Student ID: 1012869

COMP30027 Project 1 Report

Question 1

```
 -------- Total Data Confusion Matrix --------
 [[ 22  23  36   3   8   2   0   1   0   0]
  [  1  68   5   7   0   1   0   0   0   0]
  [  7  14  85   8   5   2   0   0   0   0]
  [  0   0   0 174   0   0  16   0   0   0]
  [  3   4   8   0  49   0   0   2   0   0]
  [  0  25   4   1   2  14   1   5   0   0]
  [  3   1   1   0   0   0  64   0   4   0]
  [  1   0   0   0   1   0   0  59   0   2]
  [  1   0   2   0   1   1   1   4  44   5]
  [  0   0   0   0   1   0   1   0   3  57]]
```

Performance metrics for each class [TP, FP, TN, FN]

{'bridge': array([22, 16, 752, 73]), 'childs': array([68, 67, 714, 14]), 'downwarddog': array([85, 56, 686, 36]), 'mountain': array([174, 19, 654, 16]), 'plank': array([49, 18, 779, 17]), 'seatedforwardbend': array([14,   6, 805, 38]), 'tree': array([64, 19, 771,   9]), 'trianglepose': array([59, 12, 788,   4]), 'warrior1': array([44,   7, 797, 15]), 'warrior2': array([57,   7, 794,   5])}

```
-------- Total Data Macro-averaging Statistics --------
Precision: 0.737383 , Recall: 0.716911 , F-Score: 0.727003
-------- Total Data Micro-averaging Statistics --------
Precision: 0.736964 , Recall: 0.736964 , F-Score: 0.736964
-------- Total Data Weighted-averaging Statistics --------
Precision: 0.736964 , Recall: 0.796995 , F-Score: 0.765805
```

Macro-averaging vs Micro-averaging

In macro-averaging, precision and recall are calculated independently per class and an average is calculated. This treats all the classes equally.

$$Precision_M = \frac{\sum_{i=1}^{c} Precision(i)}{c} \qquad Recall_M = \frac{\sum_{i=1}^{c} Recall\ (i)}{c}$$

On the other hand, precision and recall in micro-averaging are calculated by summing up all the classes' true positives, false positives and false negatives shown below to compute the average value for all classes.

$$Precision_\mu = \frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c} TP_i + FP_i} \qquad Recall_\mu = \frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c} TP_i + FN_i}$$

In the statistics, precision for macro-averaging is slightly higher than micro-averaging. This is because the average class precision for macro is better, for example classes like “mountain” or “tree” maintain a high precision of 0.90 and 0.77 respectively, which brings the overall average precision higher. In contrast, micro-averaging takes into consideration class imbalance, so classes with more classified instances like “downwarddog” or “childs” will contribute more to this precision value. These classes may have many instances of FP relative to TP, which can decrease precision.

Recall for micro-averaging is slightly higher than macro-averaging. This is because the average recall for each class is slightly lower (in macro than micro), classes like “bridge” have high FN relative to TP. In micro-averaging, there are also other (larger) classes that have better recall scores and will contribution larger to the overall recall like “mountain” or “downwarddog” which have large TP values but quite low FN values.

'bridge' = 22 TP 16FP 73FN
'childs' = 68 TP 67FP 14FN
'downwarddog' = 85TP 56FP 36FN
'mountain' = 174TP 19FP 16FN
'plank' = 49TP 18FP 17FN
'seatedforwardbend' = 14TP 6FP 38FN
'tree' = 64TP 19FP 9FN
'trianglepose' = 59TP 12FP 4FN
'warrior1' = 44TP 7FP 15FN
'warrior2' = 57TP 7FP 5FN

<u>Macro-averaging vs Weighted-averaging</u>

Weighted-averaging is similar to macro-averaging although it considers the proportion of each class in the dataset.

$$Precision_W = \sum_{i=1}^{c} \left(\frac{n_i}{N}\right) Precision(i) \qquad Recall_W = \sum_{i=1}^{c} \left(\frac{n_i}{N}\right) Recall(i)$$

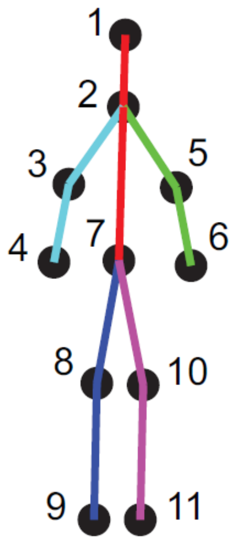
For larger classes that appear more in the dataset, it has more significance. For example classes like “mountain” or “downwarddog, its precision/recall will contributes more to the overall value. The precision for weighted is lower than macro-averaging, this shows that for classes that appear more often in the dataset, precision for predicting it is lower. However the recall for weighted is higher than macro-averaging. This shows that FN relative to TP values are lower for larger classes.

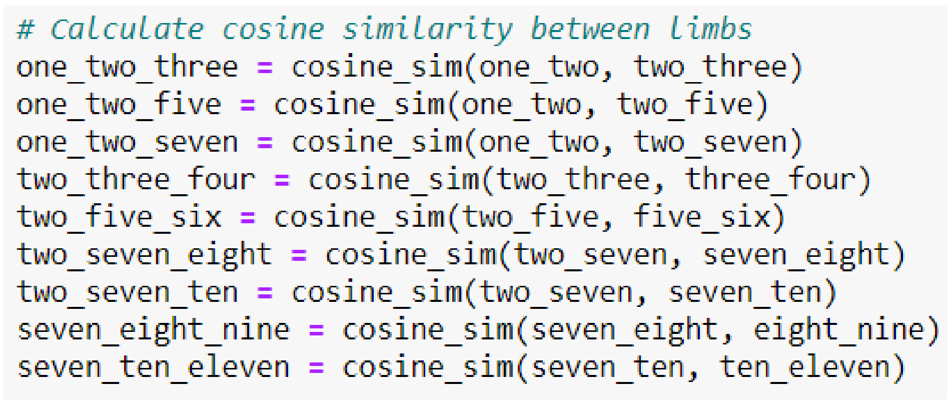
```
mountain             190
downwarddog          121
bridge                95
childs                82
tree                  73
plank                 66
trianglepose          63
warrior2              62
warrior1              59
seatedforwardbend     52
Name: class, dtype: int64
```

Student ID: 1012869

Question 6

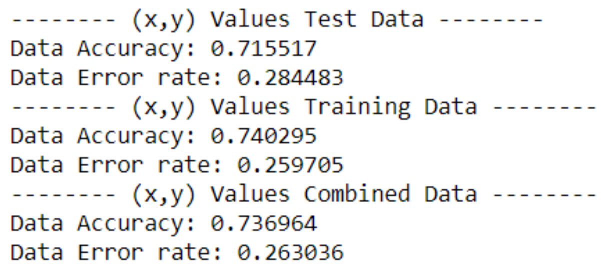
A new feature was engineered by calculating the cosine similarity between 2 (connected) limbs. For example, parts 1, 2 and 5 and parts 7, 8 and 9. The limbs used are shown below. A vector is calculated between 2 points, then a cosine similarity value is calculated between 2 vectors. If either of the points used to calculate the vector is np.NaN then the vector and the cosine similarity will also be np.NaN. This is then passed to the train, predict and evaluate functions used in the earlier (x,y) value classifier to use Naive Bayes similarly.



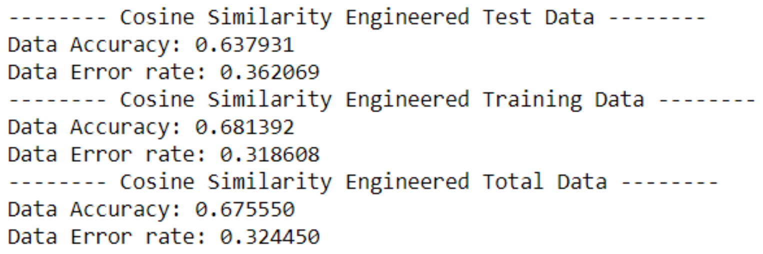


```
# Calculate cosine similarity between limbs
one_two_three = cosine_sim(one_two, two_three)
one_two_five = cosine_sim(one_two, two_five)
one_two_seven = cosine_sim(one_two, two_seven)
two_three_four = cosine_sim(two_three, three_four)
two_five_six = cosine_sim(two_five, five_six)
two_seven_eight = cosine_sim(two_seven, seven_eight)
two_seven_ten = cosine_sim(two_seven, seven_ten)
seven_eight_nine = cosine_sim(seven_eight, eight_nine)
seven_ten_eleven = cosine_sim(seven_ten, ten_eleven)
```

Position (x,y) based classifier vs Cosine similarity (between connected limbs)



```
-------- (x,y) Values Test Data --------
Data Accuracy: 0.715517
Data Error rate: 0.284483
-------- (x,y) Values Training Data --------
Data Accuracy: 0.740295
Data Error rate: 0.259705
-------- (x,y) Values Combined Data --------
Data Accuracy: 0.736964
Data Error rate: 0.263036
```



```
-------- Cosine Similarity Engineered Test Data --------
Data Accuracy: 0.637931
Data Error rate: 0.362069
-------- Cosine Similarity Engineered Training Data --------
Data Accuracy: 0.681392
Data Error rate: 0.318608
-------- Cosine Similarity Engineered Total Data --------
Data Accuracy: 0.675550
Data Error rate: 0.324450
```

As shown, the positional classifier yields better accuracy than cosine similarity based for all testing data. This may be because three points are used for each attribute of this feature, if there are NaN values in any of the points used, the attribute value will also be NaN and ignored in our Naive Bayes classifier. The absence of data will contribute to the loss of accuracy.

Student ID: 1012869

Performance Metrics

```
-------- Total Data Confusion Matrix --------
[[ 22  23  36   3   8   2   0   1   0   0]
 [  1  68   5   7   0   1   0   0   0   0]
 [  7  14  85   8   5   2   0   0   0   0]
 [  0   0   0 174   0   0  16   0   0   0]
 [  3   4   8   0  49   0   0   2   0   0]
 [  0  25   4   1   2  14   1   5   0   0]
 [  3   1   1   0   0   0  64   0   4   0]
 [  1   0   0   0   1   0   0  59   0   2]
 [  1   0   2   0   1   1   1   4  44   5]
 [  0   0   0   0   1   0   1   0   3  57]]
```

```
-------- Cosine Similarity Confusion Matrix --------
[[ 48   1  13  12   5   2   3   2   3   6]
 [  1  43  13  14   2   1   2   4   2   0]
 [  2   0  50  39   6  11   2   4   1   6]
 [  0   0   0 186   3   0   1   0   0   0]
 [  0   0   5  15  29   0   1  10   0   6]
 [  1  10   8   5   3  12   4   4   1   4]
 [  0   0   1   3   1   0  68   0   0   0]
 [  1   0   1   0   0   1   0  55   1   4]
 [  2   0   8   0   1   0   0   4  34  10]
 [  1   0   1   0   1   0   0   1   0  58]]
```

(X,Y) Position Based [TP, FP, TN, FN]

{'bridge': array([22, 16, 752, 73]), 'childs': array([68, 67, 714, 14]), 'downwarddog': array([85, 56, 686, 36]), 'mountain': array([174, 19, 654, 16]), 'plank': array([49, 18, 779, 17]), 'seatedforwardbend': array([14,   6, 805, 38]), 'tree': array([64, 19, 771,   9]), 'trianglepose': array([59, 12, 788,   4]), 'warrior1': array([44,   7, 797, 15]), 'warrior2': array([57,   7, 794,   5])}

Cosine Similarity Based [TP, FP, TN, FN]

{'bridge': array([48,   8, 760, 47]), 'childs': array([43, 11, 770, 39]), 'downwarddog': array([50, 50, 692, 71]), 'mountain': array([186, 88, 585,   4]), 'plank': array([29, 22, 775, 37]), 'seatedforwardbend': array([12, 15, 796, 40]), 'tree': array([68, 13, 777,   5]), 'trianglepose': array([55, 29, 771,   8]), 'warrior1': array([34,   8, 796, 25]), 'warrior2': array([58, 36, 765,   4])}

```
-------- Total Data Macro-averaging Statistics --------
Precision: 0.737383 , Recall: 0.716911 , F-Score: 0.727003
-------- Total Data Micro-averaging Statistics --------
Precision: 0.736964 , Recall: 0.736964 , F-Score: 0.736964
-------- Total Data Weighted-averaging Statistics --------
Precision: 0.736964 , Recall: 0.796995 , F-Score: 0.765805
```

```
-------- Total Cosine Similarity Data Macro-averaging Statistics --------
Precision: 0.676616 , Recall: 0.640826 , F-Score: 0.658235
-------- Total Cosine Similarity Data Micro-averaging Statistics --------
Precision: 0.675550 , Recall: 0.675550 , F-Score: 0.675550
-------- Total Cosine Similarity Data Weighted-averaging Statistics --------
Precision: 0.675550 , Recall: 0.759826 , F-Score: 0.715214
```

As shown, position based precision and recall are better. However, cosine similarity predicts better only for some classes but worse in others. For example, in “bridge” class, the precision for (x,y) based is only 22/(22+16) = 0.579 compared to 48/(48+8) = 0.857 in cosine similarity. Also in “plank” class, the recall in (x,y) based is 49/(49+17) = 0.742 compared to just 29/(29+37) = 0.439. This shows that in cosine similarity based, it can better distinguish between classes like “bridge”, “childs” and “downwarddog”. For example when the truth class is “childs”, cosine similarity only predicted 1 class as “bridge” instead of 23 (from positional). Although, there are more FP in other classes present than position based. This shows that cosine similarity only works better for some positions.