BAX 452 HW 3

Course Instructor : Dr. Rahul Makhijani

Submit the assignment on canvas. The deadline is Tuesday Feb 11 11:59 pm

Bagging and Boosting on California Housing DataSet 20 points

1. Using the `RandomForestRegressor` from `sklearn.ensemble`, train a Random Forest model on the California Housing Dataset. Experiment with `n_estimators=100`, `random_state=42`. Train the model and print its default parameters.

2. **Tune the hyperparameters:**

   - **Task:** Use Grid Search or Random Search (`GridSearchCV` or `RandomizedSearchCV`) to optimize key hyperparameters such as:
     - Number of trees (`n_estimators`).
     - Maximum depth of trees (`max_depth`).
     - Minimum samples per leaf (`min_samples_leaf`).
   - **Question:** How does each parameter (above) influence the model's performance?

3. How does Gradient Boosting perform compared to other methods?

   - Create a residual plot (actual values vs. predicted values) to analyze the model's errors.
   - Are there any observable patterns in the prediction errors? Are errors larger for certain ranges of house prices?
   - Calculate the percentage error for high-price and low-price houses. Does the model perform equally well across the entire target range?
   - Observe how a very low or high value affects model accuracy, convergence speed, and overfitting.

4. **Experiment with the learning rate:**

  - Observe how a very low or high value affects model accuracy, convergence speed, and overfitting.

IRIS multi-classifcation

Dataset Exploration and Preparation

- Select the *Iris* dafrom scikit-learn for classification. Load the dataset using `sklearn.datasets.load_*` functions.
- Display the dataset's feature names, target names, and a sample from the dataset. **Question:** Is the dataset well-balanced across the class labels? Comment on the distribution of target labels.

Data Preprocessing

- Split the dataset into training and testing datasets (80/20 split) using `train_test_split`.
- Normalize the feature values using `StandardScaler`.
- Explain the importance of feature scaling in KNN.

Implementing K-Nearest Neighbors

- Use scikit-learn's `KNeighborsClassifier` to train the KNN model.
- Train the model using the default parameters (`n_neighbors=5`, `metric='minkowski'`, `p=2` for Euclidean distance).
  - Fit the model on the training dataset and test the performance in terms of F1 score, precision and recall on the test set.
  - **Evaluate the impact of different values of k (`n_neighbors`):**
  - Train the model for different values of `k` ranging from 1 to 20.
  - Create a line plot of `k` vs. accuracy (for classification).
  - Find the optimal value of `k`.
  - **Question:** What value of `k` gives the best performance? Explain why the choice of `k` affects the model's performance.

Model Evaluation

- Generate a confusion matrix and a classification report (using `classification_report`) for `y_test` predictions.
- Interpret the confusion matrix and explain the precision, recall, and F1-score for each target class.
- Tune the KNN model's hyperparameters.
  - Perform grid search or random search (`GridSearchCV`
    - Number of neighbors (`n_neighbors`).
    - Distance metrics (`metric`, e.g., Euclidean, Manhattan).
    - Weighting schemes (`weights`, e.g., `uniform`, `distance`).
  - **Question:** How does choosing different distance metrics (e.g., Euclidean vs. Manhattan) affect model performance?

Comparison with Other Algorithms

  - Compare KNN's performance with other models like:
    - Multinomial Logistic Regression.
    - Random Forest
  - Explore the impact of dropping certain features on the KNN model's performance.
- **Visualization:**
  - For 2D datasets (e.g., *Iris*), visualize decision boundaries using `matplotlib` for different values of `k`.
  - Comment on how decision boundaries change as `k` increases.

Implementing K-Means Clustering from Scratch (Iris Dataset)

The goal of this problem is to implement the K-Means clustering algorithm from scratch without relying on pre-built libraries like scikit-learn (except for loading or preprocessing the dataset). You will cluster data points into k groups using only the numeric features from the **Iris dataset**.

1. **Dataset Preparation:**
    * Load the Iris dataset from `sklearn.datasets`.
    * Use only the numeric features (*sepal length*, *sepal width*, *petal length*, *petal width*).
    * Normalize the features using a scaler (e.g., StandardScaler or MinMaxScaler).
2. **K-Means Implementation:**
    * Randomly initialize k centroids, choosing k data points randomly from the dataset. (**Note** this is slightly different from the initialization we had in class but this works as well)
    * Assign each data point to the nearest cluster centroid by calculating the Euclidean distance.
    * Update the cluster centroids to the mean of all points assigned to each cluster.
    * Repeat these steps until:
        · The cluster assignments do not change.
        · A maximum number of iterations (e.g., 100) is reached.
    * Implement the algorithm in Python, structuring it with reusable helper functions:
        · A function to calculate distances between points and centroids.
        · A function to assign clusters.
        · A function to update centroids.
3. **Evaluation:**
    * Use the **Elbow Method**:

  - · Apply your K-Means implementation for k ranging from 1 to 10.
  - · Calculate the total within-cluster variance for each value of k.
  - · Plot the total within-cluster variance values to identify the optimal number of clusters (elbow point).

4. **Cluster Visualization:**
   - ∗ Reduce the numeric data to 2D or 3D using **PCA** (Principal Component Analysis).
   - ∗ Visualize the clusters and centroids using a scatter plot.

5. **Analysis and Implementation Questions:**
   - ∗ What are the final centroids of the clusters for the chosen value of k?
   - ∗ How does the within-cluster variance change as k increases?
   - ∗ Does your algorithm perform well for the Iris dataset? Why or why not?

Decision Tree

Consider a two-category classification task with the following training data:

attr1	**attr2**	**attr3**	**attr4**	**class**
a	1	c	-1	c1
b	0	c	-1	c1
a	0	c	1	c1
b	1	c	1	c1
b	0	c	1	c2
a	0	a	-1	c2
a	1	a	-1	c2
b	1	c	-1	c2

Table 1: Training Data for Two-Category Classification Task

1. Using the training data provided in Table 1, construct a complete, unpruned decision tree.
2. Use **information gain** as your splitting criterion for selecting the best attribute at each node.
3. Show all calculations for:
   * The entropy of the dataset at each node.
   * The weighted entropy for splits on each attribute.
   * The information gain for each attribute.
4. Continue splitting until all examples in a node belong to the same class.

Hints and Definitions

* The formula for **entropy** for a dataset D is:

$$H(D) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

  where p_i is the proportion of examples belonging to class i, and C is the number of classes.

* The **information gain** for a split on attribute A is:

$$IG(D, A) = H(D) - \sum_{v \in \text{values}(A)} \frac{|D_v|}{|D|} H(D_v)$$

  where:
  - $H(D)$ is the entropy of dataset D before the split.
  - D_v is the subset of D where attribute A has value v.
  - $|D_v|/|D|$ is the weight of subset D_v relative to the entire dataset.
  - $H(D_v)$ is the entropy of subset D_v.
* Use the provided formulas for entropy and information gain to decide the best attribute to split at each step.