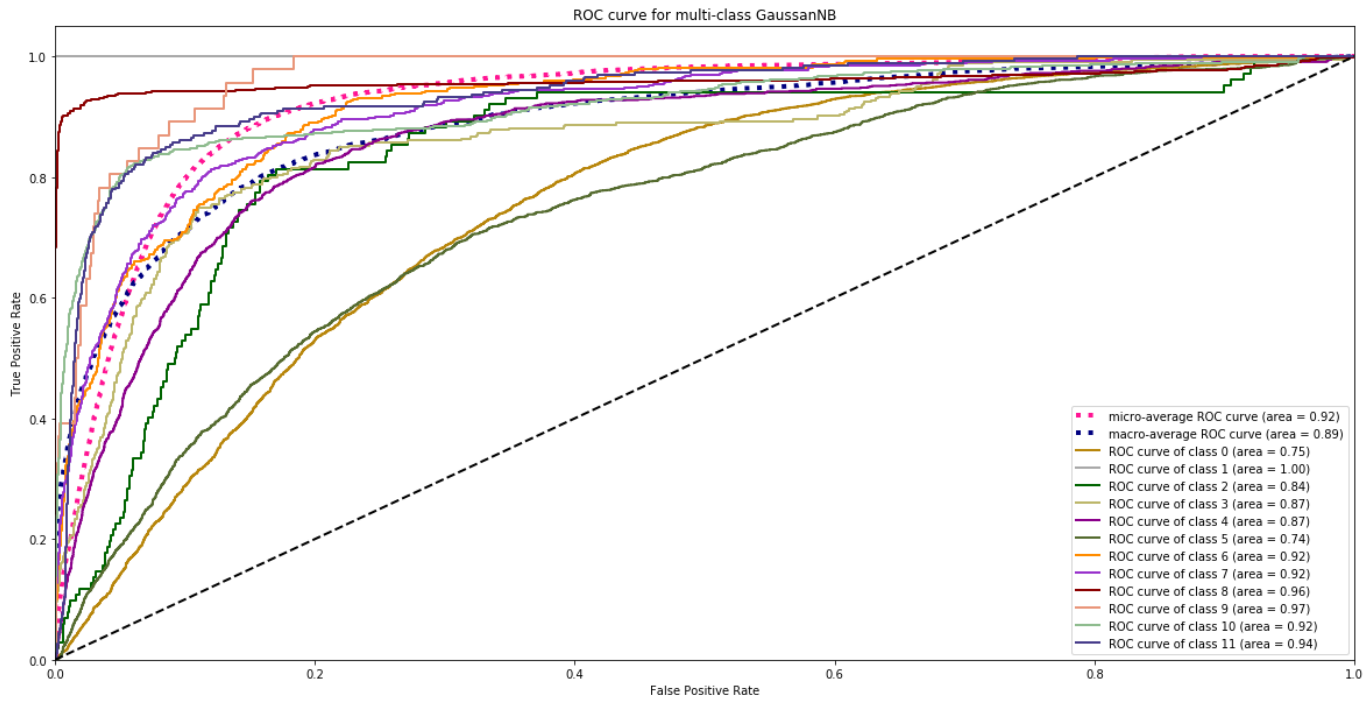


True label \ Predicted label	0	1	2	3	4	5	6	7	8	9	10	11
0	0.54	0.00	0.03	0.00	0.01	0.24	0.08	0.05	0.01	0.02	0.02	0.00
1	0.10	0.90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.16	0.00	0.17	0.05	0.26	0.31	0.00	0.01	0.00	0.00	0.04	0.00
3	0.26	0.00	0.03	0.16	0.27	0.14	0.00	0.09	0.00	0.00	0.03	0.01
4	0.23	0.00	0.04	0.04	0.33	0.10	0.03	0.11	0.00	0.01	0.11	0.01
5	0.29	0.00	0.02	0.00	0.02	0.47	0.13	0.07	0.01	0.00	0.00	0.00
6	0.21	0.00	0.00	0.00	0.02	0.04	0.66	0.05	0.00	0.00	0.02	0.00
7	0.14	0.00	0.03	0.00	0.01	0.08	0.08	0.65	0.01	0.00	0.00	0.00
8	0.07	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.89	0.00	0.00	0.01
9	0.36	0.00	0.08	0.00	0.07	0.00	0.00	0.00	0.00	0.36	0.13	0.00
10	0.05	0.00	0.01	0.00	0.07	0.04	0.00	0.03	0.00	0.03	0.71	0.07
11	0.19	0.00	0.01	0.00	0.03	0.01	0.00	0.00	0.00	0.00	0.22	0.54

Confusion Matrix of cross validation (Normalized)

As we can see in Confusion Matrix above, the model can predict the correct class very accurate for some letters (e.g. B, I and Y). But not as accurate for some other letters (such as C, D, E, F and X). For the rest of the letters the accuracy is acceptable (above 50 percent). Also most of the density is scattered around the diameter which indicates the good performance of the classifier.

ROC curve



Area under ROC curve

In the class, we have seen how ROC curve be computed for binary classification. In this project, the target values have 12 classes. Therefore, we have to binarize the output each time to get the ROC curve for each class label.

Each point on the **ROC curve** represents a sensitivity/specificity pair corresponding to a particular decision threshold. The **area under** the **ROC curve** (**AUC**) is a measure of how well a parameter can distinguish between two diagnostic groups (diseased/normal). Therefore, our model can distinguish perfectly for class 2, letter B, (area underneath= 1.00) and in the worst case for class 5, letter F, (area underneath= 0.74).
Micro- and macro-averages (for whatever metric) will compute slightly different things, and thus their interpretation differs. A macro-average will compute the metric

independently for each class and then take the average (hence treating all classes equally), whereas a micro-average will aggregate the contributions of all classes to compute the average metric. In a multi-class classification setup, micro-average is preferable if you suspect there might be class imbalance (i.e you may have many more examples of one class than of other classes). Therefore, our Micro-average area is slightly higher than our Macro-average area, because our classes are not very equal as the data description clearly states.