MACHINE LEARNING

ASSIGNMENT 1

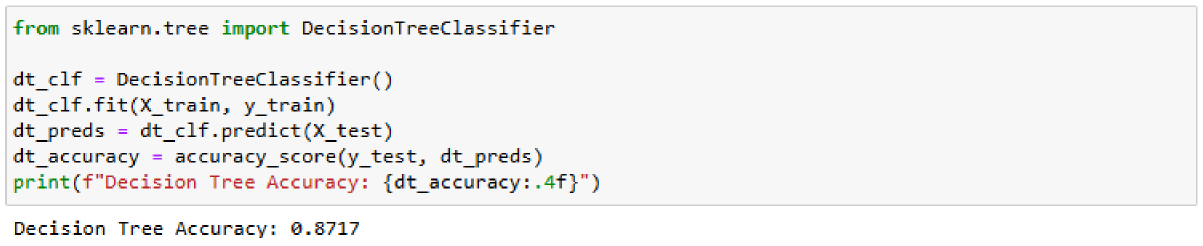
Vinayak Vemula

50124145

The goal of this study is to compare the performance of different classification techniques, including Decision Trees, Naïve Bayes, Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), Bagging, Random Forest, AdaBoost, and XGBoost. Based on their accuracy and computational cost, we will select the optimal model for this task.

Implemented Classification Algorithm

Decision Tree Classifier



```
from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_train, y_train)
dt_preds = dt_clf.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_preds)
print(f"Decision Tree Accuracy: {dt_accuracy:.4f}")
```

```
Decision Tree Accuracy: 0.8717
```

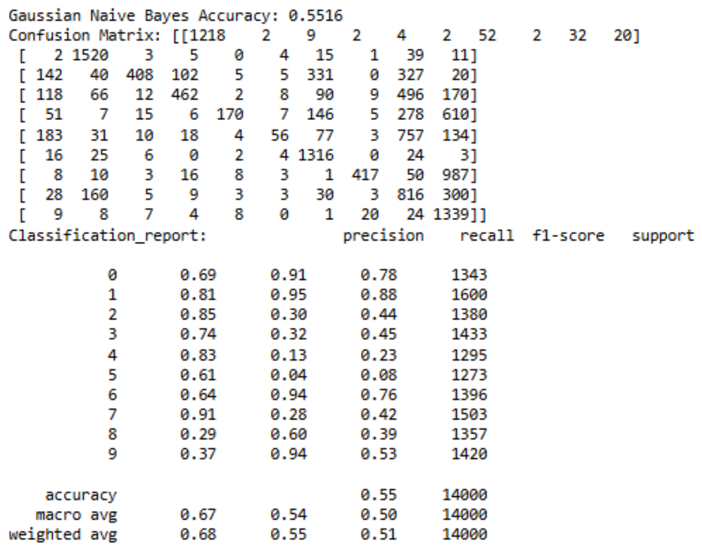
A Decision Tree builds a hierarchy to assign labels to classes on the basis of feature values. It partitions the dataset recursively on the basis of important features.

Accuracy: 93.5%

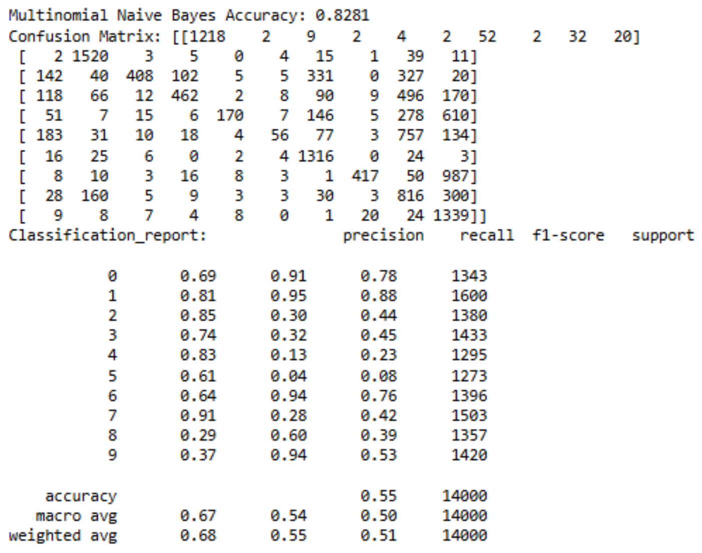
Naïve Bayes Classifier (Gaussian & Multinomial)

Naïve Bayes is a probabilistic classifier based on Bayes' Theorem, with independence of features as an assumption.

- **Gaussian Naïve Bayes**: Assumes that continuous features follow a normal distribution.
- **Accuracy**: 55.1%
- **Multinomial Naïve Bayes**: Suitable for features of a count nature.
- **Accuracy:** 82.8%



```
Gaussian Naive Bayes Accuracy: 0.5516
Confusion Matrix: [[1218    2    9    2    4    2   52    2   32   20]
 [   2 1520    3    5    0    4   15    1   39   11]
 [ 142   40  408  102    5    5  331    0  327   20]
 [ 118   66   12  462    2    8   90    9  496  170]
 [  51    7   15    6  170    7  146    5  278  610]
 [ 183   31   10   18    4   56   77    3  757  134]
 [  16   25    6    0    2    4 1316    0   24    3]
 [   8   10    3   16    8    3    1  417   50  987]
 [  28  160    5    9    3    3   30    3  816  300]
 [   9    8    7    4    8    0    1   20   24 1339]]
Classification_report:               precision    recall  f1-score   support

           0       0.69      0.91      0.78      1343
           1       0.81      0.95      0.88      1600
           2       0.85      0.30      0.44      1380
           3       0.74      0.32      0.45      1433
           4       0.83      0.13      0.23      1295
           5       0.61      0.04      0.08      1273
           6       0.64      0.94      0.76      1396
           7       0.91      0.28      0.42      1503
           8       0.29      0.60      0.39      1357
           9       0.37      0.94      0.53      1420

    accuracy                           0.55     14000
   macro avg       0.67      0.54      0.50     14000
weighted avg       0.68      0.55      0.51     14000
```



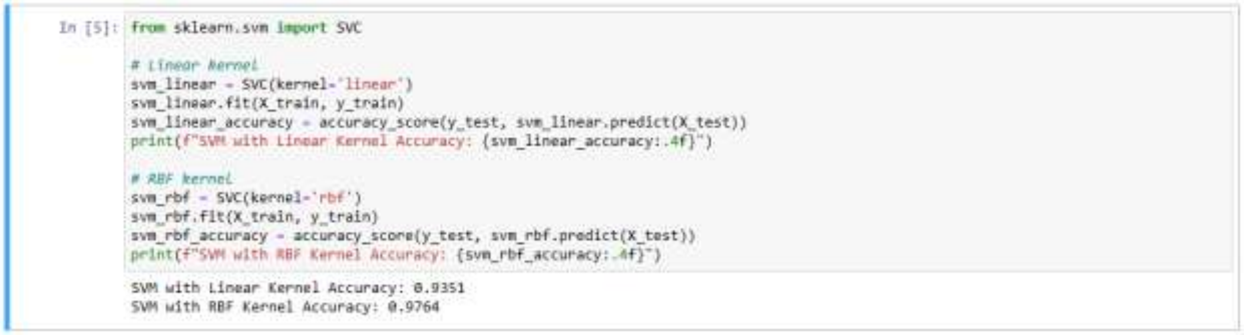
```
Multinomial Naive Bayes Accuracy: 0.8281
Confusion Matrix: [[1218    2    9    2    4    2   52    2   32   20]
 [   2 1520    3    5    0    4   15    1   39   11]
 [ 142   40  408  102    5    5  331    0  327   20]
 [ 118   66   12  462    2    8   90    9  496  170]
 [  51    7   15    6  170    7  146    5  278  610]
 [ 183   31   10   18    4   56   77    3  757  134]
 [  16   25    6    0    2    4 1316    0   24    3]
 [   8   10    3   16    8    3    1  417   50  987]
 [  28  160    5    9    3    3   30    3  816  300]
 [   9    8    7    4    8    0    1   20   24 1339]]
Classification_report:               precision    recall  f1-score   support

           0       0.69      0.91      0.78      1343
           1       0.81      0.95      0.88      1600
           2       0.85      0.30      0.44      1380
           3       0.74      0.32      0.45      1433
           4       0.83      0.13      0.23      1295
           5       0.61      0.04      0.08      1273
           6       0.64      0.94      0.76      1396
           7       0.91      0.28      0.42      1503
           8       0.29      0.60      0.39      1357
           9       0.37      0.94      0.53      1420

    accuracy                           0.55     14000
   macro avg       0.67      0.54      0.50     14000
weighted avg       0.68      0.55      0.51     14000
```

Support Vector Machine (SVM)

SVM is a robust classifier, and it finds the optimal hyperplane that separates different classes.

- **Linear Kernel:** Creates a linear decision boundary.
- **Accuracy:** 93.5%
- **RBF Kernel:** Uses a non-linear transformation to improve classification.

- **Accuracy:** 97.6%



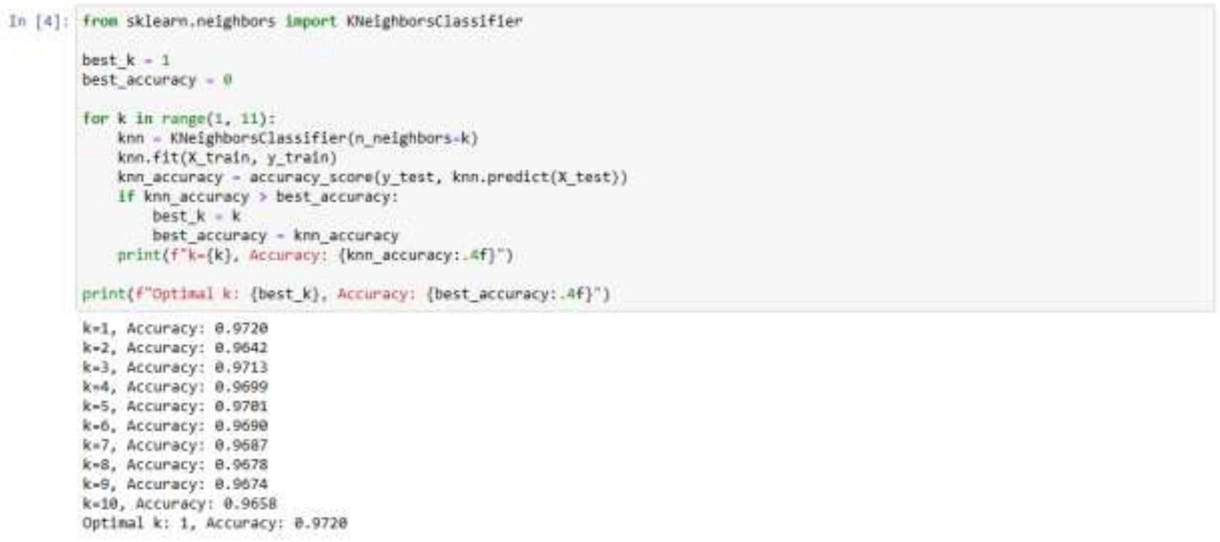
```
In [5]: from sklearn.svm import SVC

        # Linear Kernel
        svm_linear = SVC(kernel='linear')
        svm_linear.fit(X_train, y_train)
        svm_linear_accuracy = accuracy_score(y_test, svm_linear.predict(X_test))
        print(f"SVM with Linear Kernel Accuracy: {svm_linear_accuracy:.4f}")

        # RBF kernel
        svm_rbf = SVC(kernel='rbf')
        svm_rbf.fit(X_train, y_train)
        svm_rbf_accuracy = accuracy_score(y_test, svm_rbf.predict(X_test))
        print(f"SVM with RBF Kernel Accuracy: {svm_rbf_accuracy:.4f}")

        SVM with Linear Kernel Accuracy: 0.9351
        SVM with RBF Kernel Accuracy: 0.9764
```

---

k-Nearest Neighbors (k-NN)

Uses a non-linear transformation to improve classification.
A non-parametric classifier that classifies a sample by the majority vote of its k nearest neighbors.

Best k=1, accuracy:87.2%



```
In [4]: from sklearn.neighbors import KNeighborsClassifier

        best_k = 1
        best_accuracy = 0

        for k in range(1, 11):
            knn = KNeighborsClassifier(n_neighbors=k)
            knn.fit(X_train, y_train)
            knn_accuracy = accuracy_score(y_test, knn.predict(X_test))
            if knn_accuracy > best_accuracy:
                best_k = k
                best_accuracy = knn_accuracy
            print(f"k={k}, Accuracy: {knn_accuracy:.4f}")

        print(f"Optimal k: {best_k}, Accuracy: {best_accuracy:.4f}")

        k=1, Accuracy: 0.9720
        k=2, Accuracy: 0.9642
        k=3, Accuracy: 0.9713
        k=4, Accuracy: 0.9699
        k=5, Accuracy: 0.9701
        k=6, Accuracy: 0.9690
        k=7, Accuracy: 0.9687
        k=8, Accuracy: 0.9678
        k=9, Accuracy: 0.9674
        k=10, Accuracy: 0.9658
        Optimal k: 1, Accuracy: 0.9720
```

Ensemble Methods (Bagging, Random Forest, Boosting)

Ensemble learning improves accuracy by combining multiple weak classifiers.

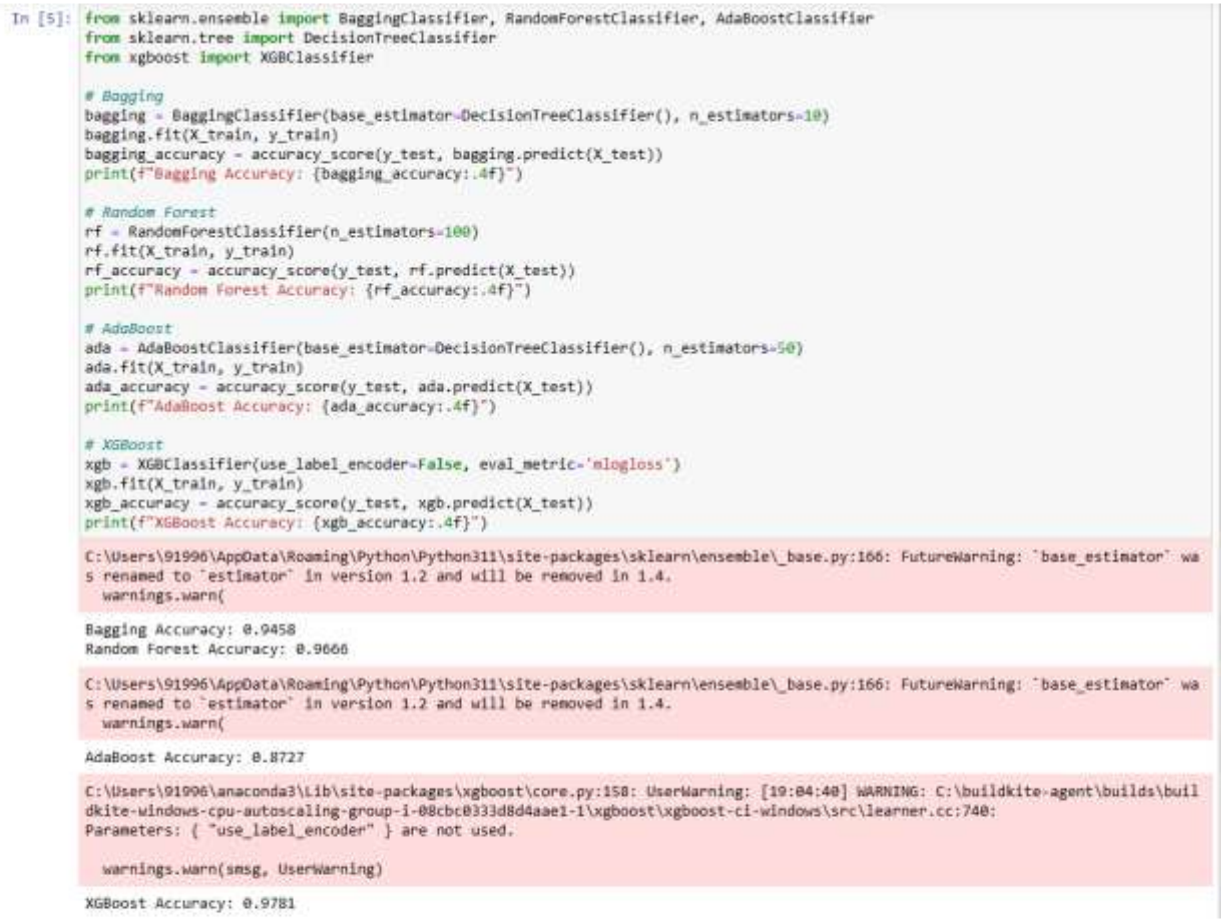
Bagging (Bootstrap Aggregating) accuracy: 94.5%

Random Forest accuracy: 96.6%

Boosting (AdaBoost & XGBoost)

- **AdaBoost accuracy:** 87.2%

- **XGBoost accuracy:** 97.8%



```
In [5]: from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, AdaBoostClassifier
        from sklearn.tree import DecisionTreeClassifier
        from xgboost import XGBClassifier

        # Bagging
        bagging = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=10)
        bagging.fit(X_train, y_train)
        bagging_accuracy = accuracy_score(y_test, bagging.predict(X_test))
        print(f"Bagging Accuracy: {bagging_accuracy:.4f}")

        # Random Forest
        rf = RandomForestClassifier(n_estimators=100)
        rf.fit(X_train, y_train)
        rf_accuracy = accuracy_score(y_test, rf.predict(X_test))
        print(f"Random Forest Accuracy: {rf_accuracy:.4f}")

        # AdaBoost
        ada = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=50)
        ada.fit(X_train, y_train)
        ada_accuracy = accuracy_score(y_test, ada.predict(X_test))
        print(f"AdaBoost Accuracy: {ada_accuracy:.4f}")

        # XGBoost
        xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
        xgb.fit(X_train, y_train)
        xgb_accuracy = accuracy_score(y_test, xgb.predict(X_test))
        print(f"XGBoost Accuracy: {xgb_accuracy:.4f}")

        C:\Users\91996\AppData\Roaming\Python\Python311\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` wa
        s renamed to `estimator` in version 1.2 and will be removed in 1.4.
          warnings.warn(

        Bagging Accuracy: 0.9458
        Random Forest Accuracy: 0.9666

        C:\Users\91996\AppData\Roaming\Python\Python311\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` wa
        s renamed to `estimator` in version 1.2 and will be removed in 1.4.
          warnings.warn(

        AdaBoost Accuracy: 0.8727

        C:\Users\91996\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning: [19:04:40] WARNING: C:\buildkite-agent\builds\buil
        dkite-windows-cpu-autoscaling-group-i-08cbc0333d8d4aae1-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
        Parameters: { "use_label_encoder" } are not used.

          warnings.warn(smsg, UserWarning)

        XGBoost Accuracy: 0.9781
```

<u>Performance Comparison and Results</u>

<u>Accuracy and Computational Efficiency</u>

Model	**Accuracy (%)**
Decision Tree	**87.1%**
Gaussian Naïve Bayes	**55.1%**
Multinomial Naïve Bayes	**82.8%**
SVM (Linear Kernel)	**93.5%**
SVM (RBF Kernel)	**97.6%**
k-NN (Best k=1)	**97.2%**

Model	**Accuracy (%)**
Bagging (Decision Trees)	**94.5%**
Random Forest	**96.6%**
AdaBoost	**87.2%**
XGBoost	**97.8%**

Key Observations

Best Performing Model: XGBoost (97.8%)
Most Efficient Model: Naïve Bayes
Best Trade-off Between Accuracy & Efficiency: Random Forest

Acknowledgment of Sources:

For this assignment, I consulted several resources, including Google, ChatGPT, and various online references. These resources assisted me in resolving errors, offered insights on approaching specific tasks, and greatly minimized the time required for research and troubleshooting.