COMP6200 - Cheat Sheet
4 June, 2024

Multiple Choices

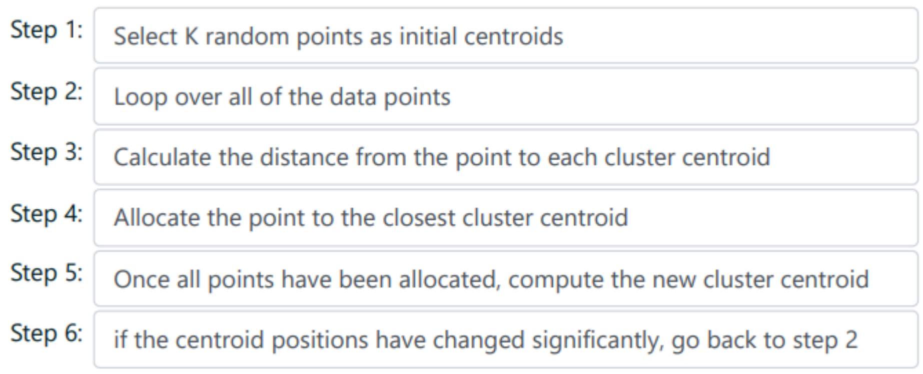
1. Unsupervised (instances are not labeled): Clustering (K-means and Hierarchical)
2. Supervised (instances are labeled): Regression and Classification

Evaluation Metrics for Supervised Learning

- MSE (Mean Squared Error, smaller-better), R-square (closer21-better)
- Accuracy: a classification metric that measures the proportion of correct predictions out of the total number of predictions = (True Positives + True Negatives) / (Total Number of Instances))
- Precision: a classification metric that measures the proportion of true positive predictions out of all positive predictions made by the model = True Positives / (True Positives + False Positives)
- Recall: a classification metric that measures the proportion of true positive predictions out of the actual positive instances = True Positives / (True Positives + False Negatives)
- F1-score: the harmonic mean of precision and recall, providing a single metric that balances both measures = 2 * (Precision * Recall) / (Precision + Recall).

Unsupervised Models

1. **K-Means Clustering** (compute centroid, clustering points to their closest centroids, update centroids)
    a. Preprocess: Data normalization
    b. Overfitting probably occurs if K is a large number.
    c. Steps



Step 1: Select K random points as initial centroids
Step 2: Loop over all of the data points
Step 3: Calculate the distance from the point to each cluster centroid
Step 4: Allocate the point to the closest cluster centroid
Step 5: Once all points have been allocated, compute the new cluster centroid
Step 6: if the centroid positions have changed significantly, go back to step 2

K-Means Clustering
Advantages

- Relatively simple and easy to understand
- Computationally efficient for large datasets
- Converges quickly to a local optimum

Disadvantages

- Requires specifying the number of clusters (k) in advance
- Sensitive to outliers and initial centroid positions
- Assumes clusters are spherical and of similar sizes
- May converge to different solutions with different initializations

2. **Hierarchical Clustering** (bottom up learning process by merging two closest points or clusters)
   The hierarchical clustering algorithm is a bottom-up approach that starts by treating each data point as its own cluster. It then iteratively merges the two closest clusters based on a specific distance metric until all data points are merged into a single cluster. This process is known as agglomerative clustering.
   a. How to compute the distance between two clusters
      - **Single Linkage** (Nearest Neighbor): The distance between two clusters is the minimum distance between any pair of points, one from each cluster.
      - **Complete Linkage** (Furthest Neighbor): The distance between two clusters is the maximum distance between any pair of points, one from each cluster.
      - **Average Linkage**: The distance between two clusters is the average distance between all pairs of points, where one point is from each cluster.
      - **Ward's Method**: This method minimizes the sum of squared differences within all clusters. It aims to minimize the increase in the total within-cluster variance after merging.

   b. How to choose the number of clusters based on Dendrogram
      A dendrogram is a tree-like diagram that visualizes the hierarchical clustering process. It shows the merging of clusters at different levels of similarity (or distance). To choose the number of clusters from the dendrogram, one can:
      - Visually inspect the dendrogram and look for natural breaks or long vertical lines that separate distinct clusters.
      - Use a heuristic method like the elbow method, which looks for the point where the marginal gain in explanation (or reduction in variance) drops significantly.
      - Apply a statistical approach like the gap statistic or silhouette analysis to determine the optimal number of clusters.

Hierarchical Clustering
Advantages

- Does not require specifying the number of clusters in advance
- Can handle non-spherical and different-sized clusters
- Provides a visual representation (dendrogram) of the clustering process
- Can be applied to both distance and similarity measures

Disadvantages

- Computationally expensive for large datasets (quadratic time complexity)
- Can produce different results based on the linkage method used
- Prone to noise and outliers (especially with single linkage)
- Difficult to interpret the optimal number of clusters from the dendrogram

Suitability
In general, K-Means is preferred for larger datasets and when the number of clusters is known or can be reasonably estimated. Hierarchical clustering is more suitable when the cluster structure is unknown and when the dataset is relatively small.

Supervised Models

1. **K Nearest Neighbour (KNN)**
Distinguish regression (continuous output) and classification (categorical output).
A learning task has: 1) Dataset; 2) Learning Algorithm; 3) Performance Measure

Learning Process

- Learning with the training dataset and testing with the testing dataset. For KNN, how to define similarity? Euclidean distance
- K is the number of nearest neighbors to look at before classifying a new instance

Overfitting probably occurs if K is small

How to split the dataset for training and testing?
A common practice is to use a random sampling approach to split the dataset into two subsets: a training set and a test set. The training set is used to train the model, while the test set is used to evaluate its performance on unseen data. A typical split ratio is 80% for training and 20% for testing, but other ratios like 70/30 or 75/25 are also common.

Cross Validation - Curse of Dimensionality in KNN
Cross-validation is a technique used to assess the performance of a model and to tune its hyperparameters. It involves partitioning the dataset into multiple subsets, training the model on a subset (training set), and evaluating its performance on the remaining subset (validation set). This process is repeated for different partitions, and the results are averaged to obtain a more reliable estimate of the model's performance.

The curse of dimensionality can significantly impact the performance of the k-Nearest Neighbors (KNN) algorithm in high-dimensional spaces. As the number of dimensions (features) increases, the following issues arise:

- **Distance Concentration:** Distances between data points tend to become concentrated around their mean value, making it difficult to distinguish between different classes or patterns.
- **Computational Complexity:** The computational cost of calculating distances between the query point and all training points grows exponentially with the number of dimensions.
- **Irrelevant Features:** Irrelevant or redundant features can introduce noise and distort the distance calculations, leading to incorrect neighbor assignments.
- **Curse of Dimensionality in Nearest Neighbor Search:** Finding the nearest neighbors in high-dimensional spaces is a challenging problem in itself.

Advantages and Disadvantages of KNN

Advantages of KNN:

Simple and intuitive algorithm
Non-parametric method, does not make assumptions about the underlying data distribution
Effective for multi-class classification and regression tasks
Can adapt well to the local structure of the data
Efficient for low-dimensional data

Disadvantages of KNN:

Sensitive to the curse of dimensionality (as discussed earlier)
Computationally expensive for large datasets, especially during prediction
Requires storing the entire training dataset
Sensitive to the choice of the distance metric and the value of k
Easily influenced by irrelevant or redundant features
Does not provide confidence scores or probability estimates for predictions

While KNN is a simple and effective algorithm for low-dimensional data, its performance can degrade in high-dimensional spaces due to the curse of dimensionality. In such cases, dimensionality reduction techniques, feature selection, or alternative algorithms may be more appropriate.

2. Naïve Bayes Model

Bayes Rules

$$P(C|\mathbf{X}) = \frac{P(\mathbf{X}|C)P(C)}{P(\mathbf{X})}$$

Perceive features as random variables P(X|C) where C is the class and X is the vector of features. Given features X, find class C such that:

$$\text{Label}(x) \leftarrow \arg\max_{C_k}\{p(C_k|\boldsymbol{x})\}$$

Important Assumption: All features are independent random variables
Naïve Bayes model estimates P(Xi|C) separately from the training dataset, where Xi is the feature and C is the class

Objective is to maximize a posterior:

$$[P(x_1|c^*) \cdots P(x_n|c^*)]P(c^*) > [P(x_1|c) \cdots P(x_n|c)]P(c), \qquad c \neq c^*, c = c_1, \cdots, c_L$$

How to deal with continuous random variable
Naïve Bayes classifiers are well-suited for handling discrete or categorical data, but they can also handle continuous random variables with some adjustments. Here are a few common approaches to deal with continuous random variables in Naïve Bayes models:

> Discretization: One approach is to discretize the continuous variables by binning them into discrete intervals or categories. This can be done using techniques like equal-width binning, equal-frequency binning, or more advanced methods like entropy-based discretization.

> Gaussian Naïve Bayes: If the continuous variables follow a Gaussian (normal) distribution, you can use the Gaussian Naïve Bayes model. This model assumes that the continuous features follow a Gaussian distribution and estimates the mean and variance of each feature for each class.

> Kernel Density Estimation: Instead of assuming a specific distribution (like Gaussian), you can estimate the probability density function (PDF) of each continuous feature using non-parametric methods like kernel density estimation (KDE). The estimated PDFs are then used to calculate the likelihood of observing a particular feature value given the class.

> Semi-Naïve Bayes: For mixed datasets containing both discrete and continuous variables, you can use a semi-Naïve Bayes approach. This involves handling the discrete variables using the standard Naïve Bayes method and treating the continuous variables separately, either by discretization or using a Gaussian or kernel density estimation approach.

Advantages and Disadvantages of Naïve Bayes model

Advantages

- Simple and easy to understand and implement.
- Computationally efficient, especially for large datasets.
- Requires a relatively small amount of training data.
- Handles missing data well by ignoring the missing feature(s) during calculation.
- Can be used for both classification and regression tasks.

Disadvantages

- Assumes independence between features, which may not always hold true in real-world data.
- Struggles with highly correlated features or when the independence assumption is severely violated.
- Can be outperformed by more advanced algorithms in complex scenarios or when the underlying assumptions are not met.
- Sensitive to the presence of redundant features.
- Continuous features may require additional preprocessing (e.g., discretization, kernel density estimation) to be used in Naïve Bayes models.

Despite its simplicity and assumptions, Naïve Bayes models can perform surprisingly well in many real-world applications, especially when the feature independence assumption is roughly met or when the dataset is not too complex. However, for more complex scenarios or highly correlated features, other algorithms like decision trees, random forests, or neural networks may be more appropriate.

3. **Neural Network**

Logistic Regression for classification
Objective of Logistic Regression: Cross Entropy
Three activation functions for Neural Network
Understand the role of regularization to prevent overfitting

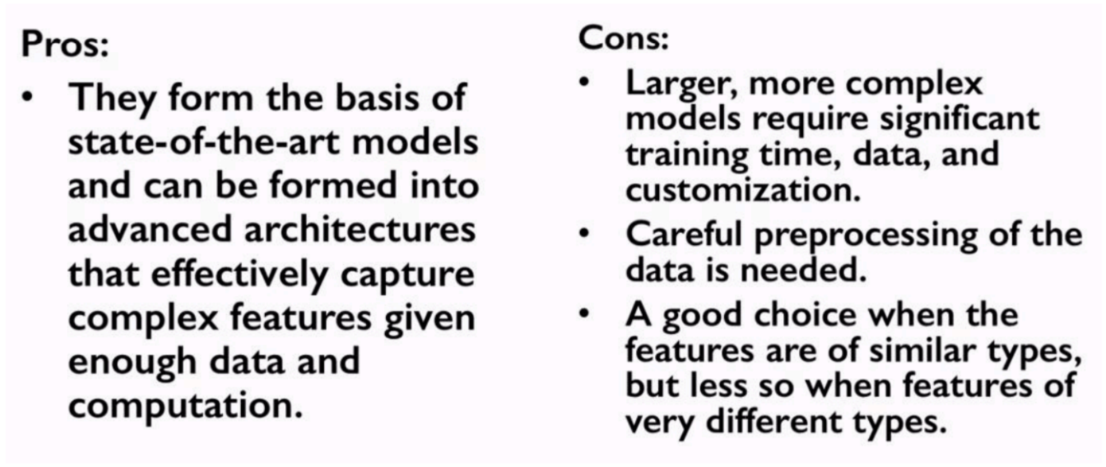
How to train a neural network?
1/ Feedforward Neural Network: How to compute output from input and edge weights
2/ Backpropagation Algorithm: How to backpropagate error from output to hidden layers
3/ Understand the updating of weights
>>> Conduct 1,2,3 iteratively for multiple rounds.

The training process typically involves the following steps:

1. Initialize the weights and biases of the neural network with small random values.
2. Forward propagate the input data through the network to obtain the output predictions.

3. Calculate the loss between the predicted output and the expected output using a loss function.
4. Back-propagate the errors from the output layer to the input layer, computing the gradients of the loss function with respect to the weights and biases.
5. Update the weights and biases using gradient descent, adjusting them in the direction that minimizes the loss function.
6. Repeat steps 2-5 for multiple epochs, iterating over the entire training dataset until the loss converges or a desired performance is achieved.

Advantages and disadvantages of Neural Networks



Pros:

- They form the basis of state-of-the-art models and can be formed into advanced architectures that effectively capture complex features given enough data and computation.

Cons:

- Larger, more complex models require significant training time, data, and customization.
- Careful preprocessing of the data is needed.
- A good choice when the features are of similar types, but less so when features of very different types.

4. **Decision Tree (classification) =** a process to split a dataset into subsets based on features.

   Objective: to purify the labels of instances in each subset.

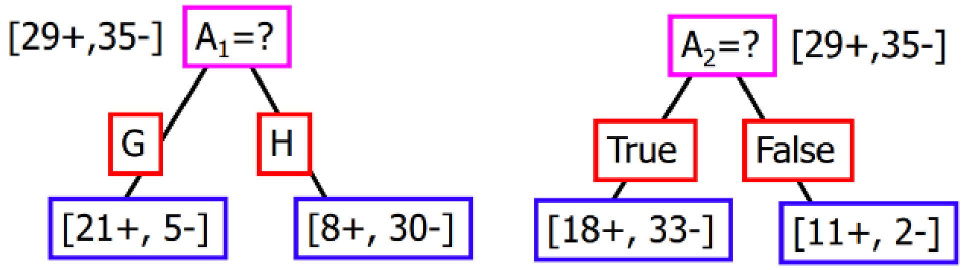
   Criteria (to split the dataset): **information gain** (the change of Entropy before and after dataset splitting).

   Compute Entropy and Information Gain

- Gain(S,A): expected reduction in entropy due to sorting S on attribute A

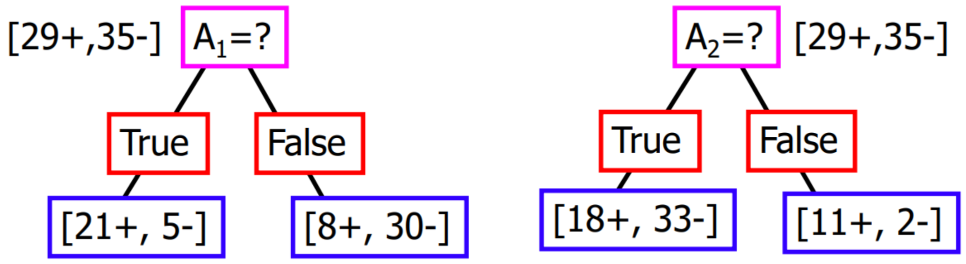
$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in D_A} \frac{|S_v|}{|S|} Entropy(S_v)$$

Entropy([29+,35-]) = -29/64 log2 29/64 – 35/64 log2 35/64
= 0.99



Entropy([21+,5-]) = 0.71
Entropy([8+,30-]) = 0.74
Gain(S,A1)=Entropy(S)
-26/64*Entropy([21+,5-])
-38/64*Entropy([8+,30-])
=0.27

Entropy([18+,33-]) = 0.94
Entropy([11+,2-]) = 0.62
Gain(S,A2)=Entropy(S)
-51/64*Entropy([18+,33-])
-13/64*Entropy([11+,2-])
=0.12



How to prevent overfitting of a Decision Tree
Overfitting occurs **when the decision tree model becomes too complex** and captures the noise or irregularities in the training data, leading to poor generalization performance on unseen data.

Here are some solutions.
Pruning

- Pre-pruning: This method stops growing the tree before it becomes too complex, based on a predefined stopping criterion, such as a maximum depth or a minimum number of samples in a node.
- Post-pruning: This technique involves growing a large tree first and then removing subtrees that do not improve the model's performance on a validation set or based on a complexity measure like cost-complexity pruning.

Minimum Sample Leaf Node: Setting a minimum number of samples required in a leaf node can prevent the tree from generating very specific rules based on too few instances.

Maximum Leaf Nodes: Limiting the maximum number of leaf nodes in the tree can control its complexity and prevent overfitting.

Feature Selection: Removing irrelevant or redundant features from the dataset can reduce noise and the chances of overfitting.

Ensemble Methods

- **Random Forests:** Building an ensemble of decision trees, where each tree is trained on a random subset of features and instances, can help reduce overfitting and improve generalization.
- **Setting Maximum Depth:** Limiting the maximum depth of the tree can prevent it from growing too deep and overfitting the training data.
- **Boosting**: Boosting algorithms like Gradient Boosting iteratively build an ensemble of weak decision trees, where each subsequent tree tries to correct the errors made by the previous ones, leading to better generalization.
- **Regularization:** Introducing a cost or penalty term for model complexity during the tree-building process can discourage overfitting.
- **Cross-Validation:** Using cross-validation techniques to evaluate the model's performance on unseen data and tuning the hyperparameters accordingly can help prevent overfitting.

It's important to note that overfitting is **a trade-off between model complexity and generalization performance.** The chosen technique(s) should balance the model's ability to capture the underlying patterns in the data while avoiding capturing noise or irrelevant details.

Advantages and Disadvantages of Decision Trees

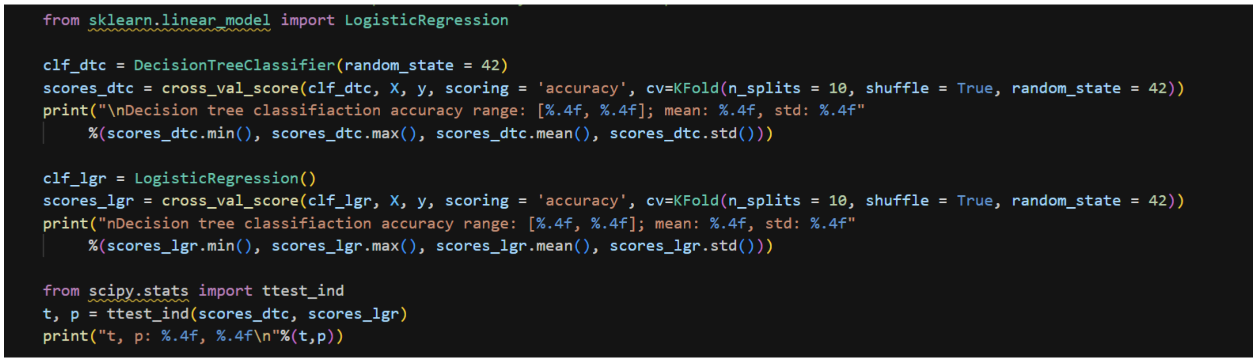
Advantages

- Easy to Understand
- Useful in Data exploration
- Less data cleaning required
    - It is not influenced by outliers and missing values to a fair degree.

- Data type is not a constraint
    - It can handle both numerical and categorical variables.

Disadvantages

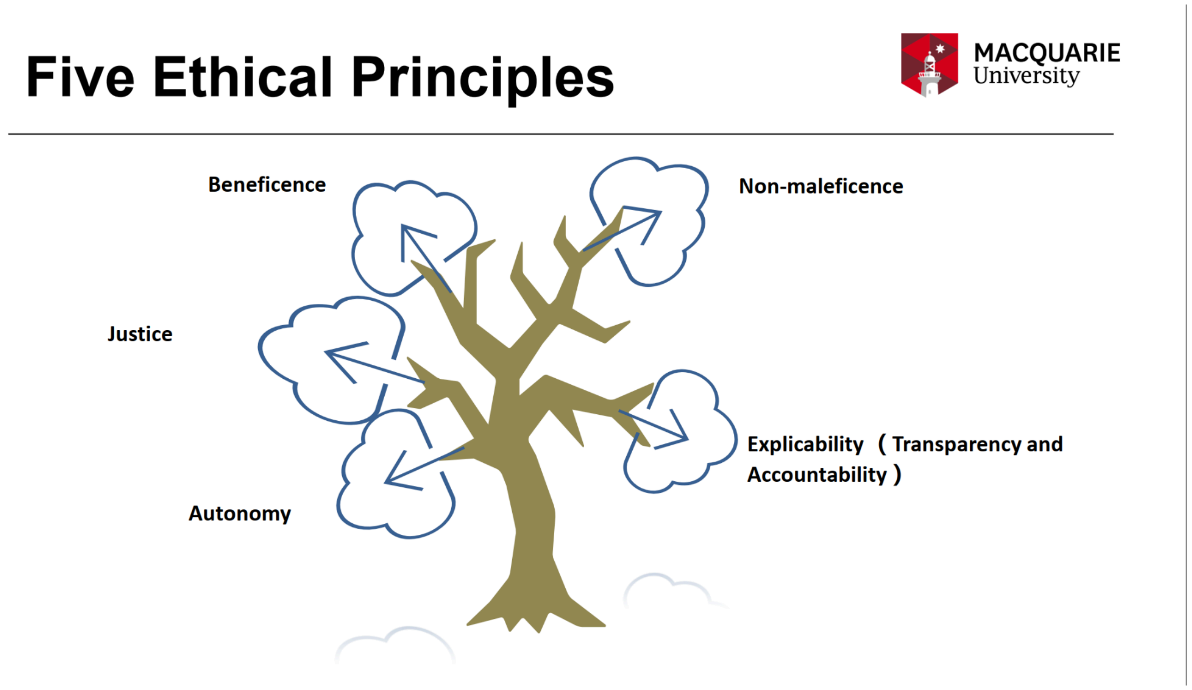
- Easy to overfit
- Scalability issues. As the number of features and/or instances in the dataset increases, the construction and interpretation of decision trees can become computationally expensive and complex.
- Not fit for continuous variables
    - While working with continuous numerical variables, the decision tree loses information when it categorizes variables in different categories.



```python
from sklearn.linear_model import LogisticRegression

clf_dtc = DecisionTreeClassifier(random_state = 42)
scores_dtc = cross_val_score(clf_dtc, X, y, scoring = 'accuracy', cv=KFold(n_splits = 10, shuffle = True, random_state = 42))
print("\nDecision tree classifiaction accuracy range: [%.4f, %.4f]; mean: %.4f, std: %.4f"
    %(scores_dtc.min(), scores_dtc.max(), scores_dtc.mean(), scores_dtc.std()))

clf_lgr = LogisticRegression()
scores_lgr = cross_val_score(clf_lgr, X, y, scoring = 'accuracy', cv=KFold(n_splits = 10, shuffle = True, random_state = 42))
print("nDecision tree classifiaction accuracy range: [%.4f, %.4f]; mean: %.4f, std: %.4f"
    %(scores_lgr.min(), scores_lgr.max(), scores_lgr.mean(), scores_lgr.std()))

from scipy.stats import ttest_ind
t, p = ttest_ind(scores_dtc, scores_lgr)
print("t, p: %.4f, %.4f\n"%(t,p))
```

Questions:
1/ Which model is regression and which is classification? Or both?
2/ How to compute the distance between two clusters in hierarchical clustering?

Week 5:



1. Data Ownership

2. Informed Consent

3. Intellectual Property

4. Data Privacy

5. Right To Be Forgotten

6. Dataset Bias

7. Data Quality

8. Algorithm Fairness

9. Misrepresentation

10. Free Choice