BSDS-203 - Machine Learning for DS

Course Instructor - Dr. Aashima Bangia

Gauri Sharan - BSc Data Science (Sem 3)

January 6, 2024

Outline

- Overview of Machine Learning
- Supervised Learning Algorithms
- Ensemble Learning
- Neural Networks Basics
- Evaluating Models through Performance Metrics

Overview of Machine Learning

- Machine learning is a subset of artificial intelligence that focuses on the development of algorithms and models enabling computers to learn from data.
- Applications in data science involve predicting outcomes, uncovering patterns, and making data-driven decisions.
- Types: Supervised learning (labeled data), Unsupervised learning (unlabeled data), Semi-supervised learning (combination of both).
- Components of Learning Models: Geometric Models (e.g., SVM), Probabilistic Models (e.g., Naïve Bayes), Logic Models (e.g., Decision Trees).
- Essential concepts of matrix-vector operations in data representation, handling features.

Supervised Learning Algorithms

- **Regression:** Predicting a continuous outcome.
  - *Linear Regression:* Modeling the relationship between a dependent variable and one or more independent variables using a linear equation.
  - *Multiple Linear Regression:* Extending linear regression to multiple independent variables.
- **Classification:** Predicting a categorical outcome.
  - *Logistic Regression:* Used for binary classification problems.
  - *KNN (K-Nearest Neighbors):* Assigns a class label based on the majority class among its k-nearest neighbors.
  - *Naïve Bayes:* Probability-based algorithm using Bayes' theorem for classification.
  - *Support Vector Machines:* Constructs a hyperplane to separate classes in feature space.

Supervised Learning Algorithms

Decision Trees: Recursive partitioning of data based on feature values.

- *Random Forests:* Ensemble method using multiple decision trees.
- *Gradient Boosting:* Builds trees sequentially, correcting errors of previous models.

Clustering Algorithms: Grouping similar data points together.

- *k-means:* Partitioning data into k clusters based on mean values.
- *Hierarchical Clustering:* Agglomerative or divisive clustering methods.
- *DBSCAN (Density-Based Spatial Clustering of Applications with Noise):* Clustering based on density.
- *Fuzzy Clustering:* Assigning membership values to data points.

Supervised Learning Algorithms

Dimensionality Reduction: Reducing the number of features.

- *PCA (Principal Component Analysis):* Transforming data into principal components.
- *t-SNE (t-Distributed Stochastic Neighbor Embedding):* Visualization of high-dimensional data in two or three dimensions.

Association Rule Mining and Frequent Pattern Mining: Identifying patterns in data, often used in market basket analysis.

Ensemble Learning

- **Introduction to Bagging & Boosting Techniques:** Ensemble methods combining multiple models for improved performance.
- **Types of Bagging Algorithms:**
  - *Random Forest:* Ensemble of decision trees, each trained on a random subset of the data.
  - *Bootstrapping:* Sampling with replacement to create multiple datasets.
- **Types of Boosting Algorithms:**
  - *Adaboost:* Boosting algorithm giving more weight to misclassified data points.
  - *XGBoost:* Extreme Gradient Boosting, popular for its speed and performance.
  - *CatBoost:* Gradient boosting library designed for categorical features.

Ensemble Learning

- **Model Combination Schemes:**
  - *Voting:* Combining predictions from multiple models.
  - *Error-Correcting Output Codes:* Coding scheme for multi-class classification problems.
- **Gaussian Mixture Models & EM Algorithm:**
  - *Expectation-Maximization (EM) Algorithm:* Iterative method for finding maximum likelihood estimates of parameters in probabilistic models.
  - *Information Criteria:* Metrics for model selection based on likelihood and complexity.
  - *Distance Measures:* Measures for comparing similarity or dissimilarity between data points.

Ensemble Learning

- **Model Selection Techniques:**
  - *Cross-validation:* Dividing data into multiple subsets for training and testing.
  - *Hyperparameter Tuning:* Adjusting parameters to optimize model performance.

Basics of Neural Networks

- **Activation Functions:** Non-linear functions applied to neuron outputs.
  - *Sigmoid:* Outputs values between 0 and 1, commonly used in the output layer for binary classification.
  - *Softmax:* Converts raw scores into probability distributions for multi-class classification.
- **Loss Function, Cross Entropy Loss Functions, Descent:** Metrics used to evaluate model performance during training.

Basics of Neural Networks

- **Optimizers:** Algorithms for updating model parameters during training.
  - *RMSprop:* Adaptive learning rate method.
  - *Adam:* Adaptive moment estimation, combines ideas from RMSprop and momentum.
  - *Gradient Descent:* Standard optimization method.
  - *Stochastic Gradient Descent:* Variant of gradient descent using a random subset of data.
- **Types of Neural Networks:**
  - *ANN (Artificial Neural Network):* Basic structure of connected nodes (neurons) resembling the human brain.
  - *RNN (Recurrent Neural Network):* Designed for sequential data, with connections forming cycles.
  - *CNN (Convolutional Neural Network):* Specialized for processing grid-like data, e.g., images.

Evaluating Models through Performance Metrics

- **Need for Evaluation, Threshold, Adjusting Thresholds:** Importance of assessing model performance and adjusting decision thresholds.
- **AUC ROC Curve:** Receiver Operating Characteristic curve illustrating trade-offs between sensitivity and specificity.
- **Performance Metrics:**
  - *Confusion Matrix:* Matrix representing true positives, true negatives, false positives, and false negatives.
  - *Precision:* Proportion of true positives among all predicted positives.
  - *Accuracy:* Overall correctness of the model.
  - *F-score:* Harmonic mean of precision and recall.
  - *G-score:* Geometric mean of precision and recall.

Evaluating Models through Performance Metrics

- **Sensitivity/Specificity with Mathematical Concepts:** Understanding true positive rate (sensitivity) and true negative rate (specificity).
- **Handling Imbalanced Datasets and Model Interpretation Techniques:**
  - *Under-sampling:* Reducing the number of instances from the over-represented class.
  - *Over-sampling:* Increasing the number of instances in the under-represented class.
  - *Bias:* Systematic errors in model predictions.
  - *Variance:* Model's sensitivity to changes in the training set.
  - *Model Complexity:* Balance between model simplicity and capturing underlying patterns.

Evaluating Models through Performance Metrics

- **Cross Validation and Hyperparameter Tuning for Model Selection:**
  - *Grid Search:* Exhaustive search over specified parameter values.
  - *Randomized Search Approach:* Random sampling of hyperparameter combinations.
  - *GridSearchCV Parameters:* Parameters used in grid search.
  - *Select Best Model:* Choosing the model with the best performance.
  - *Randomized Search:* Randomized search for hyperparameter tuning.

Hands on using Python

In [9]:
```
decision_tree = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)

decision_tree.fit(X_train, y_train)
```

Out[9]:
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [10]:
```
y_pred = decision_tree.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 83.33%
```

In [11]:
```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

plt.figure(figsize=(12, 8))
plot_tree(decision_tree, filled=True, feature_names=X.columns, class_names=['Species0', 'Species1'
plt.show()
```

Figure: Decision Tree Classifier - Iris Dataset

Hands on using Python

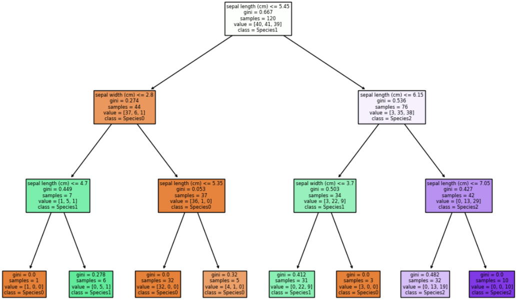

Figure: DT Classifier Output

Hands on using Python

```
In [12]:
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

```
In [13]:
regressor = DecisionTreeRegressor(max_depth=3, random_state=42)
regressor.fit(X_train, y_train)
```

```
Out[13]: DecisionTreeRegressor(max_depth=3, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [14]:
y_pred = regressor.predict(X_test)
```

```
In [15]:
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

```
Mean Squared Error: 0.2253159818558154
R-squared: 0.6776082930520924
```

```
In [16]:
from sklearn.tree import plot_tree

plt.figure(figsize=(10, 6))
plot_tree(regressor, filled=True, feature_names=X.columns)
plt.show()
```

Figure: Decision Tree Regressor

Hands on using Python

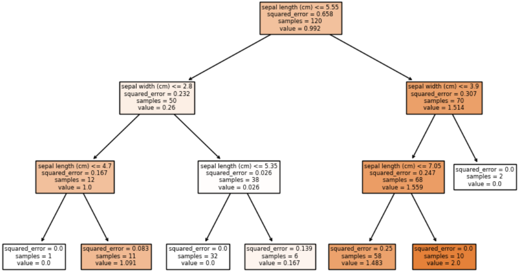

Figure: DT Regressor Output

Hands on using Python

```
In [21]: from imblearn.under_sampling import RandomUnderSampler
         rus = RandomUnderSampler(random_state=0)
         X_resample, y_resample = rus.fit_resample(X,y)
         len(X_resample)

Out[21]: 43388

In [29]: from imblearn.under_sampling import TomekLinks
         tl=TomekLinks(sampling_strategy='majority')
         X_resample_tl, y_resample_tl = tl.fit_resample(X, y)
         len(X_resample_tl)

Out[29]: 583127
```

Figure: Undersampling using TomekLink-Porto Surego Safe Driver Prediction

Hands on using Python

[5]:
```python
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(iris.data, iris.target,test_size=0.2, random_state=0)
```

[6]:
```python
import xgboost as xgb
train=xgb.DMatrix(x_train, label = y_train)
test=xgb.DMatrix(x_test, label = y_test)
```

[7]:
```python
param={
        'max_depth':4,
        'eta':0.3,
        'objective':'multi:softmax',
        'num_class':3}
epochs=10
```

[8]:
```python
model=xgb.train(param, train, epochs)
predictions=model.predict(test)
print('\n Predicted Values:', predictions)
```

Figure: XGBoost in iris dataset

Hands on using Python

```
Predicted Values: [2. 1. 0. 2. 0. 2. 0. 1. 1. 1. 2. 1. 1. 1. 1. 0. 1. 1. 0. 0. 2. 1. 0. 0.
2. 0. 0. 1. 1. 0.]
```

```
[9]: from sklearn.metrics import accuracy_score
a=accuracy_score(y_test, predictions)
print('\n Accuracy score:', a)
```

```
Accuracy score: 1.0
```

```
[10]: from xgboost import plot_importance
from matplotlib import pyplot
plot_importance(model)
pyplot.show()
```

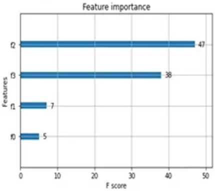

Figure: XGBoost in iris dataset (contd.)

Hands on using Python

```
[20]: from sklearn.cluster import KMeans
wcss = []
for k in range(1,11):
    kmeans = KMeans(n_clusters=k, init="k-means++")
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.figure(figsize=(12,6))
plt.grid()
plt.plot(range(1,11),wcss, linewidth=2, color="blue", marker ="8")
plt.xlabel("K Value")
plt.xticks(np.arange(1,11,1))
plt.ylabel("WCSS")
plt.show()
```

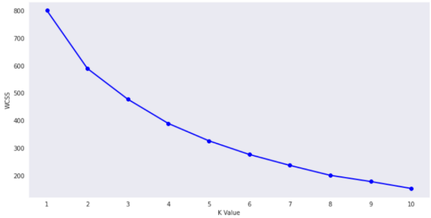

Figure: Applying elbow method on k-means clustering

Hands on using Python

```
[23]: km = KMeans(n_clusters = 5)
      clusters = km.fit_predict(df)
      df['label'] = clusters
      centroids = km.cluster_centers_
```

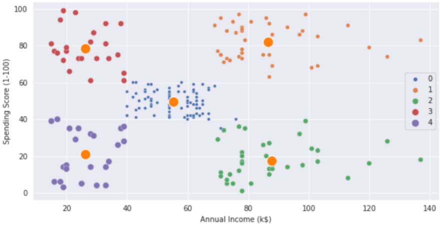

Figure: setting k = 5

Hands on using Python

```
from sklearn.cluster import DBSCAN
```

```
from sklearn import metrics
# Compute DBSCAN
db = DBSCAN(eps=0.45).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
```

```
print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Silhouette Coefficient: %0.3f"% metrics.silhouette_score(X, labels))
```

```
Estimated number of clusters: 6
Estimated number of noise points: 137
Silhouette Coefficient: -0.109
```

Figure: DBSCAN in Python

Hands on using Python

```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customer')
plt.ylabel('Euclidean Distance')
plt.show()
```

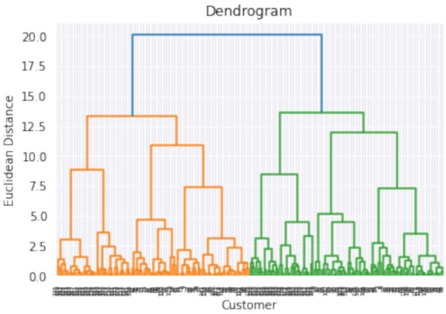

Figure: Hierarchical Clustering Dendogram

Hands on using Python

```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage =
    'ward')
labels = hc.fit_predict(X)
print(f"CLusters : {set(labels)}")
df['label'] = labels

print("Silhouette Coefficient: %0.3f"% metrics.silhouette_score(X, labels))

"""Label Plots for Annual Income (k$) and Spendng Score (1-100)"""
plt.figure(figsize = (10,5))
sns.scatterplot(df['Annual Income (k$)'], df['Spending Score (1-100)'], hue =
    df.label, palette="deep", size=df.label)
# sns.scatterplot(centroids[:,2] , centroids[:,3] , s = 200)
plt.show()
```

```
CLusters : {0, 1, 2}
Silhouette Coefficient: 0.248
```

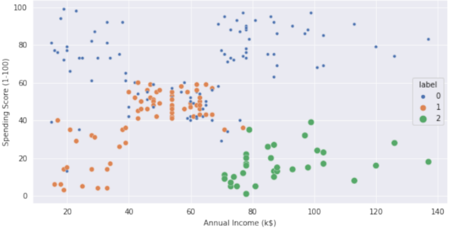

Figure: Agglomerative Clustering on the basis of dendogram above