Sunday, February 14, 2021

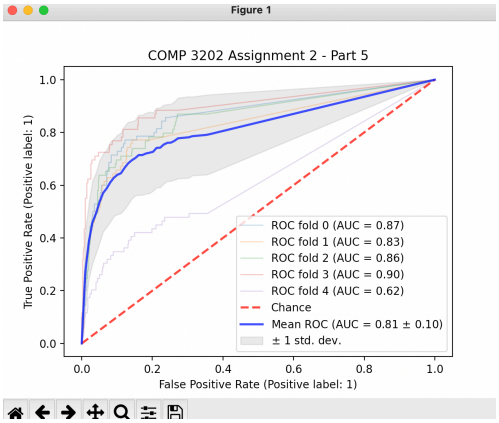
CV and Performance Curves

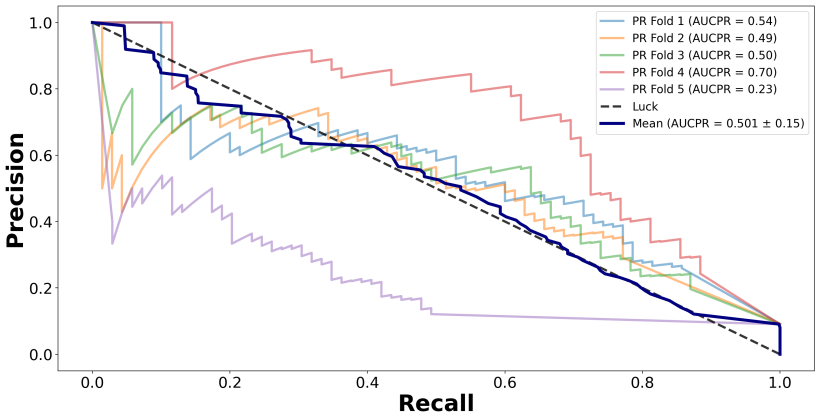
CS3202

1. ~~**A brief description and justification of the grid and performance metric you used for grid-search cv.**~~ To avoid overfitting, we created our grid-search cv with the KNeighborsClassifier class as our estimator, then passing in the grid parameters which maps to the number of neighbors (5,11,19,29), the weights (uniform, distance), and the metric (euclidean, manhattan). In addition to using f1 as our scoring parameter; f1 provides more precise measurements because it considers more than just accuracy.
2. ~~**Best parameter setting**~~ (metric: manhattan, n_neighbors: 11, weights: distance)
3. ~~**the cv grid scores**~~

```
{'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance'}
    mean_fit_time  std_fit_time  mean_score_time  std_score_time param_metric param_n_neighbors  ... split2_test_score split3_test_score  split4_test_score  mean_test_score  std_test_score  rank_test_score
0        0.003030      0.000707         0.049384        0.006881    euclidean                 5  ...          0.421053          0.593750           0.290323         0.439146        0.098348                9
1        0.002239      0.000142         0.032074        0.001990    euclidean                 5  ...          0.428571          0.593750           0.300000         0.450213        0.097764                6
2        0.002387      0.000301         0.045485        0.001249    euclidean                11  ...          0.421053          0.598291           0.275862         0.434247        0.106211               11
3        0.002174      0.000065         0.031408        0.001832    euclidean                11  ...          0.434783          0.629032           0.245614         0.451155        0.127962                4
4        0.002336      0.000276         0.044908        0.001228    euclidean                19  ...          0.363636          0.500000           0.228070         0.384982        0.095829               16
5        0.002105      0.000050         0.031349        0.001445    euclidean                19  ...          0.392157          0.556522           0.250000         0.415677        0.105151               14
6        0.002015      0.000029         0.045696        0.002268    euclidean                29  ...          0.400000          0.538462           0.254545         0.407055        0.097555               15
7        0.002037      0.000078         0.031099        0.001039    euclidean                29  ...          0.391753          0.589286           0.254545         0.421589        0.121054               13
8        0.001999      0.000026         0.116341        0.000918    manhattan                 5  ...          0.448276          0.598425           0.285714         0.450953        0.103577                5
9        0.002012      0.000056         0.103778        0.001448    manhattan                 5  ...          0.444444          0.587302           0.288288         0.455235        0.097400                3
10       0.002116      0.000157         0.116608        0.001587    manhattan                11  ...          0.368932          0.637931           0.278261         0.450070        0.122708                7
11       0.002025      0.000101         0.104284        0.001130    manhattan                11  ...          0.464286          0.661017           0.280702         0.476685        0.122805                1
12       0.002009      0.000118         0.116021        0.001249    manhattan                19  ...          0.400000          0.612613           0.254545         0.438588        0.121452               10
13       0.001954      0.000015         0.102717        0.000543    manhattan                19  ...          0.396040          0.666667           0.261682         0.458903        0.135843                2
14       0.001919      0.000011         0.115862        0.000590    manhattan                29  ...          0.421053          0.580000           0.245283         0.424115        0.110908               12
15       0.001928      0.000012         0.102943        0.000469    manhattan                29  ...          0.428571          0.603774           0.264151         0.444596        0.109481                8
```

4. ~~**the ROC and PR curves (make sure that your figures have axis labels**~~ **and captions)**





5. ~~**brief answers to the questions given in the next section**~~

    1. ~~**Did you normalize some of the attributes? Why? If yes, which attributes?**~~ No, the attributes required by KNN are already normalized.

2. **~~Would it make sense to expand your grid (i.em explore other values for the hyper-parameters)? Why? If yes, which values would you include?~~** No, we have the optimal range for our grid, expanding would lead to possible overfitting.
3. **~~Did you use stratified cross-validation? why?~~** Yes, because we have a binary target, we want to have an equal mean response value among each fold.
4. **~~Looking at the ROC and PR curves, where would you recommend to have the threshold for predicting positives (you can indicate the point(s) using coordinates referring to your plots)? why?~~** I'd recommend a threshold at (0.75,0.4) as the true positive rate begins to plateau at that point.
5. **~~Which graphical representation of performance (ROC or PR curve) is more suitable for this task?~~** ROC is more suitable as each fold returns a better AUC value compared to PR.

6. **~~An acknowledgement section listing your collaborations and online sources~~**
    1. https://scikit-learn.org/
7. **~~A program specification section~~**

Program Specification

1. Version Python 3.9.1
2. sklearn.model_selection (GridSearchCV)
3. sklearn.neighbors (KNeighborsClassifier)
4. numpy
5. pandas