Machine learning - Part 3

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

Welcome back!

In the last class we learned about

- classification and its use cases
- summary and applications of knn algorithm
- implementation of the knn algorithm on training data
- cross-validation and its use cases

Today we will

- apply cross-validation to understand what is the optimal model accuracy
- learn about hyperparameters and GridSearch
- take a look into logistic regression ad its applications

Warm-up

Check out this ***list*** of cool machine learning projects that you might want to try out in your free time.

Questions to consider:

- Did you find any of these especially interesting? Why?
- Are any of these projects similar to how you wold use machine learning for work?

Module completion checklist

Objective	Complete
Perform cross-validation to understand better what the optimal model accuracy might be	
Find optimal hyperparameters with a grid search	
Use GridSearchCV to find optimal number of nearest neighbors and define optimal model	
Discuss reasons we would or would not use kNN	
Determine when to use logistic regression for classification and transformation of target variable	
Summarize the process and the math behind logistic regression	

Loading packages

Let's load the packages we will be using:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# New today - we will introduce it when we use it
import pickle
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import scale
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import metrics

# Scikit-learn package for logistic regression.
from sklearn import linear_model
```

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your `skillsoft-machine-learning-2021` folder
- `data_dir` be the variable corresponding to your `data` folder

```
# Set 'main_dir' to location of the project folder
from pathlib import Path
home_dir = Path(".").resolve()
main_dir = home_dir.parent
print(main_dir)
```

```
data_dir = str(main_dir) + "/data"
print(data_dir)
```

Loading data into Python

- Let’s load the clean dataset that we saved in the previous module

```
costa_knn_cleaned = pd.read_csv(data_dir + '/costa_knn_cleaned.csv')
print(costa_knn_cleaned.head())
```

```
  household_id  rooms  num_adults  Target
0    21eb7fcc1      3           1    True
1    0e5d7a658      4           1    True
2    2c7317ea8      8           1    True
3    2b58d945f      5           2    True
4    2b58d945f      5           2    True
```

Data prep: scaling variables

- As discussed in our previous module, let's scale our predictors, we split our data into `X` and `y`

```
# Split the data into X and y - y is categorical, so can't scale.
X = costa_knn_cleaned[['rooms', 'num_adults']]
y = np.array(costa_knn_cleaned['Target'])

# Scale X.
X_scaled = scale(X)
print(X_scaled[0:5])
```

```
[[-1.33182893 -1.3657179 ]
 [-0.65077114 -1.3657179 ]
 [ 2.07346003 -1.3657179 ]
 [ 0.03028665 -0.5080948 ]
 [ 0.03028665 -0.5080948 ]]
```

Train & test: small scale before n-fold

- Before we actually use n-fold cross-validation:
  - We split our data into a train and test set
  - We run kNN initially on the training data

```
# Set the seed.
np.random.seed(1)

# Split into train and test.
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
                                                    y,
                                                    test_size = 0.3)
```

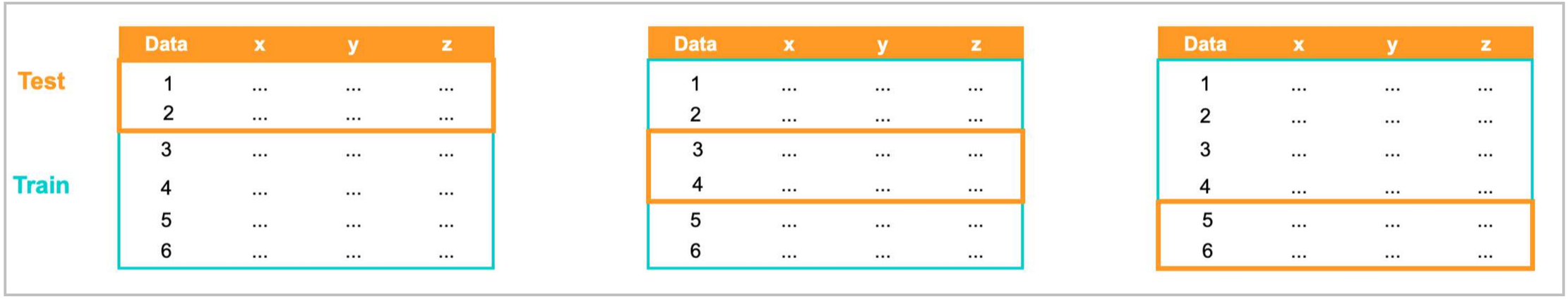
Finding optimal k

1. **We have now:**
2. Run kNN on our training data
3. Predicted using the kNN model, on our test data
4. Reviewed performance metrics for classification algorithms
5. Built a confusion matrix for the predicted model
6. **We will now:**
7. Use cross-validation and re-run the model on the training data
8. Find what our optimal k value is
9. Evaluate the new predictions

Cross-validation: n-fold

To quickly refresh ourselves, here is how cross-validation works:

1. Split the dataset into several subsets (“n” number of subsets) of equal size
2. **Use each subset as the test dataset** and **use the rest of the data as the training dataset**
3. Repeat the process for every subset you create



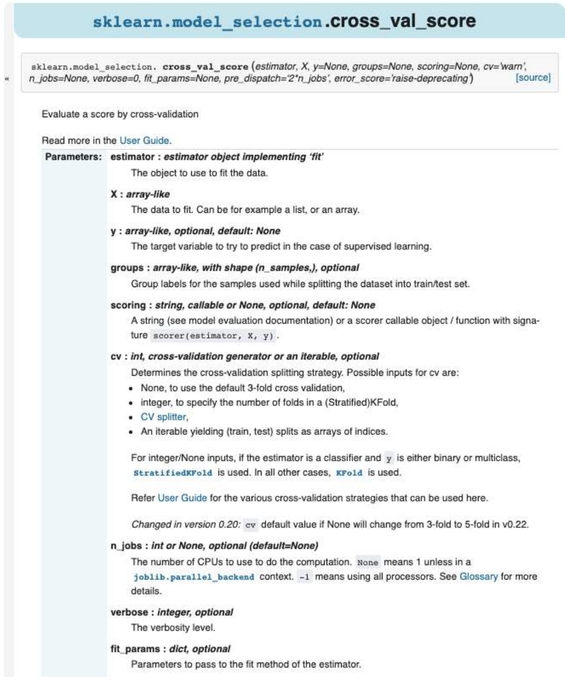
Cross-validation helps prevent overfitting by performing the model on multiple subsets

Cross-validation in Python

- We will be using two functions from `sklearn.model_selection` for cross-validation:
  - `cross_val_score`
  - `GridSearchCV`
- They are both model selection / optimization functions that use cross-validation and they are both part of the `sklearn.model_selection` package
  - `cross_val_score` will help us find the potential optimal model score
  - `GridSearchCV` will run through cross-validation multiple times for a range of given `k` and help us find the optimal value

cross_val_score for optimal accuracy

- Here is a little about `cross_val_score`



sklearn.model_selection.cross_val_score

`sklearn.model_selection.` **`cross_val_score`**(*estimator, X, y=None, groups=None, scoring=None, cv='warn', n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score='raise-deprecating'*) [source]

Evaluate a score by cross-validation

Read more in the User Guide.

Parameters:

estimator : ***estimator object implementing 'fit'***
The object to use to fit the data.

X : ***array-like***
The data to fit. Can be for example a list, or an array.

y : ***array-like, optional, default: None***
The target variable to try to predict in the case of supervised learning.

groups : ***array-like, with shape (n_samples,), optional***
Group labels for the samples used while splitting the dataset into train/test set.

scoring : ***string, callable or None, optional, default: None***
A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)`.

cv : ***int, cross-validation generator or an iterable, optional***
Determines the cross-validation splitting strategy. Possible inputs for cv are:
- None, to use the default 3-fold cross validation,
- integer, to specify the number of folds in a (Stratified)KFold,
- CV splitter,
- An iterable yielding (train, test) splits as arrays of indices.

For integer/None inputs, if the estimator is a classifier and `y` is either binary or multiclass, `StratifiedKFold` is used. In all other cases, `KFold` is used.

Refer User Guide for the various cross-validation strategies that can be used here.

Changed in version 0.20: `cv` default value if None will change from 3-fold to 5-fold in v0.22.

n_jobs : ***int or None, optional (default=None)***
The number of CPUs to use to do the computation. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

verbose : ***integer, optional***
The verbosity level.

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

Cross-validation for optimal accuracy

- First, let's use `cross_val_score` to understand what our optimal accuracy should be
- We'll take the following steps:
  - using our `knn` model, we train a model with a cross-validation reoccurrence of five times
  - we look at each cross-validation accuracy score
  - we then average the accuracy scores over the 5 times and we use that as our potential optimal model score

```
# Train model with CV of 5.
knn = KNeighborsClassifier(n_neighbors = 5)
cv_scores = cross_val_score(knn, X_scaled, y, cv = 5)
```

- Notice that we don't use our split data, the input is our entire `X_scaled` and `y`
- This is because the cross-validation occurs when the function holds out samples for us

Cross-validation for optimal accuracy

- Let's look at our output

```
# Print each cv score (accuracy) and average them.
print(cv_scores)
```

```
[0.59100418 0.56119247 0.60858189 0.59863946 0.59654631]
```

```
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

```
cv_scores mean:0.5911928627260367
```

```
mean = np.mean(cv_scores)
print("Optimal cv score is:", round(mean, 4))
```

```
Optimal cv score is: 0.5912
```

Exercise 1

Module completion checklist

Objective	Complete
Perform cross-validation to understand better what the optimal model accuracy might be	✔
Find optimal hyperparameters with a grid search	
Use GridSearchCV to find optimal number of nearest neighbors and define optimal model	
Discuss reasons we would or would not use kNN	
Determine when to use logistic regression for classification and transformation of target variable	
Summarize the process and the math behind logistic regression	

Finding optimal k

1. **We have now:**
2. Run kNN on our training data
3. Predicted using the kNN model on our test data
4. Reviewed performance metrics for classification algorithms
5. Built a confusion matrix for the predicted model
6. **We will now:**
7. Implement Grid search to know what our optimal k value is
8. Evaluate the new predictions

Parameters vs. hyperparameters

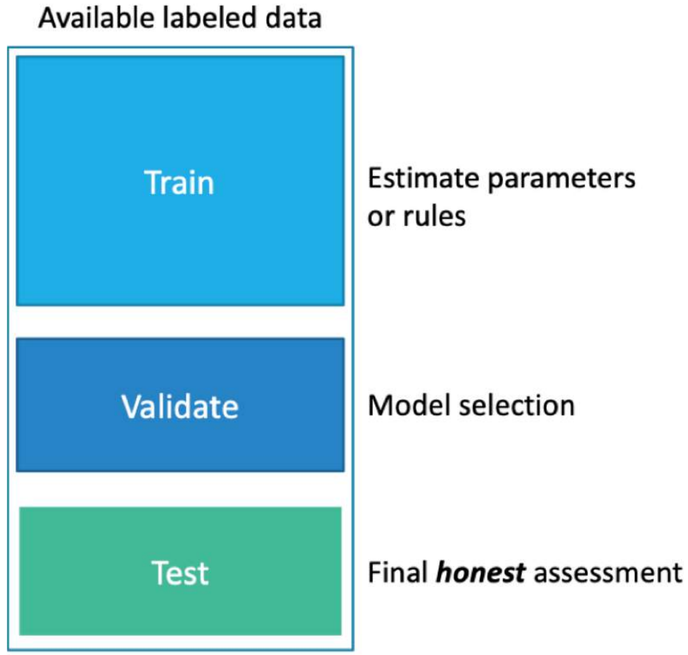
- **Parameters**
  - Parameters are derived from training data
  - Example: the weights of a predictor in a regression
  - They are learned by the algorithm from the data
- **Hyperparameters**
  - Hyperparameters are manually set before the training process
  - Example: k in kNN, number of trees in random forest, penalty in penalized regression
  - They are found using a **grid search**

What is a grid search?

- **Grid search helps us find the optimal hyperparameters**
  - Grid search allows us to search over a list of hyperparameter values to find the optimal hyperparameter
  - It is a brute force approach: it creates a model for every hyperparameter value in the list
  - We choose the hyperparameter value that created the model with the smallest error
- **Example kNN**
  - Grid search allows us to find the optimal `k`
  - We can use a grid search to search over k=1, k=2, k=3, k=4, etc.
  - Grid search creates models for every value of `k` in the list
  - We can choose the `k` that yields the smallest error

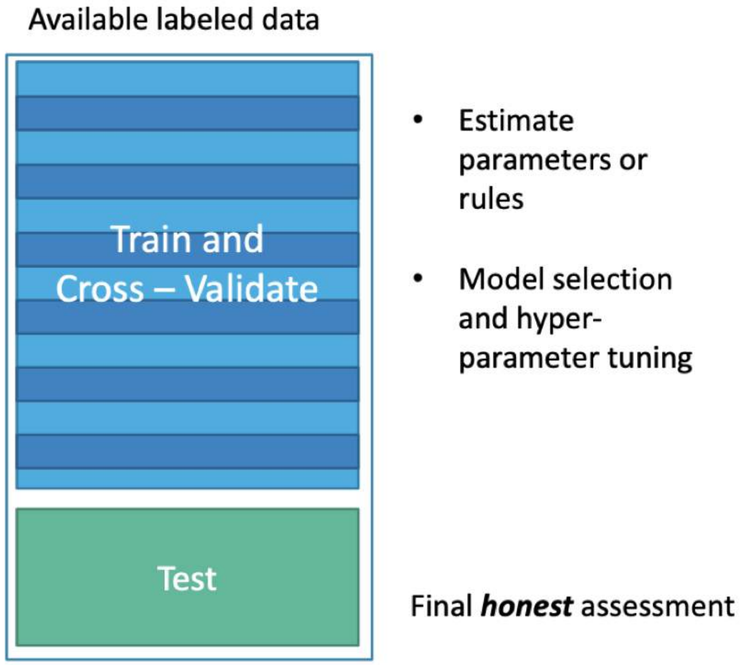
Finding the smallest error with grid search

- So far, we have used a **train - test split** to train and evaluate our models
- This worked because we only had parameters, but no hyperparameters
- Now that we also have hyperparameters in our models, we need a **train - validation - test split**
- The validation set allows us to compare the different models created by the grid search and choose the optimal hyperparameters



Using cross-validation with grid search

- Instead of using train - validation - test split, we can use **cross-validation**
- Cross-validation allows us to perform the split multiple times on the same dataset
- We have new train and validation sets for each fold `n`
- This leads to more accurate results, but is computationally intensive
- It is best suited for small datasets
- Today, we will use cross-validation with our grid search



Keeping test data separate

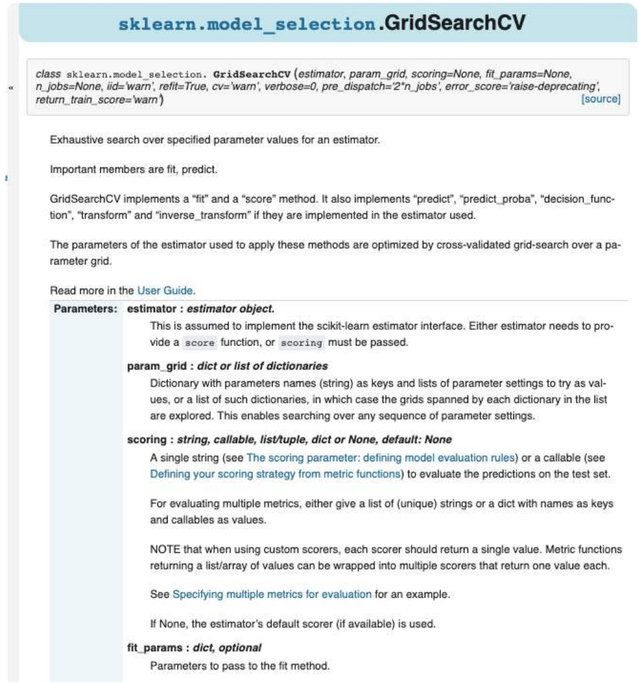
- Whether you use **train - validation - test** or **cross-validation**, you always want to have a separate test set
- The test set must not be involved in the model training and selection process to allow for a **true assessment**
- Only a true assessment tells you how your model will perform on previously unseen data
- The errors on your train and validation sets will be lower than the error you can expect on previously unseen data

Module completion checklist

Objective	Complete
Perform cross-validation to understand better what the optimal model accuracy might be	✔
Find optimal hyperparameters with a grid search	✔
Use GridSearchCV to find optimal number of nearest neighbors and define optimal model	
Discuss reasons we would or would not use kNN	
Determine when to use logistic regression for classification and transformation of target variable	
Summarize the process and the math behind logistic regression	

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 use the `GridSearchCV` function now so that we can 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 that 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 = 5.
```

```
{'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

```
# Fit the grid with data.
grid.fit(X_scaled, y)
```

```
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'])
```

```
[0.57549366 0.52829471 0.58543572 0.54923459 0.58741648 0.58333392
 0.59506309 0.57151613 0.5762216  0.57151493 0.58858014 0.57968433
 0.58690486 0.58104132 0.60436987 0.59223214 0.59935026 0.59171066
 0.60008248 0.60217497 0.61148853 0.61044207 0.60604843 0.60803742
 0.61578698 0.61400567 0.61954939 0.62059421 0.62676948 0.6177683 ]
```

Finding optimal k

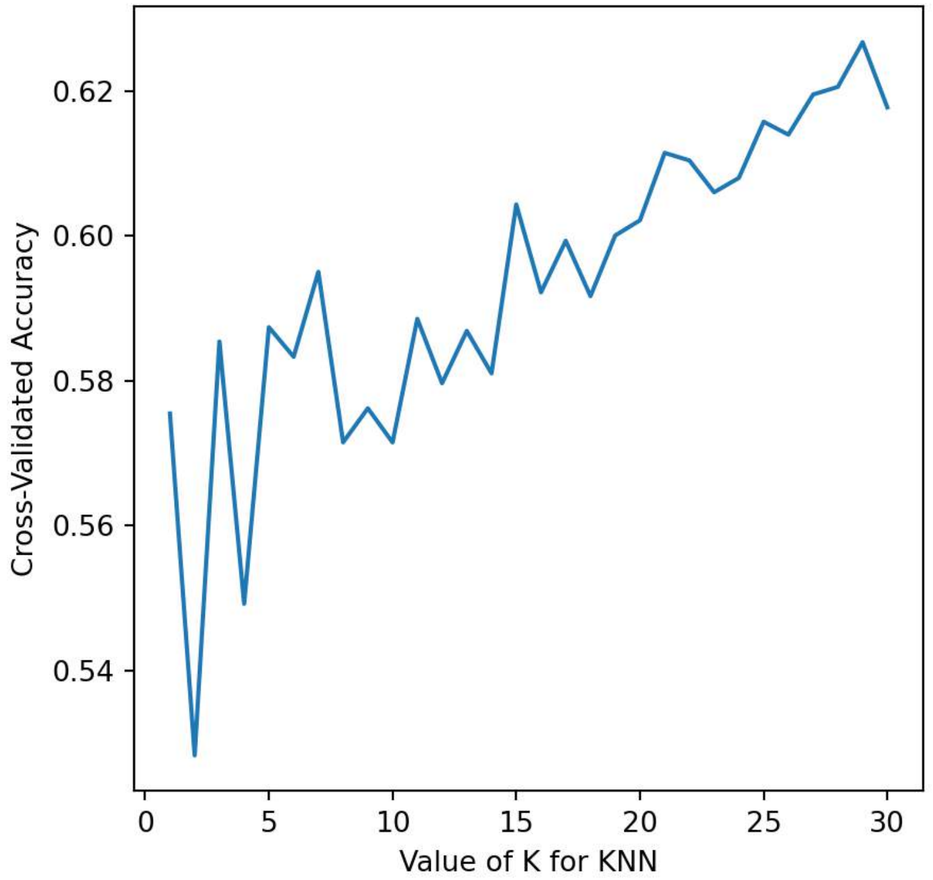
- First, we create a list of mean scores

```
# 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)
```

```
[0.5754936581305177, 0.5282947052509365, 0.5854357160069225, 0.5492345944051348,
0.5874164822887687, 0.5833339175009311, 0.5950630901005499, 0.5715161339788385,
0.5762216039781813, 0.5715149291331683, 0.5885801441433547, 0.5796843304344017,
0.5869048610046222, 0.5810413152533462, 0.6043698657144735, 0.5922321409012246,
0.5993502595894762, 0.5917106617888672, 0.600082477162698, 0.6021749654976014,
0.611488532059848, 0.6104420688295471, 0.606048434795943, 0.6080374159346317,
0.6157869832855046, 0.6140056737277926, 0.6195493877193367, 0.6205942079782689,
0.626769480163859, 0.6177682972244737]
```

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')
```



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 was run to achieve that score

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

```
0.626769480163859
```

```
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': 29}
```

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

```
KNeighborsClassifier(n_neighbors=29)
```

Add GridSearchCV score to the final scores

- So we have it, let's add this score to the saved pickle file that we created earlier

```
model_final = pickle.load(open(data_dir + "/model_final.sav","rb"))
```

```
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  0.6046              knn_5
1   accuracy  0.6268  knn_GridSearchCV
2   accuracy  0.6287             knn_29
3   accuracy  0.6268  knn_GridSearchCV
4   accuracy  0.6287             knn_29
5   accuracy  0.6268  knn_GridSearchCV
6   accuracy  0.6287             knn_29
7   accuracy  0.6268  knn_GridSearchCV
8   accuracy  0.6287             knn_29
9   accuracy  0.6268  knn_GridSearchCV
10  accuracy  0.6287             knn_29
11  accuracy  0.6268  knn_GridSearchCV
12  accuracy  0.6287             knn_29
13  accuracy  0.6268  knn_GridSearchCV
14  accuracy  0.6287             knn_29
15  accuracy  0.6268  knn_GridSearchCV
16  accuracy  0.6287             knn_29
17  accuracy  0.6268  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, which is `k = 29`
- Although it is an optimized model, it does not mean it will perform better than our previous model, it is just more accurate
- This is essential when finding a model champion

```
knn_best = grid.best_estimator_

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

```
0.6286610878661087
```

```
knn_champ = knn_best.score(X_test, y_test)
```

Model champion dataframe

- Let's add our final model to the dataframe and then pickle the dataframe
- This way, we can use the `model_final` dataframe across all our classification algorithms to choose our final model champion!

```
# Add this final model champion to our dataframe.
model_final = model_final.append({'metrics' : "accuracy" ,
                                  'values' : round(knn_champ, 4),
                                  'model':'knn_29' } ,
                                  ignore_index = True)
print(model_final)
```

```
     metrics  values              model
0   accuracy  0.6046              knn_5
1   accuracy  0.6268  knn_GridSearchCV
2   accuracy  0.6287             knn_29
3   accuracy  0.6268  knn_GridSearchCV
4   accuracy  0.6287             knn_29
5   accuracy  0.6268  knn_GridSearchCV
6   accuracy  0.6287             knn_29
7   accuracy  0.6268  knn_GridSearchCV
8   accuracy  0.6287             knn_29
9   accuracy  0.6268  knn_GridSearchCV
10  accuracy  0.6287             knn_29
11  accuracy  0.6268  knn_GridSearchCV
12  accuracy  0.6287             knn_29
13  accuracy  0.6268  knn_GridSearchCV
14  accuracy  0.6287             knn_29
15  accuracy  0.6268  knn_GridSearchCV
```

Module completion checklist

Objective	Complete
Perform cross-validation to understand better what the optimal model accuracy might be	✔
Find optimal hyperparameters with a grid search	✔
Use GridSearchCV to find optimal number of nearest neighbors and define optimal model	✔
Discuss reasons we would or would not use kNN	
Determine when to use logistic regression for classification and transformation of target variable	
Summarize the process and the math behind logistic regression	

kNN pros and cons

Pros

- Easy to use
- Can easily handle multiple categories
- There are many options to adjust (which features to use, measurement metric, etc)

Cons

- There are 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 1

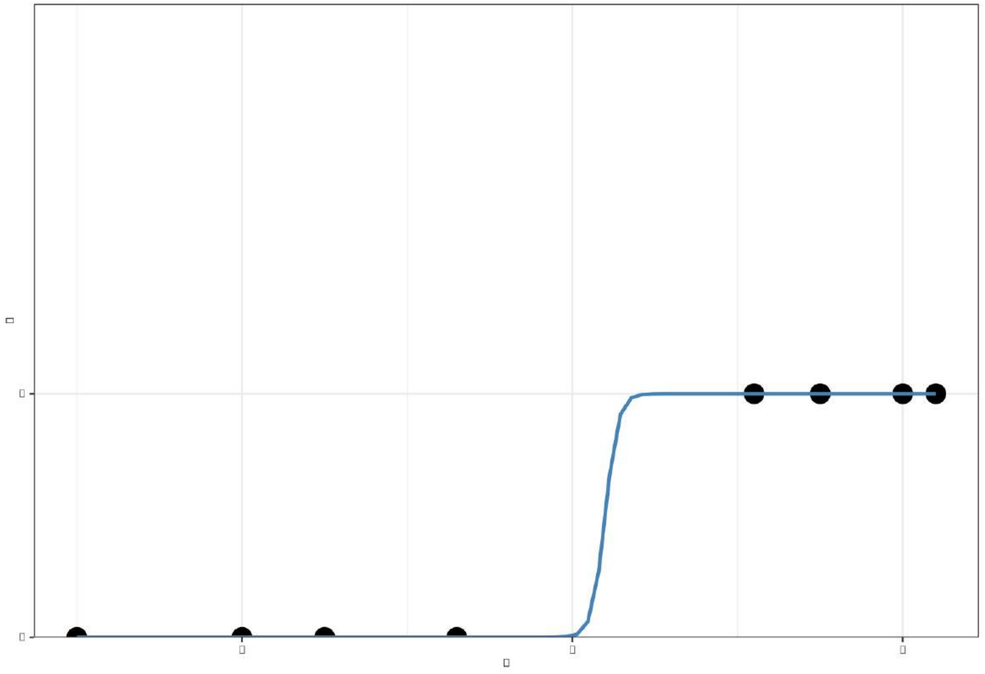
Exercise 2

Module completion checklist

Objective	Complete
Perform cross-validation to understand better what the optimal model accuracy might be	✔
Find optimal hyperparameters with a grid search	✔
Use GridSearchCV to find optimal number of nearest neighbors and define optimal model	✔
Discuss reasons we would or would not use kNN	✔
Determine when to use logistic regression for classification and transformation of target variable	
Summarize the process and the math behind logistic regression	

Logistic regression: what is it?

- **Supervised** machine learning method
- Target/dependent variable is **binary** (one/zero)
- Outputs the **probability** that an observation will be in the desired class (`y = 1`)
- Solves for coefficients to create a *curved* function to maximize the likelihood of correct classification
- `logistic` comes from the `logit` function (*a.k.a. sigmoid function*)

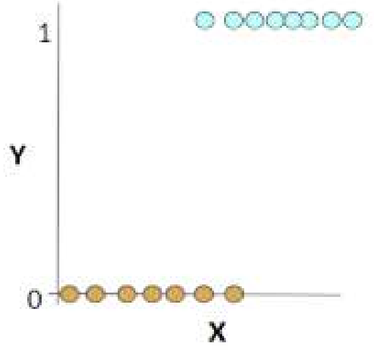


Logistic regression: when to use it?

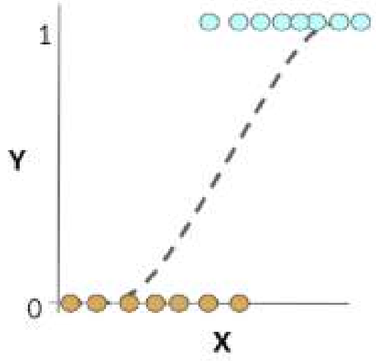
- Logistic regression is a **supervised learning algorithm**
- We use it to classify data into **categories**
- It outputs **probabilities** and not actual class labels
- Easily tweak its performance by adjusting a **cut-off probability**
- No need to re-run the model with new parameters
- It is a **well-established algorithm**
- It has many **implementations across various programming languages**
- We can create **robust**, **efficient,** and **well-optimized models**

Logistic regression: process

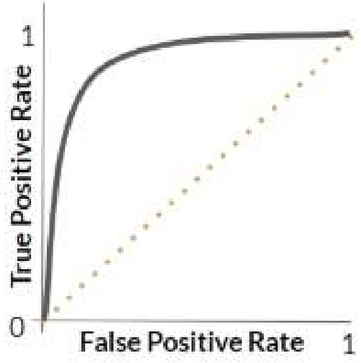
Step 1:
Convert target variable to 1/0



Step 2:
Logistic regression on training data



Step 3:
Use ROC curve & AUC to pick threshold

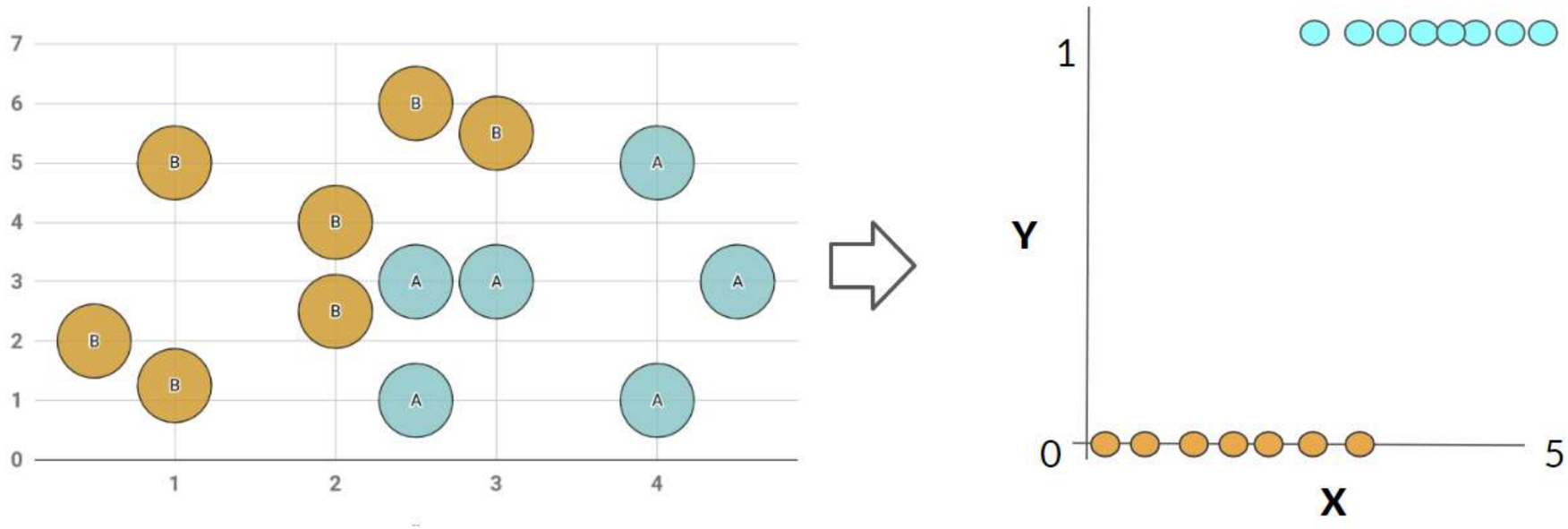


Step 4:
Check performance on test data

	Act +	Act -	
Pred +			
Pred -			

Categorical to binary target variable

Two main ways to prepare the target variable: - **First method:** translate an existing binary variable (i.e. any categorical variable with 2 classes) into `1` and `0`



Continuous to binary target variable

- **Second method:** convert a continuous numeric variable into binary one
  - We can do this by using a threshold and labeling observations that are higher than that threshold as `1` and `0` otherwise
  - If the median for the example below was `100`, then any point below the median is `0`, and any point above is `1`

Charge
193.89
0
39.99
201.65
117.9
200.88
79.99

Charge
1
0
0
1
1
1
0

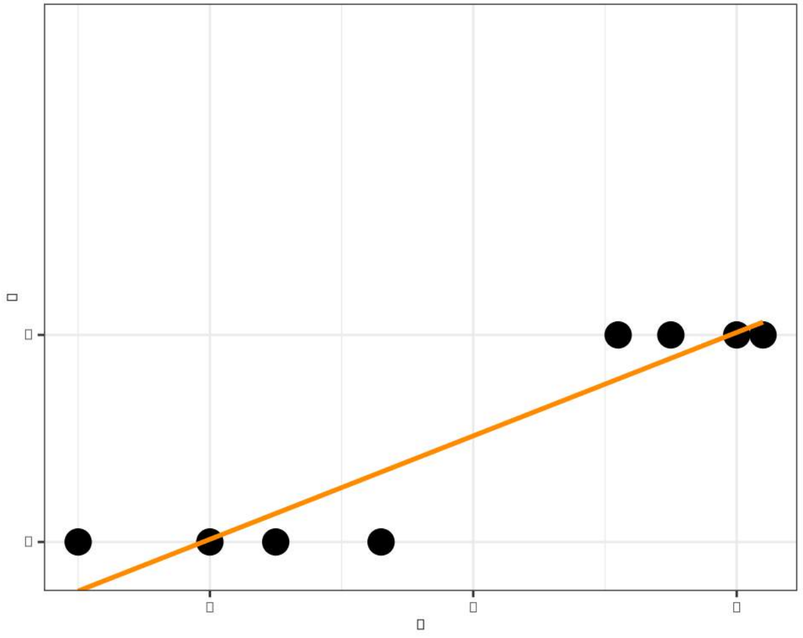
Module completion checklist

Objective	Complete
Perform cross-validation to understand better what the optimal model accuracy might be	✔
Find optimal hyperparameters with a grid search	✔
Use GridSearchCV to find optimal number of nearest neighbors and define optimal model	✔
Discuss reasons we would or would not use kNN	✔
Determine when to use logistic regression for classification and transformation of target variable	✔
Summarize the process and the math behind logistic regression	

Linear vs logistic regression

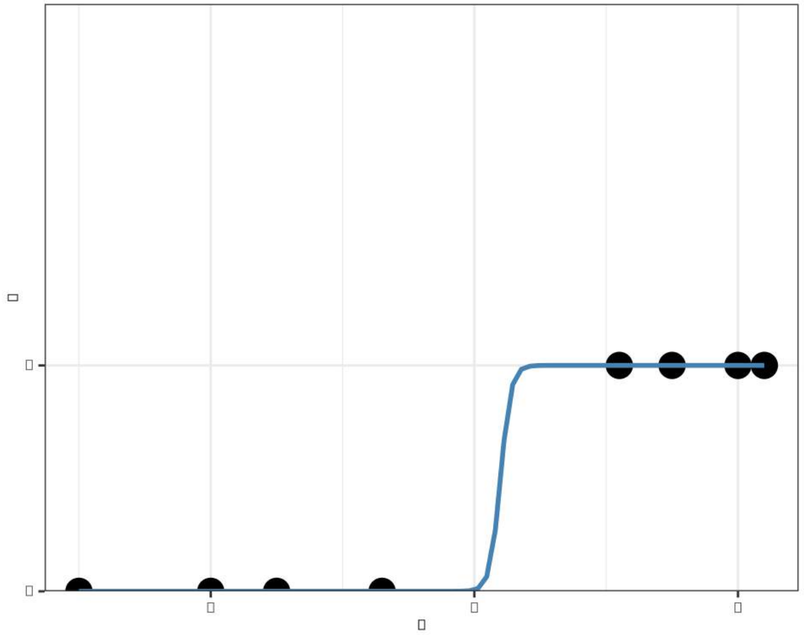
Linear regression line

- For data points $x_1, \ldots, x_n$, we have $y = 0$ or $y = 1$
- The function that "fits" the points is a simple line $\hat{y} = ax + b$



Logistic regression curve

- For the same data points $x_1, \ldots, x_n$, $y = 0$ or $y = 1$
- The function that "fits" the data points is a sigmoid $p(y = 1) = \frac{exp(ax+b)}{1+exp(ax+b)}$



Logistic regression: function

- For every value of x, we find p, i.e. probability of success, or probability that $y = 1$
- To solve for p, logistic regression uses an expression called a **sigmoid function:**

$$p = \frac{exp(ax + b)}{1 + exp(ax + b)}$$

- Although it may look a little scary (nobody likes exponents!), we can see a very **familiar equation inside of the parentheses:** $ax + b$

Logistic regression: a bit more math

Through some algebraic transformations that are beyond the scope of this course,

$$p = \frac{exp(ax + b)}{1 + exp(ax + b)}$$

can become

$$logit(p) = log\left(\frac{p}{1 - p}\right)$$

- Since `p` is the **probability of success**, `1 - p` is the **probability of failure**
- The ratio $\left(\frac{p}{1-p}\right)$ is called the **odds** ratio - it tells us the **odds** of having a successful outcome with respect to the opposite
- **Why should we care?**
- Knowing this provides useful insight into interpreting the coefficients

Logistic regression: coefficients

- In **linear** regression, the coefficients in the equation can easily be interpreted

$$ax + b$$

- An increase in x will result in an increase in y and vice versa

BUT

- In **logistic** regression, the simplest way to interpret a positive coefficient is with an increase in likelihood
- A larger value of x increases the likelihood that $y = 1$

Knowledge check 2

Module completion checklist

Objective	Complete
Perform cross-validation to understand better what the optimal model accuracy might be	✔
Find optimal hyperparameters with a grid search	✔
Use GridSearchCV to find optimal number of nearest neighbors and define optimal model	✔
Discuss reasons we would or would not use kNN	✔
Determine when to use logistic regression for classification and transformation of target variable	✔
Summarize the process and the math behind logistic regression	✔

Summary

Today we learned about

- applying cross-validation to understand what is the optimal model accuracy
- using hyperparameters and GridSearch
- logistic regression ad its applications

Tomorrow we will cover

- logistic regression on a training dataset and predict on test
- classification performance metrics
- transformation of categorical variables for implementation of logistic regression
- implementation of logistic regression on the data

Congratulations on completing this module!