Unsupervised Learning Techniques

Unsupervised Learning

- We can have a dataset with several thousands of instances. However, collecting labels can be long, costly, and tedious.
- Unsupervised learning works on unlabeled data.
- Some algorithms are clustering, anomaly detection, and density estimation.

Clustering

- Clustering algorithms identify similar instances and assign them to clusters.
- Some applications of clustering include:
    - Customer segmentation: clustering customers based on their purchases and activity on a website.
    - Data analysis: simplifies analysis by forming clusters.
    - Dimensionality reduction: we can measure each instance's affinity with each cluster and create a k-dimensional vector based on k-clusters.
    - Feature engineering: using k-means to add cluster affinity features to a dataset.
    - Anomaly detection: identifying outliers and unusual behavior
    - Semi-supervised learning: performs clustering to propagate labels to all instances in the same cluster.
    - Search engines: returning images similar to the reference image.
    - Image segmentation: clustering pixels according to their color.

K-means

- Based on the Lloyd-Forgy algorithm, k-means clusters datasets into blobs.
- We can use Scikit-Learn's **KMeans()** class, which creates *n-clusters* number of blobs.
- An instance's label is the cluster index to which the algorithm assigns this instance.
    - Note that this label is different from the class labels in classification.
    - We can use KMeans's **cluster_centers_** variable to find the data points of the centroids, the centers of each cluster.
    - Going forward in this chapter, the classes or functions mentioned will originate from the Scikit-Learn API unless specified otherwise.
- The k-means algorithm does not behave well when blobs have different diameters.
- A hard-clustering algorithm assigns each instance to a single cluster, whereas a soft-clustering algorithm gives each instance a score per cluster.
    - We can use KMeans's **transform()** method, which measures the distance from each instance to every method.

- The k-means algorithm begins by randomly placing centroids, labeling the instances, updating the centroids, and so on until the centroids converge.
    - Although the algorithm guarantees that the centroids converge, they may converge to a local optimum, which may not be the best solution.

Centroid Initialization Methods

- If we know the centroids' locations a priori, we can set KMeans's *init* parameter to an array containing the coordinates of the centroids.
    - In this case, we should also set the *n_init* parameter to 1.
- Alternatively, we can run the algorithm multiple times with different random initializations and keep the best solution.
    - We will use the model's inertia (the sum of squared distances between the instances and their closest centroids) as a performance metric.
    - We can access this metric using KMeans's **inertia** variable or the **score** method.

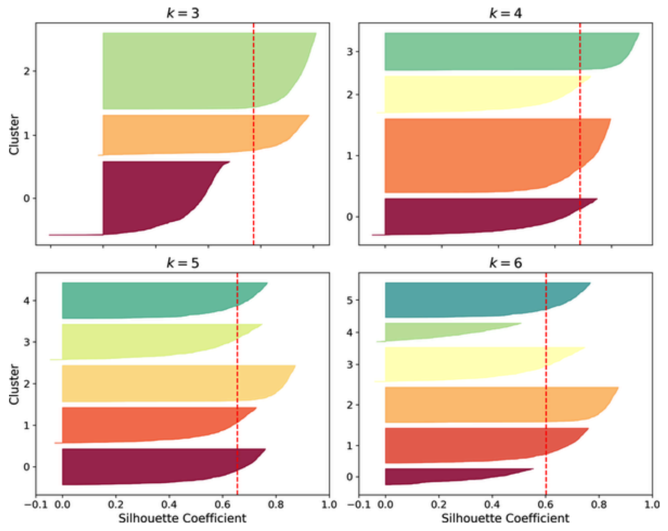
Accelerated k-means and mini-batch k-means

- The accelerated k-means algorithm exploits the triangle inequality and keeps track of the lower and upper bounds for distances between instances and centroids.
    - Depending on the dataset, this algorithm may be faster or slower than normal k-means.
    - To use this algorithm, we set KMeans's *algorithm* parameter to "elkan."
- The mini-batch k-means algorithm forms mini-batches of the dataset and moves the centroids only slightly at each iteration.
    - This algorithm is faster than the regular k-means algorithm, typically by three or four times. However, the inertia is usually slightly worse.
    - The **MiniBatchKMeans()** class performs this algorithm.
    - We can use the **memmap()** class for better memory management or the **partial_fit()** method to pass the algorithm one mini-batch at a time.

Finding the optimal number of clusters

- We cannot use inertia as a metric to decide the optimal number of clusters.
    - The more clusters there are, the closer each instance will be to its closest centroid, driving inertia down.
    - So, if we used inertia, the models with the most clusters would be chosen.
- A more precise approach would be to use the silhouette score.
    - The silhouette score is the mean silhouette coefficient over all instances.
    - An instance's silhouette coefficient is equal to $(b - a)/max(a, b)$ where:
        - a is the mean distance to the other instances in the same cluster,
        - and b is the mean distance to the instances of the next closest cluster.
    - The coefficient varies between -1 and 1.
        - If it is close to 1, the instance is well inside its cluster.
        - If it is close to 0, the instance is close to a cluster boundary.
        - If it is close to -1, the instance may be in the wrong cluster.
    - We can use the **silhouette_score()** function, providing it with all the instances in the dataset and their assigned labels (e.g., **KMeans.labels_**).

- We can create a silhouette diagram by plotting every instance's silhouette coefficient, sorted by the clusters they were assigned to and by the value of the coefficient.
    - The diagram contains one knife per cluster. The height indicates the number of instances in the cluster, and the width represents the silhouette coefficient (the wider, the better).
    - We want the clusters to have similar sizes.



Limits of k-means

- It may be necessary to run the algorithm several times to avoid suboptimal solutions.
- We also need to specify the number of clusters, which may be difficult to compute.
- The algorithm performs poorly with clusters with varying sizes, densities, or nonspherical shapes.
- We can scale the features to help the algorithm perform better.

Clustering for Image Segmentation

- In image segmentation, we partition an image into multiple segments. There are different types of segmentation.
    - Color segmentation: pixels with a similar color are assigned to the same segment.
    - Semantic segmentation: all pixels of the same object type are assigned to the same segment.
    - Instance segmentation: all pixels of the same individual object are assigned to the same segment.
- An image can be represented as a 3D array: height as the first dimension, width as the second, and the RGB values as the third.
    - The **PIL** package has powerful image manipulation capabilities. We can use NumPy's **asarray()** function to represent the image as a 3D array.
    - We will use KMeans() to create segments and PIL's **reshape()** function to apply these segments.

Clustering for Semi-Supervised Learning

- We can use any estimator or model to create labeled instances.
- Then, we can use KMeans() to create clusters to get the centroids and label them.
- Finally, we can propagate these labels to all other instances in the same cluster.
    - The **LabelSpreading()** and **LabelPropagation()** classes can propagate labels for us.
- We can ignore the instances farthest away from their cluster center to increase the model's accuracy.
- To continue improving our model, we can do active learning where a human expert interacts with the learning algorithm.
    - The expert would provide the labels for specific instances when the algorithm requests them. The basic process is:
        - First, the model is trained on labeled instances and makes predictions on all unlabeled ones.
        - Then, the instances the model is most uncertain about are given to the expert for labeling.
        - We continue this process until the performance improvements plateau.

DBSCAN

- The density-based clustering of applications with noise (DBSCAN) algorithm defines clusters as high-density regions.
    - For each instance, the algorithm counts how many instances are located within a small radius (ε) from it. This region is called the instance's neighborhood.
    - If an instance has at least *min_samples* instances in its neighborhood (including itself), it is considered a core instance located in dense regions.
    - All instances in the neighborhood of a core instance belong to the same cluster.
        - A neighborhood may include other core instances, so a long sequence of neighboring core instances forms a cluster.
    - Any instance that is not a core instance and does not have one in its neighborhood is considered an anomaly or outlier.
- We can use the **DBSCAN()** class, which has an *eps* parameter that controls the radius of each cluster.
    - If a label has a cluster index of -1, it is considered an anomaly.
    - We can access the core instances with the **components_** variable and their indices with the **core_sample_indices_** variable.
    - The DBSCAN() class does not have a predict() method, so we must use another classification algorithm, like K-nearest neighbors, to make predictions.
        - To use the k-nearest neighbors algorithm, we can use the **KNeighborsClassifier()** class.
- Other clustering algorithms include agglomerative, BIRCH, mean-shift, affinity propagation, and spectral clustering.

Gaussian Mixtures

- A Gaussian mixture model (GMM) is a probabilistic model that assumes that the instances were generated from a mixture of several Gaussian distributions.
    - All instances generated from a Gaussian distribution form a cluster that typically looks like an ellipsoid.
    - Each cluster can vary in ellipsoidal shape, size, density, and orientation.
- We can use a GMM through the **GaussianMixture()** class.
    - We must have the number *k* of Gaussian distributions in advance. The *n_components* parameter controls the number of Gaussian distributions.
    - This class gives us access to the model's weight, mean, and covariance values.
- The GMM algorithm is as follows:
    - For each instance, a cluster is picked randomly among k clusters.
    - The probability of choosing the jth cluster is the cluster's weight ϕ(j).
    - The index of the cluster chosen for the ith instance is z(i).
    - If z(i) = j, then the location x(i) of this instance is sampled randomly from the Gaussian distribution with mean μ(i) and covariance Σ(j).
        - This is noted x(i) ~ N(μ(j), Σ(j)).
    - Lastly, generate dataset X.
- More, generally:
    - The algorithm initializes the cluster parameters randomly.
    - Then, it assigns instances to clusters (the expectation step).
    - After that, it updates the clusters (the maximization step).
    - This algorithm is similar to k-means clustering except that Gaussian distribution finds the cluster centers, size, shape, orientation, and relative weights.
        - Also, like k-means, Gaussian distribution may need to be run several times to get an optimal solution.
- We can check if the algorithm converged using the **converged_** variable and how many iterations it took with the **n_iter** variable.
- GaussianMixture() uses predict() for hard clustering and predict_proba() for soft clustering.
- GMM is a generative model, so we can sample new instances using the **sample()** method.
- Lastly, we can access the model's density at any given location using the **score_samples()** method.
    - The greater the value of the score, the higher the density.

Gaussian Mixtures for Anomaly Detection

- Any instance located in a low-density region can be considered an anomaly.
- We must define a density threshold for the model to identify anomalies.
    - If the model gives us too many false positives, we can lower the threshold.
    - If it gives us too many false negatives, we can increase the threshold.
    - This reflects the precision/recall trade-off.
- We can use the GaussianMixture's score_samples() method to get the densities and NumPy's **percentile()** method to get the threshold.

- Novelty detection is related to anomaly detection, but the novelty detection algorithm assumes that the data is clean and uncontaminated by outliers.

Selecting the Right Number of Clusters

- We can try to find the model that minimizes the theoretical information criterion, such as the Bayesian information criterion (BIC) or the Akaike information criterion (AIC).
    - $BIC = log(m)p - 2log(\mathcal{L})$
    - $AIC = 2p - 2log(\mathcal{L})$
    - Where m is the number of instances, p is the number of parameters learned by the model, and $\mathcal{L}$ is the maximized value of the model's likelihood function.
- The BIC and AIC penalize models with more clusters and reward models that fit the data well.
    - They often end up choosing the same model. If not, the model selected by the BIC tends to be simpler but doesn't fit the data as well as the model selected by the AIC.
- Probability vs Likelihood: Probability describes how plausible a future outcome is (knowing the parameter values). In contrast, likelihood describes how plausible a set of parameter values is after the outcome is known.
- We can compute the BIC and AIC with GaussianMixture's **bic()** and **aic()** methods.

Bayesian Gaussian Mixture Models

- We can use the **BayesianGaussianMixture()** class to find the optimal number of clusters.
    - The class returns an array of weight values. Values that are close to zero are unnecessary clusters.
- Although GMMs work great on clusters with ellipsoidal shapes, they don't perform well on clusters with very different shapes.

Other Algorithms for Anomaly and Novelty Detection

- All these algorithms are available in Scikit-Learn's API.
    - Fast-MCD (minimum covariance determinant): useful for outlier detection, in particular, to clean up a dataset.
    - Isolation Forest: builds a random forest where each decision tree is grown randomly.
        - Isolation forest randomly picks a feature and threshold to split the dataset in two.
        - This process continues down the tree until all instances are isolated from other instances.
        - Anomalies tend to get isolated in fewer steps than normal instances.
    - Local outlier factor (LOF): compares the density of instances around a given instance to the density around its neighbors.
        - An anomaly is often more isolated than its k-nearest neighbors.
        - This algorithm is also useful for detecting local and global anomalies.
    - One-class SVM: better suited for novelty detection.

        - One-class SVM tries to separate instances in high-dimensional space from the origin.
        - In the original space, this will correspond to finding a small region encompassing all the instances.
        - If a new instance does not fall within this region, it is an anomaly.
    - PCA and other dimensionality reduction techniques with an inverse_transform() method.
        - If we compare a normal instance's reconstruction error with an anomaly's, the latter will usually be much larger.