TechNeeds
Week 5 Notes
ML UPSKILL PROGRAM
Gradient boosting

1. Introduction

- Gradient Boosting is an ensemble learning method used for regression and classification tasks.
- It builds models sequentially, each new model attempting to correct the errors of the previous ones.

2. Key Concepts

- **Ensemble Learning**: Combining predictions from multiple models to improve performance.
- **Boosting**: An iterative technique that adjusts the weights of incorrectly predicted instances to focus on them in subsequent models.
- **Gradient Descent**: An optimization algorithm used to minimize a loss function by iteratively moving towards the minimum.

3. How Gradient Boosting Works

1. **Initialize Model**: Start with an initial model (e.g., a simple model that predicts the mean of the target variable for regression).
2. **Compute Residuals**: Calculate the difference between the actual values and the predictions (residuals).
3. **Fit Base Learner**: Train a new model (base learner) on the residuals.
4. **Update Model**: Add the new model to the ensemble, adjusting its contribution with a learning rate.
5. **Iterate**: Repeat steps 2-4 for a specified number of iterations or until convergence.

4. Loss Functions

- **Regression**: Mean Squared Error (MSE), Mean Absolute Error (MAE).
- **Classification**: Log Loss (for binary classification), Multinomial Loss (for multiclass classification).

5. Components of Gradient Boosting

- **Base Learner**: Typically a decision tree, but it can be any model.
- **Loss Function**: Measures how well the model is performing.
- **Learning Rate**: A parameter that scales the contribution of each base learner, preventing overfitting.
- **Number of Estimators**: The number of boosting stages (i.e., the number of base learners).

6. Advantages

- **High Accuracy**: Often provides state-of-the-art results for many machine learning tasks.
- **Flexibility**: Can optimize a variety of loss functions and be used with different base learners.
- **Feature Importance**: Provides insights into the importance of features.

7. Disadvantages

- **Computationally Intensive**: Training can be slow, especially with large datasets.
- **Sensitive to Overfitting**: Requires careful tuning of hyperparameters.
- **Complexity**: More difficult to interpret than simpler models like decision trees.

8. Hyperparameters

- **Number of Trees (`n_estimators`)**: The number of boosting stages to be run.
- **Learning Rate (`learning_rate`)**: The rate at which the boosting algorithm updates the model.
- **Maximum Depth (`max_depth`)**: The maximum depth of the individual decision trees.
- **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.
- **Subsample**: The fraction of samples to be used for fitting the individual base learners.

9. Practical Tips

- **Hyperparameter Tuning**: Use techniques like Grid Search or Random Search to find the best hyperparameters.
- **Early Stopping**: Use a validation set to stop training when performance stops improving.
- **Feature Engineering**: Creating new features or transforming existing ones can significantly improve model performance.

10. Advanced Variants

- **XGBoost**: An optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable.

- **LightGBM**: A fast, distributed, high-performance gradient boosting framework that uses tree-based learning algorithms.
- **CatBoost**: A gradient boosting library that handles categorical features automatically.

```python
from sklearn.ensemble import GradientBoostingClassifier
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 Gradient Boosting
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)

# Train the model
gb.fit(X_train, y_train)

# Make predictions
y_pred = gb.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

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

Decision Trees

1. Introduction

- Decision Trees are a type of supervised learning algorithm used for both classification and regression tasks.
- They are simple yet powerful tools that model decisions and their possible consequences in a tree-like structure.

2. Key Concepts

- **Root Node**: The topmost node representing the entire dataset, which gets split into subsets.
- **Decision Node**: A node that splits into two or more sub-nodes.
- **Leaf Node**: The final node that doesn't split further, representing a class label (for classification) or a continuous value (for regression).
- **Splitting**: The process of dividing a node into two or more sub-nodes.
- **Branch/Sub-Tree**: A subsection of the entire tree structure.
- **Pruning**: The process of removing sub-nodes of a decision node, reducing the complexity of the model.

3. How Decision Trees Work

- **For Classification**:
    1. **Select Best Attribute**: Choose the attribute that best separates the data using criteria like Gini impurity or Information Gain.
    2. **Split Data**: Divide the data into subsets based on the best attribute.
    3. **Repeat**: Recursively apply the above steps to each subset until stopping criteria are met (e.g., all data points belong to the same class or a maximum depth is reached).
- **For Regression**:
    1. **Select Best Split**: Choose the split that minimizes the variance or mean squared error in the subsets.
    2. **Split Data**: Divide the data based on the best split.
    3. **Repeat**: Recursively apply the above steps to each subset until stopping criteria are met (e.g., the variance is below a threshold or a maximum depth is reached).

4. Splitting Criteria

- **Gini Impurity**: Measures the impurity of a node; a node is pure if all samples belong to one class.

- **Information Gain**: Measures the reduction in entropy or impurity; higher information gain indicates a better split.
- **Variance Reduction**: Used in regression, measures how much a split reduces the variance within the subsets.

5. Advantages

- **Simple and Intuitive**: Easy to understand and interpret.
- **Non-parametric**: No assumptions about the data distribution.
- **Versatile**: Can handle both numerical and categorical data.

6. Disadvantages

- **Overfitting**: Can easily overfit the training data, especially with deep trees.
- **Unstable**: Small changes in data can lead to completely different trees.
- **Bias towards Features with More Levels**: Features with many levels can dominate the splits.

7. Hyperparameters

- **Maximum Depth (`max_depth`)**: The maximum depth of the 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.
- **Criterion (`criterion`)**: The function to measure the quality of a split (e.g., `gini` for Gini impurity, `entropy` for information gain).

8. Pruning

- **Pre-Pruning (Early Stopping)**: Stop growing the tree before it reaches the maximum depth by setting constraints like `max_depth`, `min_samples_split`, and `min_samples_leaf`.
- **Post-Pruning**: Allow the tree to grow fully and then remove nodes that provide little power.

9. Advanced Topics

- **Ensemble Methods**: Improve decision trees by combining multiple trees.
    - **Random Forest**: An ensemble of decision trees using bagging and random feature selection.
    - **Gradient Boosting**: Sequentially adds decision trees to correct errors made by previous trees.

- **Feature Importance**: Decision trees can rank the importance of features based on their contribution to reducing impurity.

```python
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error

# 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)

# For Classification
# Initialize Decision Tree Classifier
clf = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Classification Accuracy: {accuracy}")
```

```python
# For Regression
# Initialize Decision Tree Regressor
reg = DecisionTreeRegressor(criterion='mse', max_depth=3, random_state=42)

# Train the model
reg.fit(X_train, y_train)

# Make predictions
y_pred = reg.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Regression Mean Squared Error: {mse}")

# Visualize the tree (optional, requires graphviz)
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
plot_tree(clf, filled=True, feature_names=feature_names, class_names=class_names)
plt.show()
```