**04: ML Concepts II, KNN**

**Functions & Concepts**

|  |  |  |  |
| --- | --- | --- | --- |
| **R** | **Python** | **Purpose** | **Differences** |
| **-** | List comprehension | Reduce the overhead and spatial memory associated with for loops | [Documentation](https://www.w3schools.com/python/python_lists_comprehension.asp) |

**Changes: KNN Notes**

|  |  |
| --- | --- |
| **Difference** | **Change** |
| Missing data visualization | Instead of `md.pattern` in R, we use seaborn to create a different visualization. It does not include value counts, but this is something that can be included by the student. Here’s a template code block:  import seaborn as sns  sns.displot(  data=df.isna().melt(value\_name="missing"),  y="variable",  hue="missing",  multiple="fill",  aspect=1.25  ) |
| Normalize numeric data | In R, use a built-in normalization function.  In Python, we use an equivalent Min-Max scaler from sklearn. Here’s a sample code block:  from sklearn import preprocessing  scaler = preprocessing.MinMaxScaler()  d = scaler.fit\_transform(df[numeric\_cols]) # conduct data transformation  scaled\_df = pd.DataFrame(d, columns=numeric\_cols) # convert back to pd df; transformation converts to array  df[numeric\_cols] = scaled\_df # put data back into the main df |
| Onehot encode | To put factorized variables in a format usable by the model  In R, use one\_hot.  In Python, use pd.get\_dummies() |
| Split into train/test | In R, createDataPartition()  In Python, train\_test\_split() |
| KNN Classifier | In R, knn()  In Python, [KNeighborsClassifier()](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html) |
| Viewing KNN results/output | In R, just view the model output by calling it by name.  In Python, we need to split it into a couple steps. Plot a confusion matrix with sklearn.metrics.plot\_confusion\_matrix(). Then, print other output with sklearn.metrics.classification\_report().  We don’t get sensitivity and specificity, so calculate this using TP/(TP+FN) for sensitivity and TN/(TN+FP) for specificity. |
| chooseK() function | Look at the KNN class notes file for more details; extensive comments are provided. |
| Probability density graph | In R, densityplot()  In Python, sns.displot() |
| Log Loss, F1 Score | In R, access with library(MLmetrics)  In Python, they’re built-in to sklearn.metrics |
| ROC/AUC curve | See KNN class notes file; instructive example and comments |
| Scaling numeric data | In R, scale() function  In Python, [sklearn.preprocessing.scale()](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scale.html) |
| Cross validation | In R, it’s built into the training function.  In Python, you:   1. Initialize a RepeatedKFold() object 2. Initialize your classifier 3. Use cross\_val\_score(classifier, X, y, scoring='accuracy', cv=object, jobs=-1)   For a more complex graph of testing and training accuracy, refer to KNN class notes |
| Variable importance | SKLearn does not calculate variable importance for KNN, so we do this ourselves by seeing how much changes to each variable affect the model.   1. Create a df to hold the data 2. Calculate the baseline accuracy 3. Copy the df 4. Permute one variable 5. Run the classifier on the new df 6. Add baseline-permuted accuracy to df 7. Repeat 4-6 until all variables are calculated.   You can do this in a loop or function! |

**Changes: Graduation Lab**

|  |  |
| --- | --- |
| **Difference** | **Change** |
| Indexing | Python indexing starts at 0 and R begins at 1. If you are converting between R and Python, make sure you update your column or row indices!! |