CSCI4150U: Data Mining
Lab 03

Syed Naqvi
100590852

October 27, 2024

Preprocessing and Exploration:

We start by standardizing the features in the training data and visualizing its distribution.

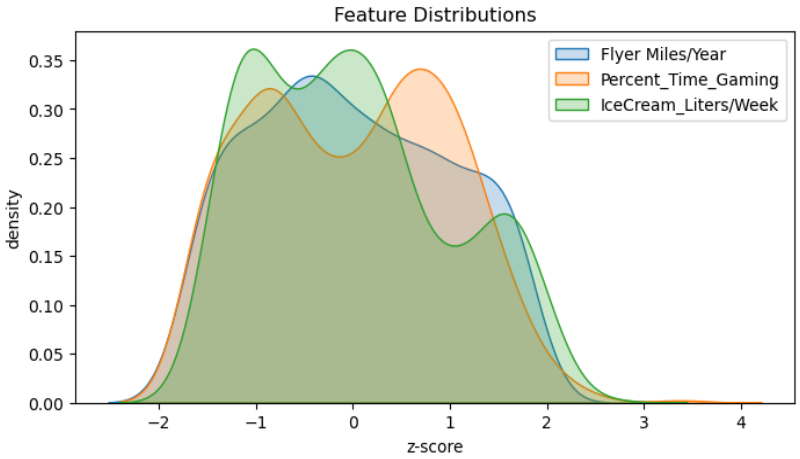

Figure 1: Standardized Distributions

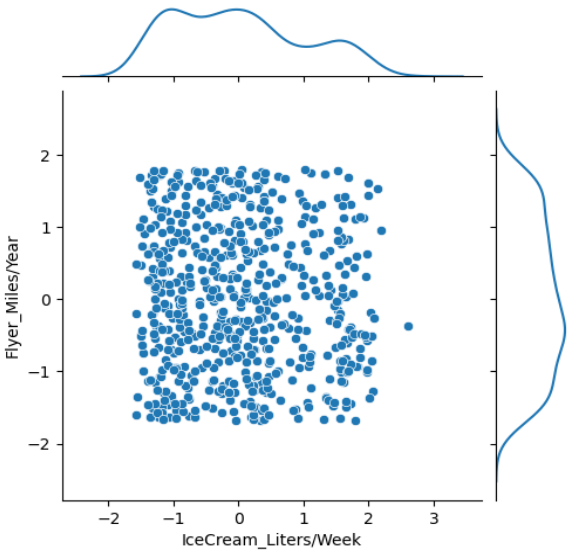

Figure 2: Flyer Miles/Year vs Ice Cream Liters/week

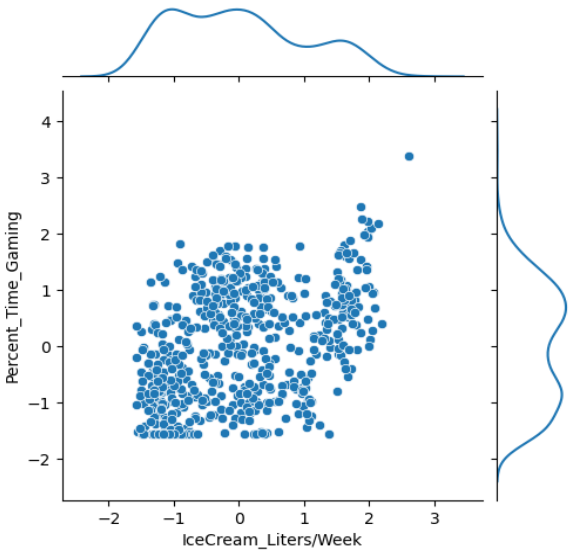

Figure 3: Percentage Time Gaming vs Ice Cream Liters/week

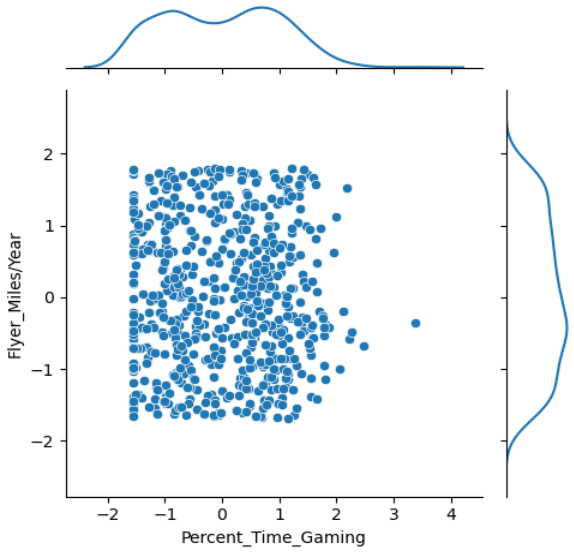

Figure 4: Flyer Miles/Year vs Percentage Time Gaming

We can also define a general cross validation helper function and store frequent performance metrics in a dictionary:

Utility Functions

```
scoring = {
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score, average='weighted'),
    'f1': make_scorer(f1_score, average='weighted'),
    'recall' : make_scorer(recall_score, average='weighted')
    }
def cross_validate_df(X, Y, scoring, model, folds):
    cv_results = pd.DataFrame(cross_validate(model, X, Y, scoring=scoring, cv=folds))
    cv_results = cv_results.iloc[:,2:]
    cv_results.index = np.arange(1, len(cv_results.iloc[:,0])+1)
    cv_results.index.name = 'folds'
    return cv_results
def holdout(X_train, X_test, Y_train, Y_test, model):
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)
    performance = pd.DataFrame({
        'accuracy': [accuracy_score(Y_test, Y_pred)],
        'precision': [precision_score(Y_test, Y_pred, average='weighted')],
        'f1-measure': [f1_score(Y_test, Y_pred, average='weighted')],
        'recall': [recall_score(Y_test, Y_pred, average='weighted')]
    })
    return performance
```

✓ 0.0s

Figure 5: Utility Functions

The following analysis will involve model selection and validation using only the **training dataset**. We will conclude with a final evaluation and comparison of the selected models using the **test dataset**.

Naive Bayes Classification (Gaussian Distribution)

Validation

Although some correlation can be observed between **Percentage Time Gaming** and **Ice Cream Liters/week**, Gaussian Naive Bayes remains a robust classification method due to roughly normal feature distributions and week/nonexistent overall feature pair correlations indicating high degree of feature independence.

```
gnb = GaussianNB()
NaiveBayesResults = cross_validate_df(dating_X, dating_Y,
                                      scoring='accuracy', model=gnb,
                                      folds=10)
display(NaiveBayesResults.mean())
```

✓ 0.0s

```
test_score    0.931667
dtype: float64
```

Figure 6: Naive Bayes Cross Validation Results

K-NN Classification

Model Selection

The training data is shuffled for each of 100 repetitions during which we iterate through different k-values for k $\in [1, 30]$, and calculate the corresponding test accuracy using 10-fold cross validation. We maintain a running total of the test accuracy for each model and then divide all accuracies by the number of repetitions (100) to get the mean accuracies. The final plot shows that $k = 18$ has the highest accuracy, and so this is the model we select.

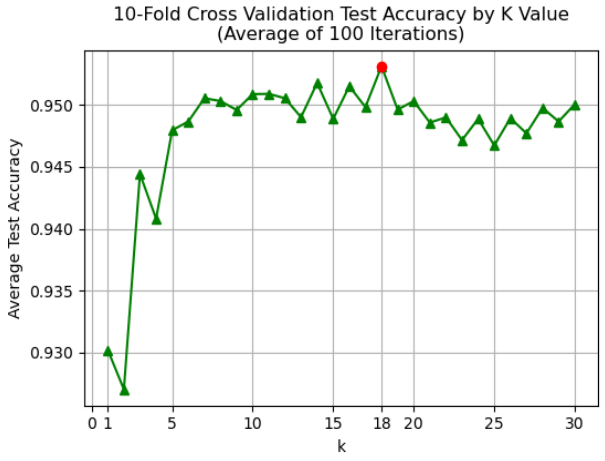

Figure 7: Best K Hyperparameter Selection Plot

```
numNeighbors = np.arange(1,31)
testAcc = np.zeros(len(numNeighbors))
iterations = 100
for iteration in range(iterations):
    dating_X, dating_Y = shuffle(dating_X, dating_Y)
    for k in numNeighbors:
        clf = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)
        acc = cross_validate_df(model=clf, X=dating_X,
                                Y=dating_Y, scoring='accuracy', folds=10)
        testAcc[k-1] += acc.mean().iloc[0]
    if (iteration%10 == 0):
        print(f"iteration {iteration} completed.")
testAcc /= iterations
```

✓ 3m 27.5s

Figure 8: Best K Hyperparameter Selection Code

Decision Tree Classification

Model Selection

Using a similar approach to K-NN classification, we begin by shuffling the training data for each of the 100 repetitions. We then iterate over a range of maximum tree depths, creating decision trees using both entropy and Gini impurity measures. For each tree, 10-fold cross-validation is applied to obtain accuracy scores, which are accumulated across repetitions for each depth and impurity measure. After completing all repetitions, the total accuracy scores are averaged by dividing by the number of repetitions, yielding the mean accuracy for each depth. Finally, we plot the average accuracies by depth for both impurity measures, allowing us to identify the optimal parameters for the decision tree model.

```
maxDepths = [2,3,4,5,6,7,8,9,10,15,20,25,30,35,40,45,50]
testAcc_GINI = np.zeros(shape=(len(maxDepths)))
testAcc_ENTROPY = np.zeros(shape=(len(maxDepths)))
repititions = 100
for rep in range(repititions):
    dating_X, dating_Y, shuffle(dating_X, dating_Y)
    for i,d in enumerate(maxDepths):
        clf_GINI = DecisionTreeClassifier(criterion='gini', max_depth=d)
        clf_ENTROPY = DecisionTreeClassifier(criterion='entropy', max_depth=d)
        gini_acc = cross_validate_df(model=clf_GINI, scoring='accuracy', X=dating_X, Y=dating_Y, folds=10)
        entropy_acc = cross_validate_df(model=clf_ENTROPY, scoring='accuracy', X=dating_X, Y=dating_Y, folds=10)
        testAcc_GINI[i] += gini_acc.mean().iloc[0]
        testAcc_ENTROPY[i] += entropy_acc.mean().iloc[0]
    if(rep%10 == 0):
        print(f"repitition {rep} compete")
testAcc_ENTROPY /= repititions
testAcc_GINI /= repititions
✓ 2m 9.2s
```

Figure 9: Decision Tree Model Selection Code

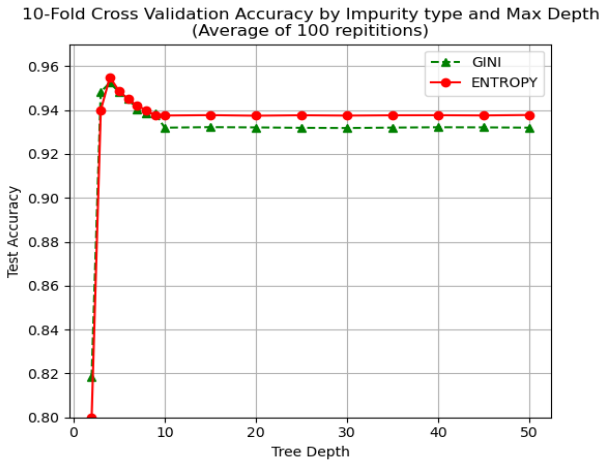

Figure 10: Accuracy by Impurity Measure and Depth

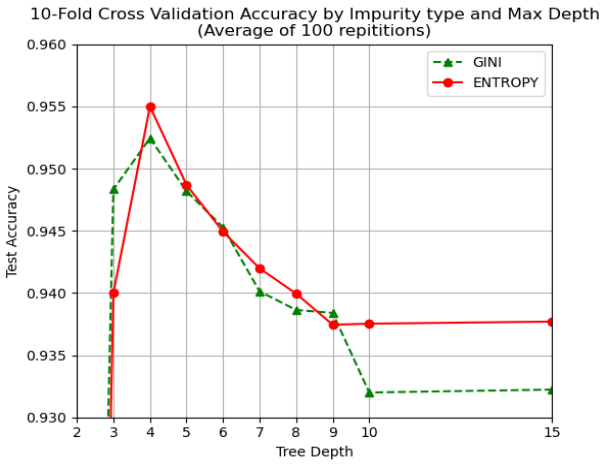

Figure 11: Accuracy by Impurity Measure and Depth (Zoomed-In)

We can see that a max depth of 4 and the entropy impurity measure yields the best performing decision tree model.

Test Set Validation

Validation

After having selected and trained the best models on the training dataset, we can now make classification predictions on the unseen instances from the test dataset.

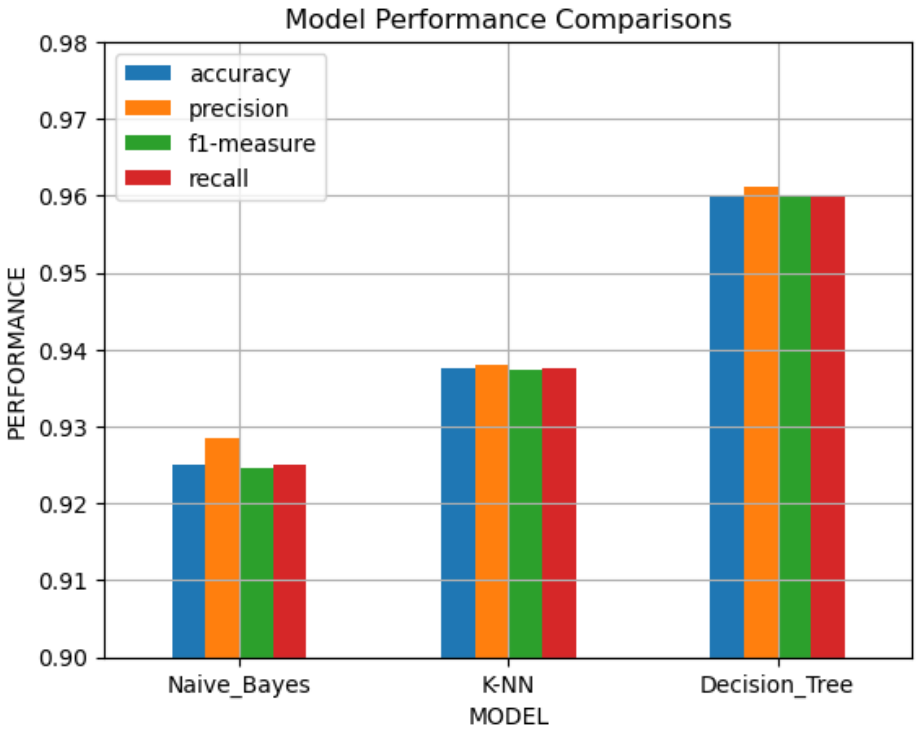

Figure 12: Model Comparison

In conclusion, a decision tree using entropy impurity and a maximum tree depth of 4 is the best model for predicting the degree to which the customer will like a given person based on the provided features.