ECE Department
CCEE460L
Artificial Intelligence Lab

Experiment 7

Name: Omar AL-Halabi(20230752)

ML

OBJECTIVES

In this Programming Project, you will create Machine learning models that will solve a Machine Learning problem of your choice and compare their performance. You need to solve one problem in supervised learning (using Naïve Bayes and another supervised learning approach) and one problem in unsupervised learning (using an unsupervised learning approach of your choice and an Agglomerative clustering method of your choice).
You will select a data set of your choice (either one you find online or one you make up yourself).

For each model (Supervised and Unsupervised learning) you will need to

(a) Describe your data set, clearly explaining your variables and what your model aims to predict.
(b) Clearly explain which Machine Learning techniques you used and why you chose those techniques.
(c) Separate your data set into 10 equal sized sets (each set will be roughly 10% of your original dataset. Repeat 10 times, training your model on 9 data sets and testing on the remaining data set (each time taking a different data set as your testing set). Report your results for each time and an average (precision, recall and F1 scores). If possible, you are strongly encouraged to use a ready-made package such as sci-kit learn to train or test your model.
(d) Write a short reflection (minimum 200 words, maximum 400 words) on the strengths and limitations of the methods you used, explaining why your model performed strongly or poorly. Provide at least two different ways that the accuracy of your model would be improved through further analysis and compare the different models in terms of performance and execution time.

Submit:

1. Complete code (.py/.java/.c),
2. A copy of your data set so that the TA can run your program using your dataset and verify your results.
3. A report that includes your answers to parts a, b, c and d for both the Supervised Learning Models and the Unsupervised Learning Models.

OUTCOMES

As a result of the lab session, the student will be able to:

- Code in Python.
- Deal with Datasets.
- Build and Test ML Models.

LAB SETUP

The lab can be performed on Google Colab or using any Python compiler/interpreter.

RESOURCES

The resources available to students during the lab session are:

- Internet;
- Lab Instructor;

Naïve Bayes Classifier

(a) Predict whether income exceeds $50K/yr based on census data. Also known as "Census Income" dataset.

(b) Naïve Bayes classification is a straightforward and powerful algorithm for the classification task. In this kernel, I implement Naive Bayes Classification algorithm with Python and Scikit-Learn. I built a Naive Bayes Classifier to predict whether a person makes over 50K a year.

(c)

```
Classification accuracy : 0.7972
Classification error : 0.2028
Precision : 0.7924
Recall or Sensitivity : 0.9298
True Positive Rate : 0.9298
False Positive Rate : 0.4449
Cross-validation scores:[0.80570175 0.79912281 0.8056165  0.80210619
0.79113646 0.79157525
 0.79947345 0.80605529 0.79245283 0.80210619]
Average cross-validation score: 0.7995
```

(d) The Naive Bayes classifier is praised for its ease of use, effectiveness, and ability to manage big, highly dimensional datasets. Because it assumes that characteristics are conditionally independent given the class label, its independence assumption enables quick and effective probability estimation. Because of this, Naive Bayes is especially good at problems like text categorization, where the existence of individual words may be considered separately.

On the test set, Naive Bayes obtained a decent accuracy score of about 79% in the adult income categorization issue. This is explained by its minimal computing cost, simplicity, and efficient handling of categorical characteristics. The effects of varying feature sizes and magnitudes were lessened by using resilient scaling for numerical variables and one-hot encoding for categorical variables. Naive Bayes does have several drawbacks, though. For many real-world datasets, the feature independence assumption could not hold true, which could result in less than ideal performance. There could be feature dependencies in the adult income dataset that Naive Bayes is unable to sufficiently identify. Performance issues with the technique might also arise from its sensitivity to outliers and difficulty with skewed or unbalanced datasets.

Investigating more sophisticated feature engineering strategies to capture feature dependencies is one way to improve the accuracy of the model. More informative features that more accurately

capture the underlying relationships in the data may be produced using methods like feature interactions, polynomial features, and dimensionality reduction. For this dataset, experimenting with other Naive Bayes algorithm versions, such Multinomial or Bernoulli Naive Bayes, could also produce better results. It would be beneficial to compare the execution durations and relative performances of Naive Bayes with other supervised learning algorithms, such as decision trees, random forests, or gradient boosting machines. Although Naive Bayes is straightforward and effective, more intricate models might be able to more accurately represent nonlinear relationships and interactions in the data. To reach ideal performance, these models may need more tuning and come with greater computational expenses. As a result, the particulars of the issue, such as the balance between interpretability, accuracy, and processing capacity, determine which approach is best.

Decision-Tree Classifier

(a) Predict whether income exceeds $50K/yr based on census data. Also known as "Census Income" dataset.

(b) I chose the Decision Tree classifier for its simplicity, interpretability, and ability to handle complex relationships in the data. It's suitable for smaller datasets and offers insights into feature importance. Additionally, decision trees can serve as a basis for more advanced ensemble methods. Overall, its flexibility and transparency made it a suitable choice for the task at hand.

(c)

```
Classification accuracy : 0.8444
Classification error : 0.1556
Precision : 0.9475
Recall or Sensitivity : 0.8612
True Positive Rate : 0.8612
False Positive Rate : 0.2401
Cross-validation scores:[0.84385965 0.8495614  0.83940325 0.85168934
0.83282141 0.84071961
 0.84379114 0.85212813 0.83457657 0.84247477]
Average cross-validation score: 0.8431
```

(d) Decision tree classifiers offer several strengths, including simplicity, interpretability, and the ability to handle both numerical and categorical data without extensive preprocessing. They are

adept at capturing complex relationships and interactions in the data, making them suitable for both classification and regression tasks. Additionally, decision trees can handle missing values and are robust to outliers, reducing the need for data preprocessing.

However, decision trees are prone to overfitting, especially when the tree grows too deep or the dataset is small. This can lead to poor generalization performance on unseen data. Another limitation is their instability, as small variations in the data can result in significantly different trees. Additionally, decision trees may struggle with capturing linear relationships in the data and may not perform well on datasets with high dimensionality.

To improve the accuracy of decision tree classifiers, one approach is to tune hyperparameters such as the maximum tree depth or minimum samples per leaf through techniques like cross-validation. Another strategy is to employ ensemble methods like random forests or gradient boosting, which combine multiple decision trees to produce a more robust and accurate model. Additionally, feature selection techniques can help identify the most relevant variables, reducing the risk of overfitting and improving model performance.

In summary, decision tree classifiers offer a balance of simplicity and interpretability but may suffer from overfitting and instability. By carefully tuning hyperparameters, employing ensemble methods, and selecting relevant features, the accuracy of decision tree classifiers can be enhanced, making them effective tools for a variety of machine learning tasks.

K-means Clustering

(a) The dataset contains social media status updates with variables like type, publication date, and engagement metrics such as reactions, comments, and shares. Each update is identified by a status ID. The model predicts engagement levels based on these metrics, aiding content creators in optimizing their strategies for better audience engagement.

(b) K-means clustering was selected for this analysis to group similar social media status updates based on engagement metrics. This choice was made due to its effectiveness in unsupervised learning tasks and its ability to reveal patterns and segments within datasets. By applying K-means clustering, hidden structures in the data, such as varying engagement levels among status

updates, can be uncovered. This provides valuable insights for content creators to tailor their strategies and enhance audience engagement and reach.

(c)

```
Metrics for each fold:
Fold 1: Accuracy=0.68, Precision=0.68, Recall=0.68, F1-score=0.68
Fold 2: Accuracy=0.73, Precision=0.73, Recall=0.73, F1-score=0.73
Fold 3: Accuracy=0.00, Precision=0.00, Recall=0.00, F1-score=0.00
Fold 4: Accuracy=0.03, Precision=0.03, Recall=0.03, F1-score=0.03
Fold 5: Accuracy=0.05, Precision=0.05, Recall=0.05, F1-score=0.05
Fold 6: Accuracy=0.02, Precision=0.02, Recall=0.02, F1-score=0.02
Fold 7: Accuracy=0.01, Precision=0.01, Recall=0.01, F1-score=0.01
Fold 8: Accuracy=0.00, Precision=0.00, Recall=0.00, F1-score=0.00
Fold 9: Accuracy=0.01, Precision=0.01, Recall=0.01, F1-score=0.01
Fold 10: Accuracy=0.02, Precision=0.02, Recall=0.02, F1-score=0.02

Mean Accuracy: 0.16, Mean Precision: 0.16, Mean Recall: 0.16, Mean F1-
score: 0.16
Result: 4340 out of 7050 samples were correctly labeled.
Accuracy score: 0.62
Result: 63 out of 7050 samples were correctly labeled.
Accuracy score: 0.01
Result: 138 out of 7050 samples were correctly labeled.
Accuracy score: 0.02
Result: 4340 out of 7050 samples were correctly labeled.
Accuracy score: 0.62
```

(d) The utilized clustering methods, namely K-Means and Hierarchical Clustering, exhibit both strengths and limitations in their application to the dataset. One notable strength lies in the simplicity and ease of implementation of these techniques, making them suitable for exploratory data analysis and identifying patterns within the data. Additionally, both methods are scalable to large datasets, allowing for efficient analysis of substantial amounts of information. However, despite these advantages, there are certain limitations that may impact their performance.

One limitation is their sensitivity to the initial selection of cluster centroids, which can lead to suboptimal clustering results. In the case of K-Means, the need to specify the number of clusters beforehand can be challenging, and selecting an inappropriate value may result in poor clustering. Similarly, Hierarchical Clustering's computational complexity increases with the size

of the dataset, potentially leading to longer execution times and memory constraints for large datasets.

To improve the accuracy of the models, one approach would be to conduct a thorough exploratory data analysis to identify relevant features and preprocess the data accordingly. Feature engineering techniques such as dimensionality reduction or feature scaling could help enhance the quality of input data and improve clustering performance. Additionally, experimenting with different distance metrics and linkage methods in Hierarchical Clustering may yield better results by capturing the underlying structure of the data more effectively. Comparing the performance of the two models, K-Means generally demonstrates faster execution times due to its simplicity and scalability. However, its performance heavily depends on the initial choice of centroids and may struggle with non-linearly separable data. On the other hand, Hierarchical Clustering can capture complex relationships between data points but may suffer from increased computational overhead, particularly with larger datasets. By considering the trade-offs between performance and computational complexity, practitioners can choose the most suitable clustering method based on the specific characteristics of the dataset and the analysis objectives.

Principal Component Analysis

(a) The dataset comprises measurements of sepal length, sepal width, petal length, and petal width for iris flowers. The model's objective is to predict the species of iris flowers—Setosa, Versicolour, or Virginica—based on these measurements.

(b) I utilized Decision Tree Classifier and Principal Component Analysis (PCA) techniques. Decision Tree Classifier is a popular algorithm for classification tasks like predicting the species of iris flowers based on their measurements. PCA is used to reduce the dimensionality of the dataset while preserving most of its variance, which helps visualize the data and potentially improve the performance of the classifier.

(c)

```
Accuracy: 0.88889
Accuracy (10-fold CV): 0.93333
Precision (10-fold CV): 0.93505
Recall (10-fold CV): 0.93333
F1-score (10-fold CV): 0.93323
1 component: 92.46% of initial variance
0.361 x sepal length (cm) + -0.085 x sepal width (cm) + 0.857 x petal
length (cm) + 0.358 x petal width (cm)
2 component: 5.31% of initial variance
0.657 x sepal length (cm) + 0.730 x sepal width (cm) + -0.173 x petal
length (cm) + -0.075 x petal width (cm)
```

(d) The Decision Tree Classifier demonstrated robust performance, achieving a high accuracy score of 0.89 on the test set. However, it's important to acknowledge the limitations of this model. Decision trees tend to overfit the training data, resulting in reduced generalization performance on unseen data. This could lead to poor performance on real-world data not present in the training set. Additionally, decision trees are sensitive to small variations in the data, which can cause instability in the model's predictions.

On the other hand, PCA effectively reduced the dimensionality of the dataset while preserving most of its variance. This allowed for easier visualization of the data and potentially improved the performance of the Decision Tree Classifier. However, PCA has its own limitations. It assumes linear relationships between variables, which may not hold true for all datasets. Additionally, PCA may not always capture the most important features for classification tasks, leading to information loss.

To improve the accuracy of the model, one approach would be to fine-tune the hyperparameters of the Decision Tree Classifier, such as the maximum depth of the tree or the minimum number of samples required to split a node. This could help prevent overfitting and improve generalization performance. Another approach would be to explore different classification algorithms, such as Random Forests or Gradient Boosting Machines, which are less prone to overfitting and often achieve higher accuracy on complex datasets.

In terms of performance and execution time, the Decision Tree Classifier is relatively fast and efficient, especially for small to medium-sized datasets. However, as the dataset grows larger, the computational complexity of training and evaluating decision trees increases. In contrast, PCA is also computationally efficient, particularly for high-dimensional datasets, as it reduces

the number of features while retaining most of the information. Overall, the combination of Decision Tree Classifier with PCA provides a balance between accuracy and computational efficiency for this particular dataset.