Ik heb van meerdere van jullie vragen gekregen wat de test en validatie inhoudt in de knn classifier. We hebben er een beetje overheen geskipt in het college dus vandaar dit korte filmpje.

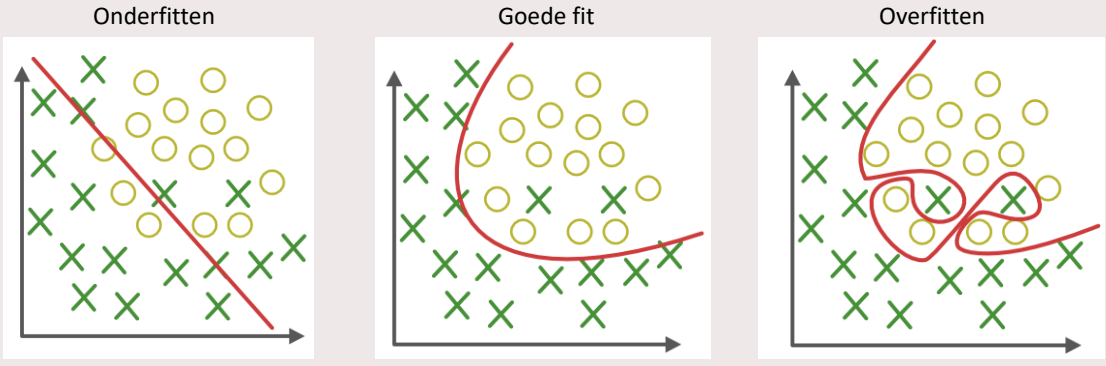
Om te begrijpen waarom we dit hebben, moet ik eerst het begrip “overfitten” uitleggen. Een plaatje zegt meer dan duizend woorden, dus here we go.

```python
def knn_classifier(X_train, y_train, X_validation, X_test, k):
    # Returns the labels for test_data, predicted by the k-NN clasifier trained on X_train and y_train
    # Input:
    # X_train - num_train x num_features matrix with features for the training data
    # y_train - num_train x 1 vector with labels for the training data
    # X_validation - num_test x num_features matrix with features for the validation data
    # X_test - num_test x num_features matrix with features for the test data
    # k - Number of neighbors to take into account
    # Output:
    # y_pred_validation - num_test x 1 predicted vector with labels for the validation data
    # y_pred_test - num_test x 1 predicted vector with labels for the test data

    X_test_val = np.vstack((X_validation, X_test))

    # Compute standardized euclidian distance of validation and test points to the other points
    D = cdist(X_test_val, X_train, metric='seuclidean')

    # Sort distances per row and return array of indices from low to high
    sort_ix = np.argsort(D, axis=1)
```

TU/e

Jullie hebben deze code gekregen, en ik kreeg de vraag “Wat is dat testen en validation?” Dat is waar we hier op in gaan.

Goede en slechte modellen



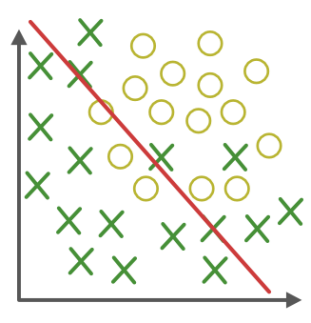
Links zien we een dataset, en een heel basic lineair classificatie model. We zien dat de scheiding tussen de categorieën niet echt top is: de data volgt meer een kromme lijn en een lineair model kan dat simpelweg niet omvatten. Dit heet “underfitten”: het model is te “simpel” voor de data, het past (of in het Engels fit) de dat niet goed. We hebben een iets geavanceerder model nodig wat de verhouding tussen de categorieën wel kan omvatten.

In het midden zien we het zoals het moet zijn. Wat opvalt is dat er een paar datapunten zijn die niet goed in het model passen. Als we kijken naar de accuracy op de training data, zal dat flink in de 90% zijn. Perfect gaan we het niet krijgen, maar that’s life.

En rechts zien we een model wat de training data 100% goed voorspelt. Het probleem is, het is wel héél erg specifiek op deze datapunten toegespitst. De kans dat dat representatief is voor wat in de echte wereld tegen gaan komen, is niet zo heel groot: het kringelt om alle datapunten heen zonder heel echt de trend te volgen van de data. Dat heet overfitten

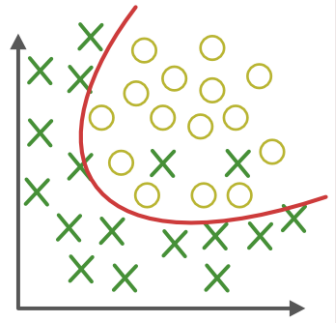
Meten met een testset

Onderfitten



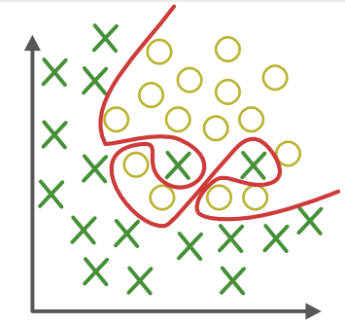
Training accuracy: 70%
Validatie dataset acc.: 70%

Goede fit



Training accuracy: 90%
Validatie dataset acc.: 90%

Overfitten



Training accuracy: 100%
Validatie dataset acc.: 40%

TU/e

Om te kijken of het model geoverfit is, kunnen we een nieuwe, extra dataset pakken die niet is gebruikt om het model te trainen. Dit heet de **validatie set**, omdat we daar ons model mee valideren. Links, bij het onderfitte model, is de accuracy op de training set niet heel hoog. Kijk je naar de nieuwe validatieset, waar dus niet op getraind is, dan is dat vergelijkbaar: ook niet super hoog maar vergelijkbaar met de training set.
Het middelste model lijkt voor ons optimaal: de training set heeft een hoge nauwkeurigheid, de validatie set ook. Maar kijken we naar het model rechts, dan zien we een flinke discrepantie. Het model is geoverfit en werkt alleen op de training set die het model al kent, niet op de nieuwe dataset waar niet op getraind is.

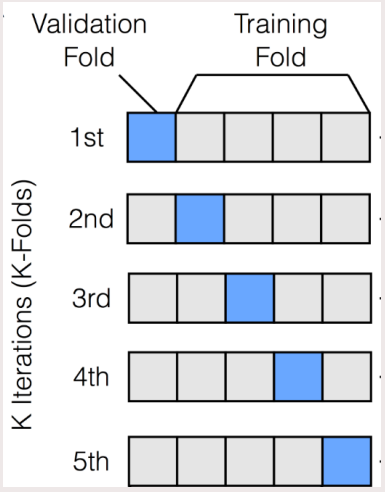
Hoe komen we aan zo'n extra set?

- Losse onafhankelijke dataset
- Cross-validatie
  - Verdeel dataset in blokjes`, gebruik 1 blokje om te testen en de rest om te trainen

Dit zijn de twee meest gebruikte manieren om aan een validatieset te komen. Het eerste wat je kan doen, is een nieuwe dataset maken die je *alleen* daarvoor gebruikt. Hoe dit bijvoorbeeld kan, is dat groep 1 de eigen set gebruikt om op te trainen, en bijv. de set van groep 2 als validatie set.
De tweede manier is met cross-validatie, dit staat ook in de voorbeeld code. Zie volgende slide:

Cross-validation

Verdeel de data in *k* delen, laat steeds een deel er uit als validatie set, en herhaal het experiment *k* keer.



TU/e

Dit is cross-validation: het plaatje rechts illustreert het. We verdelen de data in dit geval in vijf delen. Het eerste deel houden we apart, de andere vier delen gebruiken we om op te trainen. Op het eerste deel testen we het model vervolgens.
Dit herhalen we vijf keer, waarbij we elke keer een ander blokje eruit halen. Zo kunnen we over onze hele dataset testen hoe goed ons model is, zonder training en validatie data te mengen.

En wat over de test set?

Ons model heeft wat parameters die we kunnen tweaken: bij k-NN bijvoorbeeld het getal k.

Als we veel verschillende *k* proberen, is er een kans dat we een model vinden dat goed werkt op onze validatie set, maar weer niet op een nieuwe, ongeziene set.
Daarom wordt er ook vaak een extra set gebruikt om op het einde op te testen, dit is de test set. Deze wordt maar 1x gebruikt.

In de voorbeeldcode van ons, worden de validatie- en testset als één ding gebruikt. Voor dit project is het niet heel belangrijk dit goed te doen.

TU/e

Als je echt een test set wilt gebruiken, zou het het beste zijn een set van een ander groepje te pakken. Dit lijkt het beste op zoals we het in de echte wereld doen: we pakken data die we niet eerder gebruikt hebben, en testen hier 1x ons model op.