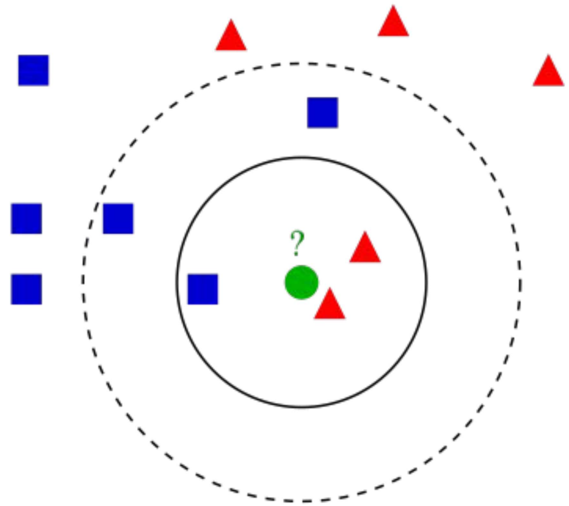
Lab 3:
kNN and Cross Validation

Machine Learning and Deep Learning
2023

Politecnico di Torino



Lab 3:
kNN and Cross Validation

Machine Learning and Deep Learning
2023

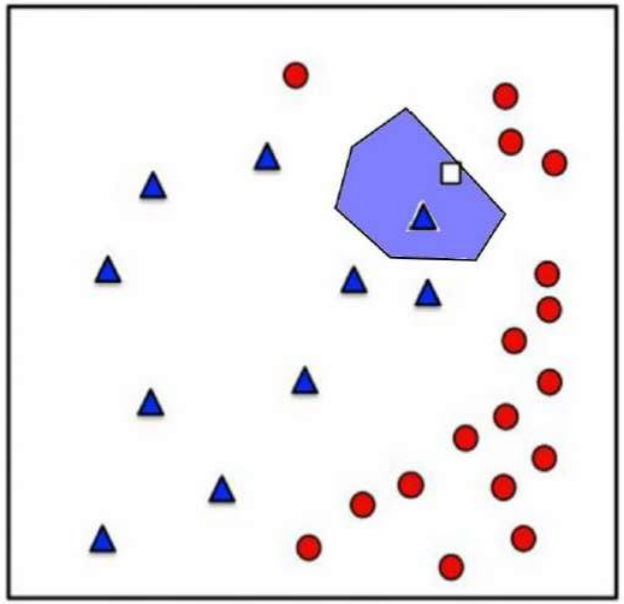
Politecnico di Torino

- Nearest Neighbor classifier
  - how does it work
  - PROs and CONs
- Model selection
  - Performance evaluation
  - Holdout evaluation
  - Cross Validation
  - k-fold Cross Validation

Nearest Neighbor classification

Intuition:

- linear classifiers have a hard time separating data in challenging distributions
- if we assume local smoothness of our data distribution, there is a simple yet clever technique to achieve non-linear boundaries:
  -> Nearest Neighbor classifier!



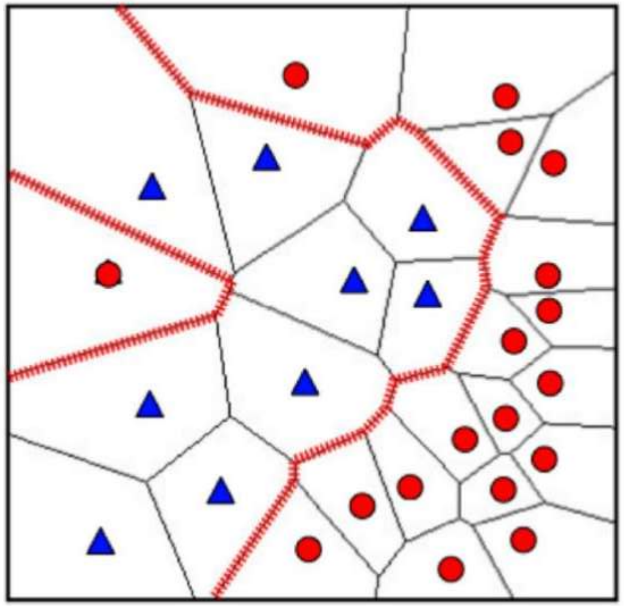
Nearest Neighbor classification

- Voronoi diagram: encloses each point within a region of space where that point is the closest neighbor. Represents the decision function

Problem: mislabelled point (or noise) can heavily affect this representation

-> Solution: use k- nearest neighbors rather than the closest neighbor

Voronoi diagram



Nearest Neighbor classification

PROs:

- few assumptions and the data
- **non-parametric** approach: no need to train a model
- easy to update when new samples are added to the dataset
- strong generalization guarantees

CONs:

- no trained model, but the cost is all the inference time
- requires to store all the features in memory
- does not scale well with data dimensionality

kNN in Sklearn

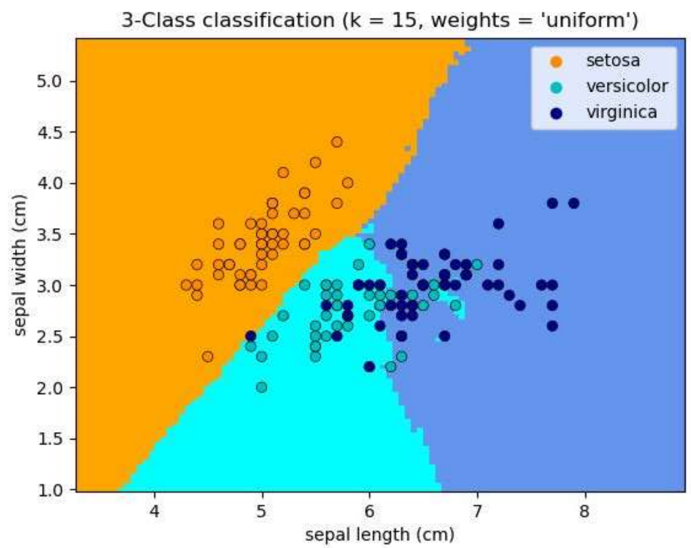
```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
n_neighbors = 3
random_state = 0
X, y = datasets.load_digits(return_X_y=True) # Load Digits dataset
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.5, stratify=y, random_state=random_state )# Split
into train/test

# Reduce dimension to 2 with PCA
pca = make_pipeline(StandardScaler(), PCA(n_components=2, random_state=random_state))
knn = KNeighborsClassifier(n_neighbors=n_neighbors)
pca.fit(X_train, y_train)  # Fit the method's model
 knn.fit(model.transform(X_train), y_train)
acc_knn = knn.score(model.transform(X_test), y_test)
```

Visualize neighbor space

```
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
from sklearn.inspection import DecisionBoundaryDisplay
n_neighbors = 15
# Create color maps
cmap_light = ListedColormap(["orange", "cyan", "cornflowerblue"])
cmap_bold = ["darkorange", "c", "darkblue"]

 _, ax = plt.subplots()
 DecisionBoundaryDisplay.from_estimator(
     clf, X, # pass here your classifier and data
    cmap=cmap_light, ax=ax,
     response_method="predict", plot_method="pcolormesh",
     shading="auto",
 )
```

Visualize neighbor space

```
# Plot also the training points
    sns.scatterplot(
        x=X[:, 0],
        y=X[:, 1],
        hue=iris.target_names[y],
        palette=cmap_bold,
        alpha=1.0,
        edgecolor="black",
    )
    plt.title(
        "3-Class classification (k = %i)" % (n_neighbors)
    )
plt.show()
```



Exercise 1: kNN on MNIST

Reproduce the previous example in Colab:

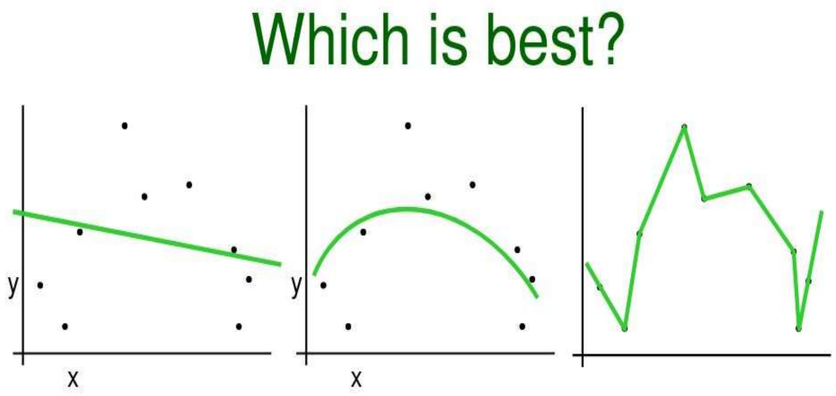
- train a kNN on MNIST
- what happens if we don't standardize?
- apply PCA and try again. Visualize the classified points, changing the number of n_neighbors

Model evaluation

How to evaluate the quality of an algorithm after training?

- Key question is : ‘How well are you going to predict future data drawn from the same distribution?’
- We do not have access to data from future
- We cannot evaluate on the train set, or we wouldn’t get a fair estimate

Example with linear regression



Model evaluation

We need to assess the quality of prediction of our model

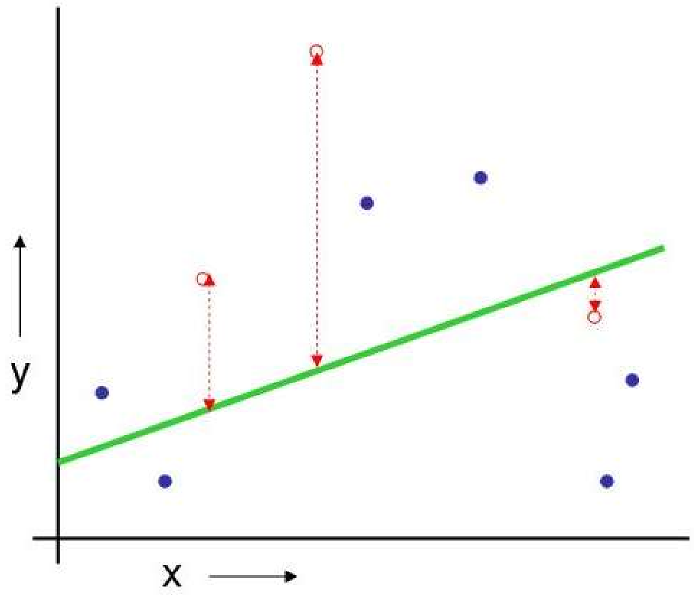
- We are interested in testing on **unseen** data

-> We simulate the performance on unseen data using known data

Holdout method

Simple strategy:

- Choose randomly a 30 % to use as a test set (possibly stratified)
- Use the rest for training
- Finally, use the test set to estimate future performances



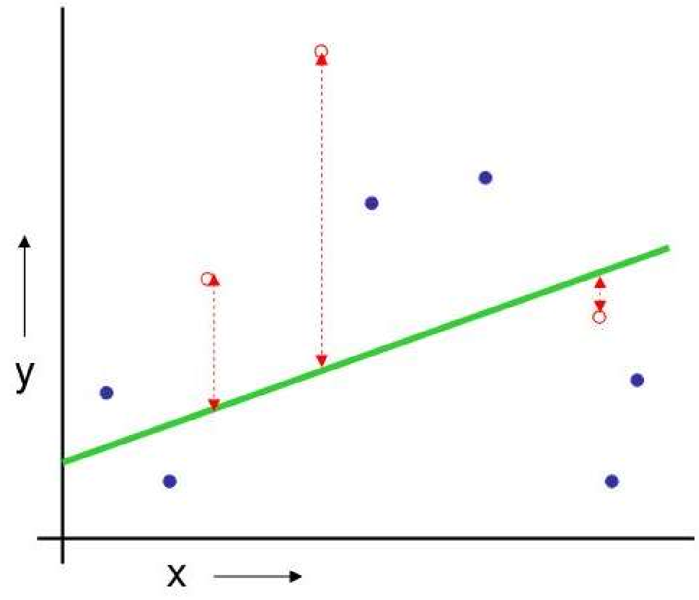
Holdout method

PROs;

- simply choose the best model

CONs;

- can be wasteful: ‘sacrifice’ 30% of the data
- especially if the dataset is small, the test set might end up being ‘lucky’ or unlucky’
    - more precisely: “the test set estimator might have high variance”



Model selection

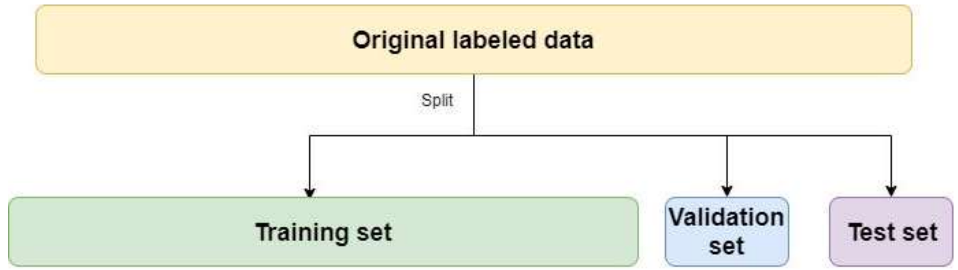
Besides the listed weaknesses…

Another big problem with test set evaluation… we need to do model selection!

- The test set should not be used to make any decision on the system!
- Otherwise we obtain a model biased on the test set, and the estimate that we get loses significance

Model selection

- We need to test out different hyperparameters and choose the best without biasing our evaluation
- We can make use of an additional part of the train set: a **validation set**
- The **validation set** used to assess influence of hyperparameters; should **NOT** be used as a final estimate

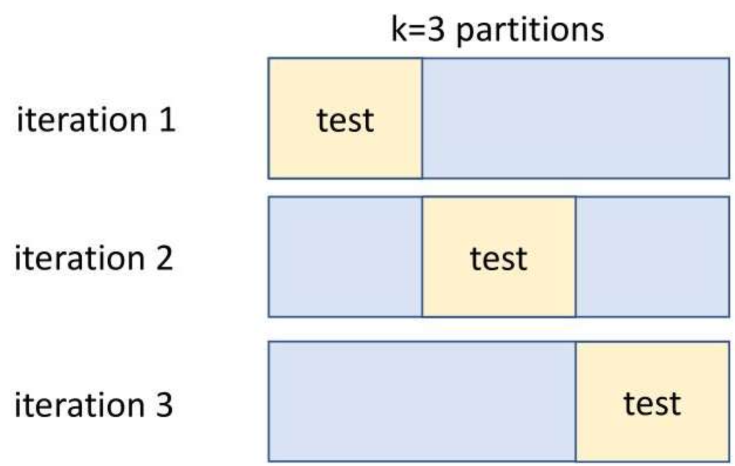


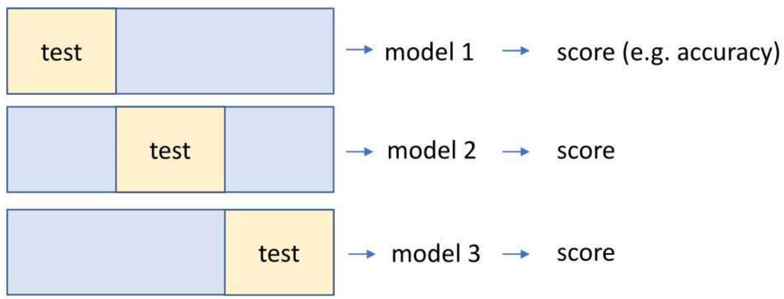

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_digits
X, y = load_digits(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)

X_train, X_val, y_train, y_val = train_test_split(
...     X_train, y_train, test_size=0.33, random_state=42)
```

Cross Validation

K- fold cross-validation:

- Divide the data into k sets, or **folds**
- Repeat training K-1 times, holding out a different fold each time
- Aggregate the results over the K folds





Cross Validation

In SkLearn:

- Option 1 : create the splits and iterate on them
- Option 2: use the provided cross_val_score, that will automatically return the scores on the K folds

Option 1

```
from sklearn.model_selection import KFold
# K-Fold with 5 splits
kfold = KFold(n_splits=5, shuffle=True)
for train_indices, test_indices in kfold.split(X, y):
... executed 5 times, 1 for each k-fold iteration ...
```

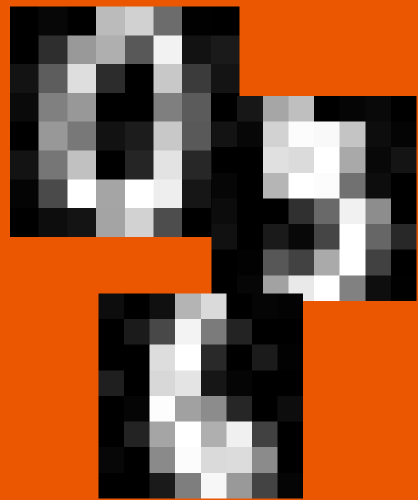
Option 2

```
from sklearn.model_selection import cross_val_score
clf = DecisionTreeClassifier()
f1 = cross_val_score(clf, X, y, cv=5,
scoring='accuracy')
```

Exercise 2: tune kNN hyperparameters

Starting from the previous example in Colab:

- Define a train-test split and further divide the train set into k=5 folds
- Define the set of hyperparameters to tune (weights, n_neighbors, p)
- Run a grid search evaluating the best model using k-fold Cross Validation
- Add PCA dimensionality to the set of hyperparameters



Exercise 3: Cross Validation

- Repeat exercise 2 without using a k-fold split, pick the best model using the test set
- Is the final score higher or lower than before?
- What if we sample a new test set?
- is step 1 of this exercise a good practice or not ?