Machine Learning for Pattern Recognition

May require background reading for machine learning.

In may computer vision applications it will take the from of inputting an image to a feature extractor where it does some image processing. This produces **featurevectors** from the image which is then taken into the machine learning part that uses these feacturevectors to make intelligent decisions.

Feacturevectors are a mathematical vector with a fixed number of elements (**dimensionality**) and each value represents a point or direciton in a **featurespace**. Vectors of differing dimensionality can't exist in the same featurespace.

These feature extractors are intended to produce vectors for similar images close to one another. The closeness can then be measure between the vectors.

We can measure in:

- Euclidean Distance (L2 distance)
  - Most intuitive, straight line between two points
  - Computed via extension of Pythagoras theorem to n dimensions
  - $D_2(p,q)=\sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}=\|p-q\|=\sqrt{(p-q)\cdot(p-q)}$
- Manhattan Distance (L1 distance)
  - Computed along paths parallel to the axes of the space
  - $D_1(p,q)=\sum_{i=1}^{n}|p_i-q_i|=\|p-q\|_1$
- Cosine Similarity
  - Measures the cosine of angle between two vectors (so at 1 they are in line)
  - It is **not a distance**!
  - This is useful if you don't care about the relative length of vectors
  - $\cos(\theta)=\frac{p\cdot q}{\|p\|\|q\|}=\frac{\sum_{i=1}^{n}p_i q_i}{\sqrt{\sum_{i=1}^{n}p_i^2}\sqrt{\sum_{i=1}^{n}q_i^2}}$

You should choose the features which allow you to distinguish objects or classes of interest. Things that are similar within classes or different between classes.
It's best to keep the number of features small as machine learning gets much more difficult as dimensionality of featurespace gets large.

Classification

Classification is the process of assigning a class label to an object (typically represented by a vector in a feature space).

A **supervised machine learning algorithm** uses a set of pre-labelled training data to learn how to assign calss labels to vectors (and the corresponding objects).

Spoiler Alert! The image in the slides is a dog (apparently trimmed to look like a cat but i'm not convinced).

A **binary** classifier only has two classes whereas a **multiclass** classifier has many classes.

Liner classifiers try to learn a **hyperplane** that sepetates two classes in featurespace with minimum error. Different linear classification algorithms apply differing constraints when learning the classifier.

Therefore to classify a new image you just see which side of the hyperplane it's on.

Linear classifiers work best when the data is linearly separable but there are some cases where you need a **non-liner** binary classifier. One such classifier is called **Kernel Support Vector Machines** which learns non-liner boundaries. It is best to keep them rather general and have mistakes than to overfit to the training data.

Multiclass classifiers:

- K Nearest Neighbours
    - We tend to use an odd K so that you can't have a situation with equal votes between classes.
    - It becomes computationally expensive if there lots of traning examples or many dimensions
    - KNN is rarely chosed due to its lack of scalability
- Linear Classifier
    - By definition these are binary!!
    - Can use for multicalss with:
        - One versus All (OvA)/ One versus Rest (OvR) = one classifier per class
        - One versus One (OvO) K(K-1)/2 classifiers
- Clustering
    - Group data without prior knowledge
    - Items of similar vectors are grouped by clustering operation
    - Some create overlapping groups but we're interested in ones that don't
- K-Means Clustering
    - Group data into K groups
    - Each group represented by a centroid (centre point)
    - You choose K, then K initial cluster centres (centroids) are chosen
    - Then iteratively:
        - Each point assigned to closest centroid
        - Centroid recomputed as mean of all assigned points
        - If no assigned points move to random new point

- Final clusters are created by assigning all points to nearest centroid
- This is proven to always converge but not necessarly optimally
    - For instance in situations where there cannot be grouped linearly