# CSE 572: Lab 10

In this lab, you will practice implementing techniques for model selection including cross validation and grid search.

To execute and make changes to this notebook, click File > Save a copy to save your own version in your Google Drive or Github. Read the step-by-step instructions below carefully. To execute the code, click on each cell below and press the SHIFT-ENTER keys simultaneously or by clicking the Play button.

When you finish executing all code/exercises, save your notebook then download a copy (.ipynb file). Submit the following **three** things:

1. a link to your Colab notebook,
2. the .ipynb file, and
3. a pdf of the executed notebook on Canvas.

To generate a pdf of the notebook, click File > Print > Save as PDF.

```
# Import libraries
import numpy as np
import pandas as pd

# Set the random seed for reproducibility
seed = 0
np.random.seed(0)
```

## Load the iris dataset

```
data = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None)
data.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'class']

data.sample(5, random_state=seed)
```

| | sepal length | sepal width | petal length | petal width | class |
|---|---|---|---|---|---|
| **114** | 5.8 | 2.8 | 5.1 | 2.4 | Iris-virginica |
| **62** | 6.0 | 2.2 | 4.0 | 1.0 | Iris-versicolor |
| **33** | 5.5 | 4.2 | 1.4 | 0.2 | Iris-setosa |
| **107** | 7.3 | 2.9 | 6.3 | 1.8 | Iris-virginica |
| **7** | 5.0 | 3.4 | 1.5 | 0.2 | Iris-setosa |

```
data.shape
```

```
(150, 5)
```

Standardize the data by subtracting the feature-wise mean and dividing by the feature-wise standard deviation for each sample.

```
# YOUR CODE HERE
SLMean = data['sepal length'].mean()
SLStd = data['sepal length'].std()
data['sepal length'] = (data['sepal length'] - SLMean) / SLStd

SWMean = data['sepal width'].mean()
SWStd = data['sepal width'].std()
data['sepal width'] = (data['sepal width'] - SWMean) / SWStd

PLMean = data['petal length'].mean()
PLStd = data['petal length'].std()
data['petal length'] = (data['petal length'] - PLMean) / PLStd

PWMean = data['petal width'].mean()
PWStd = data['petal width'].std()
data['petal width'] = (data['petal width'] - PWMean) / PWStd



data.sample(5, random_state=seed)
```

| | sepal length | sepal width | petal length | petal width | class |
|---|---|---|---|---|---|
| **114** | -0.052331 | -0.585801 | 0.760212 | 1.574155 | Iris-virginica |
| **62** | 0.189196 | -1.969583 | 0.136778 | -0.260321 | Iris-versicolor |

## k-fold Cross validation

We will use 5-fold cross validation to train and evaluate our classifier. We will not do any model selection/hyperparameter tuning in this step, so we need to split our data into a training and test set.

To split the data into 5 folds we will shuffle the rows and then split them into $k$ equal groups.

```
k = 5

# Note: np.split raises error if indices_or_sections is
# an integer and doesn't result in equal size splits
folds = np.split(data.sample(frac=1, random_state=seed), indices_or_sections=k)
```

Use a for loop to print the number of samples and number of samples from each class in each fold.

```
# YOUR CODE HERE
for i, fold in enumerate(folds):
  print('Fold {} contains a total of {} instances \n 1. Setosa: {}, \n 2. virginica: {},\n 3. versicolor: {} '.format(i+1,
                                                                                                                      fold.shape[0],
                                                                                                                      fold[fold['class'
                                                                                                                      fold[fold['class'
                                                                                                                      fold[fold['class'
```

```
Fold 1 contains a total of 30 instances
 1. Setosa: 11,
 2. virginica: 6,
 3. versicolor: 13
Fold 2 contains a total of 30 instances
 1. Setosa: 5,
 2. virginica: 15,
 3. versicolor: 10
Fold 3 contains a total of 30 instances
 1. Setosa: 10,
 2. virginica: 10,
 3. versicolor: 10
Fold 4 contains a total of 30 instances
 1. Setosa: 14,
 2. virginica: 10,
 3. versicolor: 6
Fold 5 contains a total of 30 instances
 1. Setosa: 10,
 2. virginica: 9,
 3. versicolor: 11
```

## Train a k Nearest Neighbors classifier

We will use the KNeighborsClassifier in sklearn for our classification model. Use cross validation to train and evaluate the model. Set hyperparameters to `n_neighbors=5`, `metric='l2'`, and `weights='uniform'`.

Implement a for loop to iterate through each fold, training a new kNN model each iteration with one fold assigned to validation and the remaining folds assigned to training. Compute the validation accuracy for each iteration and append it to the `accuracies` list.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

accuracies = []

# YOUR CODE HERE
for i in range(len(folds)):
  foldedValues = folds[i]
  train = pd.concat(folds[0:i] + folds[i+1:])
  knn = KNeighborsClassifier(n_neighbors=5, metric='l2', weights='uniform')
  knn.fit(train[train.columns[:-1]], train['class'])
  pred_val = knn.predict(foldedValues[foldedValues.columns[:-1]])
  accuracies.append(accuracy_score(foldedValues['class'], pred_val))
```

Print the mean and standard deviation of the accuracy from cross validation (across all $k$ folds).

```
print('Mean Acc: {}'.format(np.mean(accuracies)))
print('Standard deviation of accuracy: {}'.format(np.std(accuracies)))
```

```
Mean Acc: 0.9533333333333334
Standard deviation of accuracy: 0.0581186525805423
```

**Question 1: If you increased the number of folds, do you expect the standard deviation of the accuracy across $k$ folds to increase or decrease? Why?**

**Answer:**

YOUR ANSWER HERE

"-> which means"

More folds -> Smaller Datasets in each fold -> More variance among fold to fold.

Hence Standard Deviation would increase with more folds. Model will be overfit and will not generalize well.

## Hyperparameter selection using cross validation and grid search

In this exercise, we will use the KNeighborsClassifier again but this time we will perform hyperparameter selection using k-fold cross validation and Grid Search.

We have three model choices (hyperparameters) for our kNN model:

- Number of neighbors ($k$ or `n_neighbors`). We will consider all integer values $k \in [1, 10]$.
- Whether to treat all neighbors equally when taking majority vote, or weight them according to their distance from the query point (`weights='uniform'` or `weights='distance'`).
- The distance metric for computing distance between query point and neighbors (`metric` argument). We will consider three options for `metric`: `'l1'`, `'l2'`, and `'cosine'`.

**Question 2: How many total combinations of the above hyperparameter choices are there?**

**Answer:**

YOUR ANSWER HERE 60

10(Values of Neghbours) * 2(weights) * 3(Distance Metrics)

Instead of implementing cross validation manually as we did in the previous example, we will use the GridSearchCV class in sklearn to perform grid search and cross validation simultaneously.

First, we will split the data into a training (70%) and test (30%) test.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data[data.columns[:-1]],
                                                    data['class'],
                                                    test_size=0.3,
                                                    random_state=seed)
```

We will then use the training set for cross validation and grid search to select the optimal hyperparameter settings.

Next, we define the values for grid search using a dictionary in which the keys are the parameter names to be passed to the model function and each corresponding value is a list of possible values to try in grid search.

```
param_grid = {'n_neighbors': list(range(1, 11)),
              'weights': ['uniform', 'distance'],
              'metric': ['l1', 'l2', 'cosine']
             }
```

Next, we instantiate a kNeighborsClassifier but do not specify the hyperparameter settings yet.

```
knn = KNeighborsClassifier()
```

We can then pass this classifier and our parameter grid to a new GridSearchCV object and fit the GridSearchCV using our training data.

```
from sklearn.model_selection import GridSearchCV

clf = GridSearchCV(knn, param_grid)
```

```
clf.fit(X_train, y_train)
     GridSearchCV(estimator=KNeighborsClassifier(),
                  param_grid={'metric': ['l1', 'l2', 'cosine'],
                              'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                              'weights': ['uniform', 'distance']})
```

The cross validation results are stored as an attribute of the GridSearchCV object as a dictionary with keys as column headers and values as columns, that can be imported into a pandas DataFrame.

```
cv_results = pd.DataFrame(clf.cv_results_)
```

```
cv_results
```

| | | | | | |
|---|---|---|---|---|---|
| **49** | 0.001356 | 0.000338 | 0.001487 | 0.000115 | cosine |
| **50** | 0.001198 | 0.000022 | 0.001745 | 0.000097 | cosine |
| **51** | 0.001380 | 0.000300 | 0.001542 | 0.000195 | cosine |
| **52** | 0.001371 | 0.000111 | 0.002245 | 0.000626 | cosine |
| **53** | 0.001222 | 0.000020 | 0.001451 | 0.000033 | cosine |
| **54** | 0.002025 | 0.000776 | 0.002627 | 0.001025 | cosine |
| **55** | 0.001264 | 0.000048 | 0.001459 | 0.000031 | cosine |
| **56** | 0.001214 | 0.000021 | 0.001778 | 0.000023 | cosine |
| **57** | 0.001440 | 0.000247 | 0.001749 | 0.000296 | cosine |
| **58** | 0.001260 | 0.000116 | 0.001873 | 0.000106 | cosine |
| **59** | 0.001412 | 0.000390 | 0.001632 | 0.000135 | cosine |