TechNeeds
Week 5 Notes
ML UPSKILL PROGRAM
Random Forest

1. Introduction

- Random Forest is an ensemble learning method primarily used for classification and regression.
- It builds multiple decision trees and merges them to get a more accurate and stable prediction.

2. Key Concepts

- **Ensemble Learning**: Combining the predictions of multiple models to improve performance.
- **Decision Tree**: A tree-like model used to make predictions based on the input features.
- **Bootstrap Aggregation (Bagging)**: Technique to reduce variance by training each tree on a different random subset of the data.

3. How Random Forest Works

1. **Data Sampling**: Randomly select samples from the training dataset with replacement (bootstrap sampling).
2. **Tree Construction**: For each sample, construct a decision tree:
    - At each node, a random subset of features is selected.
    - The best feature and split are chosen based on a criterion (e.g., Gini impurity for classification, mean squared error for regression).
3. **Prediction Aggregation**: Predictions from all trees are combined:
    - For classification: Majority voting is used.
    - For regression: The average of all predictions is taken.

4. Advantages

- **Accuracy**: Generally more accurate than individual decision trees.
- **Robustness**: Less likely to overfit compared to individual decision trees.
- **Feature Importance**: Provides an estimate of feature importance, which can be useful for feature selection.

5. Disadvantages

- **Complexity**: More complex and computationally intensive than single decision trees.
- **Interpretability**: Less interpretable than single decision trees due to the ensemble nature.

6. Hyperparameters

- **Number of Trees (`n_estimators`)**: The number of decision trees in the forest.
- **Maximum Depth (`max_depth`)**: The maximum depth of each tree.
- **Minimum Samples Split (`min_samples_split`)**: The minimum number of samples required to split an internal node.
- **Minimum Samples Leaf (`min_samples_leaf`)**: The minimum number of samples required to be at a leaf node.
- **Maximum Features (`max_features`)**: The number of features to consider when looking for the best split.

7. Feature Importance

- Random Forest can rank the importance of features in the dataset.
- Importance can be measured by the decrease in impurity or by permutation importance.

8. Practical Tips

- **Hyperparameter Tuning**: Use techniques like Grid Search or Random Search to find the best hyperparameters.
- **Avoiding Overfitting**: Control tree depth and minimum samples at leaf nodes.
- **Handling Imbalanced Data**: Use techniques like class weighting or oversampling.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Load data
X, y = load_data()  # Replace with actual data loading
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize Random Forest
rf = RandomForestClassifier(n_estimators=100, max_depth=None, random_state=42)

# Train the model
rf.fit(X_train, y_train)
# Make predictions
y_pred = rf.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Feature Importance
importances = rf.feature_importances_
for i, imp in enumerate(importances):
    print(f"Feature {i}: {imp}")
```

Bayesian Network

1. Introduction

- Bayesian Networks (BNs), also known as Belief Networks or Bayes Nets, are probabilistic graphical models representing a set of variables and their conditional dependencies using a directed acyclic graph (DAG).
- They are used to model uncertainty in complex systems, allowing reasoning and inference based on observed data.

2. Key Concepts

- **Nodes**: Represent random variables, which can be discrete or continuous.
- **Edges**: Directed edges between nodes represent conditional dependencies between the variables.
- **Parents**: Nodes that have direct edges pointing to a child node.
- **Conditional Probability Table (CPT)**: Each node has an associated CPT that quantifies the effect of the parent nodes on the node.

3. Structure of a Bayesian Network

- **Directed Acyclic Graph (DAG)**: A graph with directed edges and no cycles.
- **Joint Probability Distribution**: The joint probability of a set of variables can be decomposed into the product of conditional probabilities defined by the network structure:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \text{Parents}(X_i))$$

- **Local Markov Property**: A variable is conditionally independent of its non-descendants given its parents.

4. Inference in Bayesian Networks

- **Exact Inference**: Algorithms like Variable Elimination, Belief Propagation, and Junction Tree.
- **Approximate Inference**: Algorithms like Monte Carlo methods (e.g., Gibbs Sampling), Variational Inference.

5. Learning in Bayesian Networks

- **Parameter Learning**: Estimating the CPTs given the network structure and data.
    - **Maximum Likelihood Estimation (MLE)**: Estimating parameters by maximizing the likelihood of the observed data.
    - **Bayesian Estimation**: Incorporating prior knowledge and updating beliefs based on observed data.
- **Structure Learning**: Determining the network structure from data.
    - **Score-Based Methods**: Search for the structure that maximizes a scoring function (e.g., BIC, AIC).
    - **Constraint-Based Methods**: Use statistical tests to identify conditional independencies and build the structure.

6. Advantages

- **Interpretability**: The graphical representation provides a clear understanding of variable dependencies.
- **Flexibility**: Can model complex relationships and incorporate prior knowledge.
- **Probabilistic Reasoning**: Allows for reasoning under uncertainty and updating beliefs with new evidence.

7. Disadvantages

- **Computational Complexity**: Exact inference can be computationally intensive, especially for large networks.
- **Data Requirements**: Requires sufficient data to accurately estimate conditional probabilities.
- **Structure Learning**: Determining the optimal network structure can be challenging.

8. Applications

- **Medical Diagnosis**: Modeling diseases and symptoms to aid in diagnosis.
- **Fault Diagnosis**: Identifying causes of failures in systems.
- **Decision Support Systems**: Assisting in decision-making under uncertainty.
- **Natural Language Processing**: Modeling relationships between words and concepts.

9. Advanced Topics

- **Dynamic Bayesian Networks (DBNs):** Extends Bayesian Networks to model temporal processes by connecting variables over time.
- **Hidden Markov Models (HMMs):** A type of DBN used for time series data where the system being modeled is assumed to be a Markov process with hidden states.
- **Influence Diagrams:** Extension of Bayesian Networks used for decision analysis, incorporating decision nodes and utility nodes.

10. Conclusion

- Bayesian Networks provide a robust framework for modeling and reasoning about uncertainty in complex systems.
- They offer interpretability, flexibility, and the ability to update beliefs based on new evidence, making them valuable for various applications.
- Despite computational challenges, advances in algorithms and computational power continue to enhance their practical utility.

```
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
# Define the structure of the Bayesian Network
model = BayesianNetwork([('A', 'C'), ('B', 'C')])

# Define the CPDs (Conditional Probability Distributions)
cpd_a = TabularCPD(variable='A', variable_card=2, values=[[0.2], [0.8]])
cpd_b = TabularCPD(variable='B', variable_card=2, values=[[0.7], [0.3]])
cpd_c = TabularCPD(variable='C', variable_card=2,
                   values=[[0.9, 0.6, 0.7, 0.1],
                           [0.1, 0.4, 0.3, 0.9]],
                   evidence=['A', 'B'], evidence_card=[2, 2])

# Add CPDs to the model
model.add_cpds(cpd_a, cpd_b, cpd_c)

# Check the model for correctness
assert model.check_model()

# Perform inference
inference = VariableElimination(model)
result = inference.query(variables=['C'], evidence={'A': 1, 'B': 0})
print(result)
```