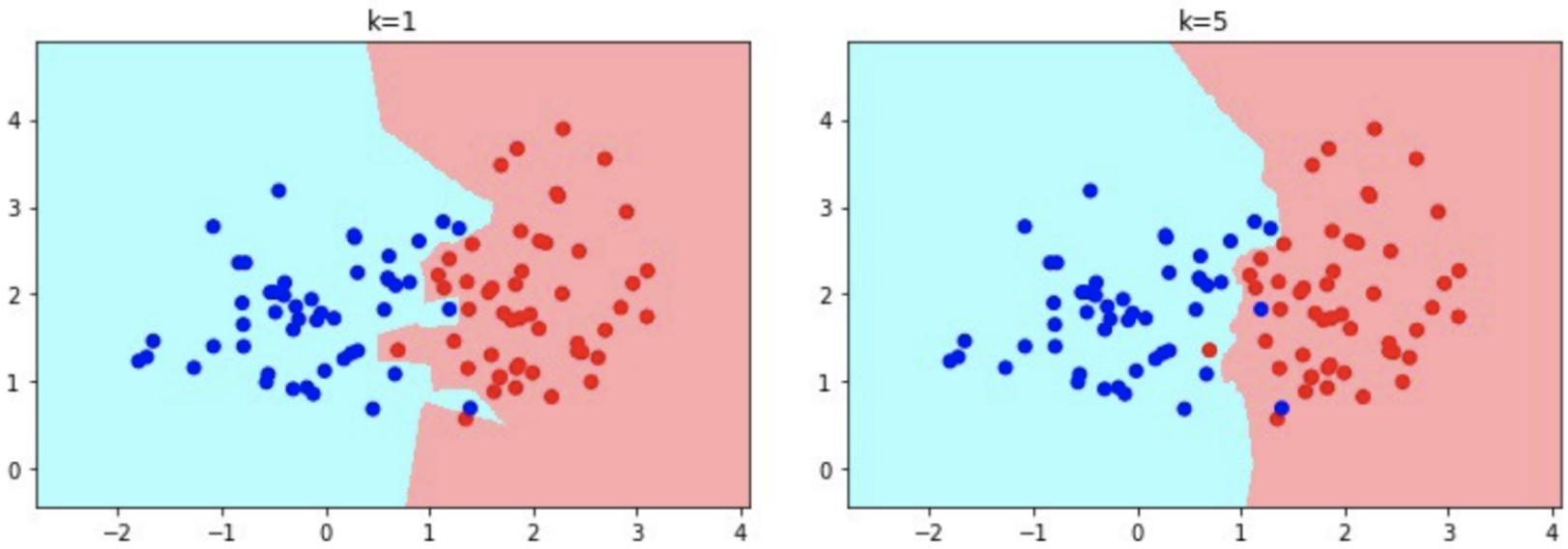
Intro to classification - kNN - 5

One should look for what is and not what he thinks should be. (Albert Einstein)

What Happens when K Changes?



Decision Surface separating the red and blue class with k=1 (left) and k=5 (right). Image by author

Source

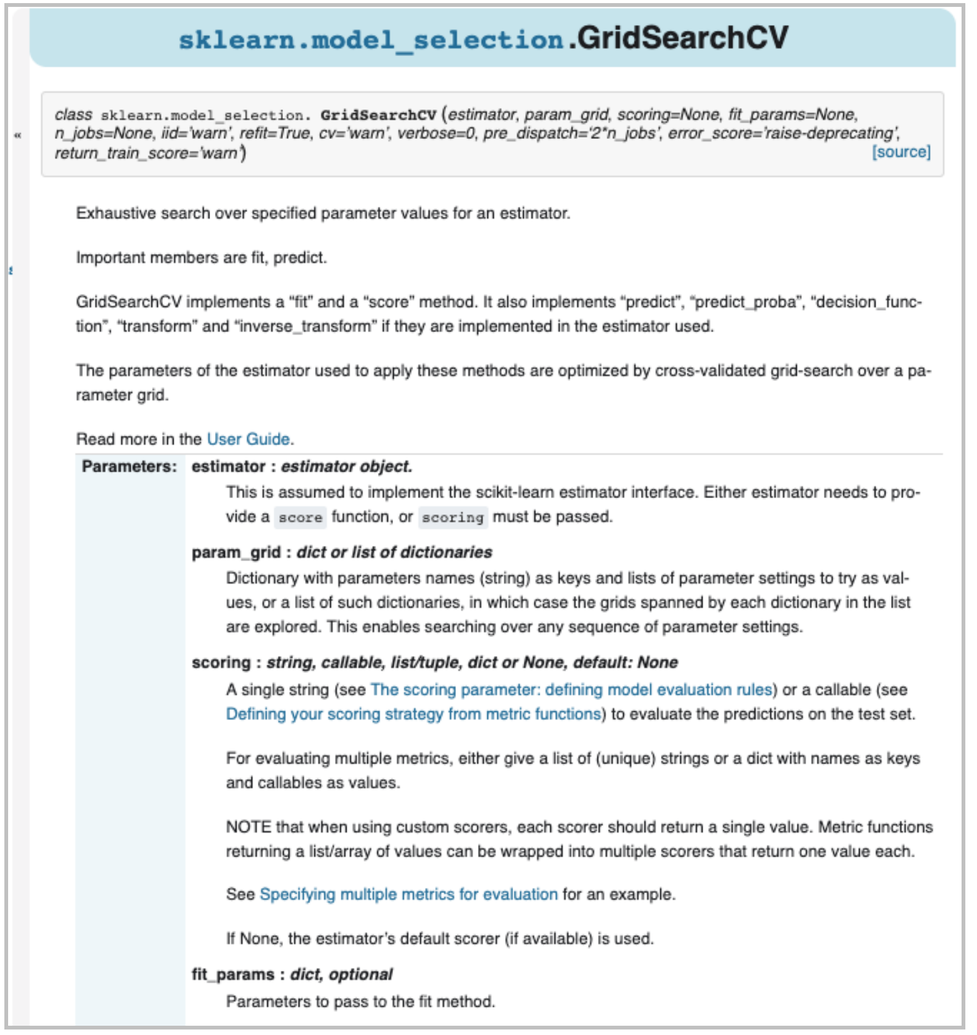
- What are the concerns with choosing the optimal K?
- What do you notice about the two choices above?
- Put your observations in the chat

Module completion checklist

Objective	Complete
Use GridSearchCV to find the optimal number of nearest neighbors and define optimal model	
Discuss reasons we would or would not use kNN	

GridSearchCV for optimal hyperparameters

- Once again, `GridSearchCV` is different from cross-validation because it performs an exhaustive search over a range of specified hyperparameter values
- Let's now learn a little more about `GridSearchCV`



sklearn.model_selection.GridSearchCV

class sklearn.model_selection. **GridSearchCV** (*estimator, param_grid, scoring=None, fit_params=None, n_jobs=None, iid='warn', refit=True, cv='warn', verbose=0, pre_dispatch='2*n_jobs', error_score='raise-deprecating', return_train_score='warn'*) [source]

Exhaustive search over specified parameter values for an estimator.

Important members are fit, predict.

GridSearchCV implements a "fit" and a "score" method. It also implements "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used.

The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Read more in the User Guide.

Parameters:

estimator : *estimator object.*
This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

param_grid : *dict or list of dictionaries*
Dictionary with parameters names (string) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

scoring : *string, callable, list/tuple, dict or None, default: None*
A single string (see The scoring parameter: defining model evaluation rules) or a callable (see Defining your scoring strategy from metric functions) to evaluate the predictions on the test set.

For evaluating multiple metrics, either give a list of (unique) strings or a dict with names as keys and callables as values.

NOTE that when using custom scorers, each scorer should return a single value. Metric functions returning a list/array of values can be wrapped into multiple scorers that return one value each.

See Specifying multiple metrics for evaluation for an example.

If None, the estimator's default scorer (if available) is used.

fit_params : *dict, optional*
Parameters to pass to the fit method.

Finding optimal k

- We will now use the `GridSearchCV` function to find the optimal number of `k`
- As you walk through the steps, keep in mind that this is a method that we can use to tune various hyperparameters depending on the model
- Let's look at the steps below:

1. Define the hyperparameter value (in this case, we are defining the range of `k`)
2. Create a hyperparameter grid that maps the values to be searched
3. Fit the grid with data from the model
4. View the scores and choose the optimal hyperparameter
5. View the hyperparameters of the "best model"

Finding optimal k - GridSearchCV

```
# Define the parameter values that should be searched.
k_range = list(range(1, 31))

# Create a parameter grid: map the parameter names to the values that should be searched by building
a Python dictionary.
# key: parameter name
# value: list of values that should be searched for that parameter
# single key-value pair for param_grid
param_grid = dict(n_neighbors = k_range)
print(param_grid)

# Instantiate the grid using our original model - kNN with k.
```

```
{'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30]}
```

```
grid = GridSearchCV(kNN, param_grid, cv = 10, scoring = 'accuracy')
```

Finding optimal k - GridSearchCV

- Again, we need to make sure to scale before our gridsearch

```
# Create a pipeline of the scaler and gridsearch
grid_search_pipeline = Pipeline([('transformer',  StandardScaler()), ('estimator', grid)])

# Fit Gridsearch pipeline
grid_search_pipeline.fit(X, y)
```

```
Pipeline(steps=[('transformer', StandardScaler()),
                ('estimator',
                 GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
                              param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7,
                                                          8, 9, 10, 11, 12, 13,
                                                          14, 15, 16, 17, 18,
                                                          19, 20, 21, 22, 23,
                                                          24, 25, 26, 27, 28,
                                                          29, 30]},
                              scoring='accuracy'))])
```

Finding optimal k - view results

```
# View the complete results (list of named tuples).
print(grid.cv_results_['mean_test_score'])
```

```
[0.91643836 0.94637965 0.94168297 0.94990215 0.9481409  0.95009785
 0.94951076 0.95088063 0.95029354 0.95107632 0.95068493 0.95088063
 0.95068493 0.95107632 0.95107632 0.95107632 0.95107632 0.95127202
 0.95127202 0.95127202 0.95127202 0.95127202 0.95127202 0.95127202
 0.95127202 0.95127202 0.95127202 0.95127202 0.95127202 0.95127202]
```

Finding optimal k

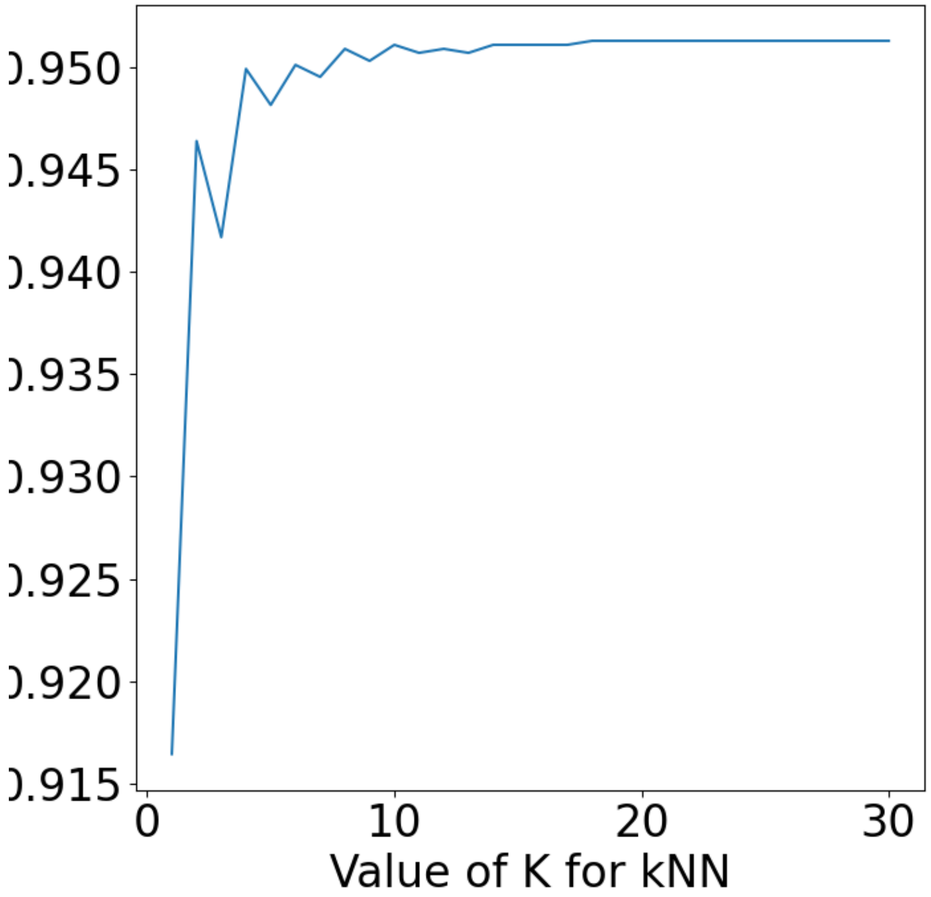
- First, we create a list of mean scores

```
# Create a list of the mean scores only by using a list comprehension to loop through
grid.cv_results_.
grid_mean_scores = [result for result in grid.cv_results_['mean_test_score']]
print(grid_mean_scores)
```

```
[0.9164383561643836, 0.9463796477495107, 0.941682974559687, 0.9499021526418787,
0.9481409001956946, 0.9500978473581212, 0.9495107632093933, 0.950880626223092,
0.9502935420743638, 0.9510763209393346, 0.9506849315068493, 0.950880626223092,
0.9506849315068493, 0.9510763209393346, 0.9510763209393346, 0.9510763209393346,
0.9510763209393346, 0.9512720156555773, 0.9512720156555773, 0.9512720156555773,
0.9512720156555773, 0.9512720156555773, 0.9512720156555773, 0.9512720156555773,
0.9512720156555773, 0.9512720156555773, 0.9512720156555773, 0.9512720156555773,
0.9512720156555773, 0.9512720156555773]
```

Finding optimal k - plot

```
# Plot the results.
_ = plt.plot(k_range, grid_mean_scores)
_ = plt.xlabel('Value of K for kNN')
_ = plt.ylabel('Cross-Validated Accuracy')
plt.show()
```



Define and examine the optimized model

- Now that we have found our optimal `k`, let's examine our results:
  - best accuracy score over all parameters of `k`
  - parameters that led to that score
  - model that we ran to achieve that score

```
# Single best score achieved across all params (k).
print(grid.best_score_)
```

```
0.9512720156555773
```

```
grid_score = grid.best_score_

# Dictionary containing the parameters (k) used to generate
that score.
print(grid.best_params_)

# Actual model object fit with those best parameters.
# Shows default parameters that we did not specify.
```

```
{'n_neighbors': 18}
```

```
print(grid.best_estimator_)
```

```
KNeighborsClassifier(n_neighbors=18)
```

Add GridSearchCV score to the final scores

- So now that we have it, let's add this score to the dataframe we created earlier

```
model_final = model_final.append({'metrics' : "accuracy" ,
                                  'values' : round(grid_score, 4),
                                  'model':'kNN_GridSearchCV' } ,
                                  ignore_index = True)
print(model_final)
```

```
    metrics  values              model
0  accuracy  0.9419              kNN_k
1  accuracy  0.9513  kNN_GridSearchCV
```

Optimal model and final thoughts

- We can use the model we just built to predict on the test set
- It is optimized because of the cross-validation and will use the optimized k
- Although it is an optimized model, it does not mean it will perform better than our previous model, it is just more accurate

```
kNN_best = grid.best_estimator_

# Check accuracy of our model on the test data.
print(kNN_best.score(X_test, y_test))
```

```
0.9458577951728636
```

```
kNN_champ = kNN_best.score(X_test, y_test)
```

Pickle - what?

- We are going to pause for a moment to learn about the library `pickle`
- When we have objects we want to carry over and do not want to rerun code, we can `pickle` these objects
- In other words, `pickle` will help us save objects from one script/ session and pull them up in new scripts
- How do we do that? We use a function in Python called `pickle`
- It is similar to **flattening** a file
  - **Pickle/saving: a Python object is converted into a byte stream**
  - **Unpickle/loading: the inverse operation where a byte stream is converted back into an object**

Final Model

- Let’s save our final model in case we need to use it another time.

```
# Save this final model
model_final = {'metrics' : "accuracy" ,
                                'values' : round(kNN_champ, 4),
                                'model':'kNN_optimized' }
print(model_final)
```

```
{'metrics': 'accuracy', 'values': 0.9459, 'model': 'kNN_optimized'}
```

```
pickle.dump(model_final, open(str(data_dir)+"/"+"model_final.sav", 'wb'))
```

Module completion checklist

Objective	Complete
Use GridSearchCV to find the optimal number of nearest neighbors and define optimal model	✔
Discuss reasons we would or would not use kNN	

kNN pros and cons

Pros

- Easy to use
- Can easily handle multiple categories
- Many options to adjust (features, measurement metric, etc.)

Cons

- Many options to adjust
- The correct distance metric is important
- Can be slow with large amounts of data
- Gets less accurate with more predictors (will not be accurate on larger datasets)
- It is LAZY–it memorizes and uses every data point when calculating kNN

Knowledge check

Exercise

You are now ready to try Tasks 15-19 in the Exercise for this topic

Module completion checklist

Objective	Complete
Use GridSearchCV to find the optimal number of nearest neighbors and define optimal model	✔
Discuss reasons we would or would not use kNN	✔

kNN : Topic summary

In this part of the course, we have covered:

- Supervised learning and its use cases
- The theory behind kNN algorithm
- Implementation of kNN on a dataset
- Performance optimization for kNN

Congratulations on completing this module!