CSCI4150U: Data Mining
Lab 03

Syed Naqvi
100590852

October 23, 2024

Preprocessing and Exploration:

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

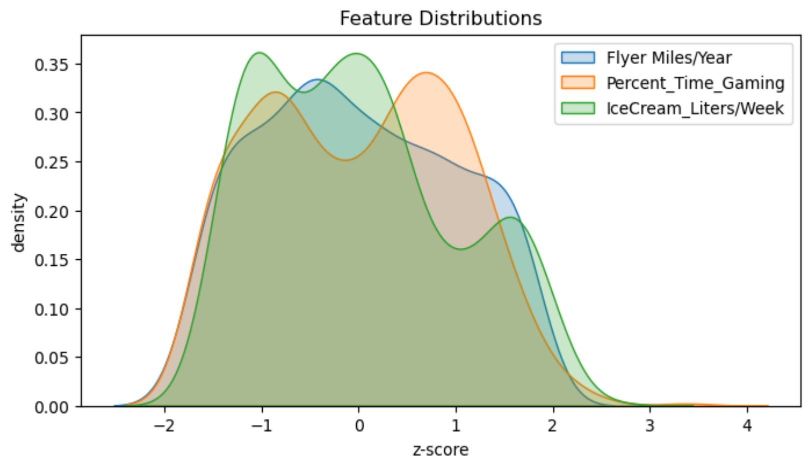


Figure 1: Standardized Distributions

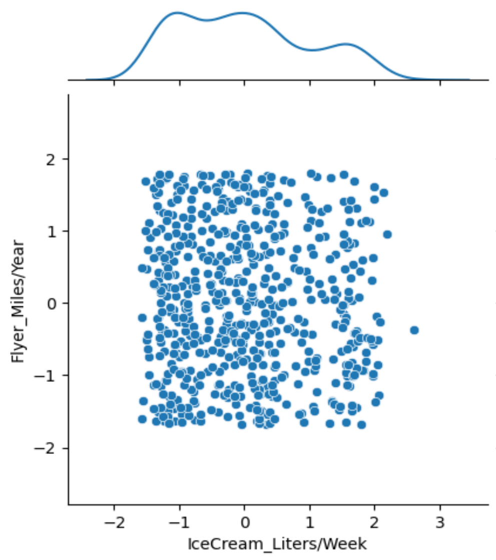


Figure 2: Flyer Miles/Year vs Ice Cream Liters/week

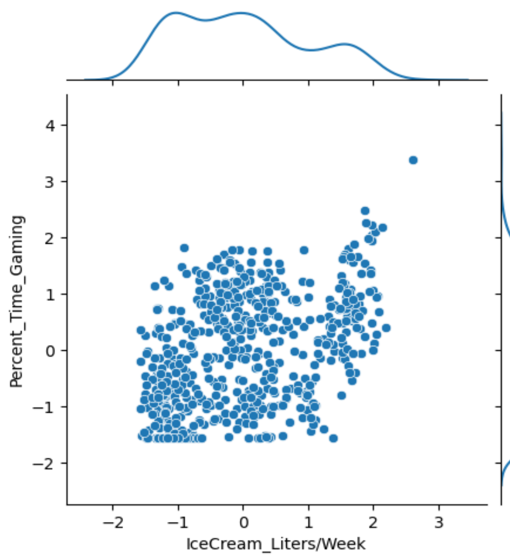


Figure 3: Percentage Time Gaming vs Ice Cream Liters/week

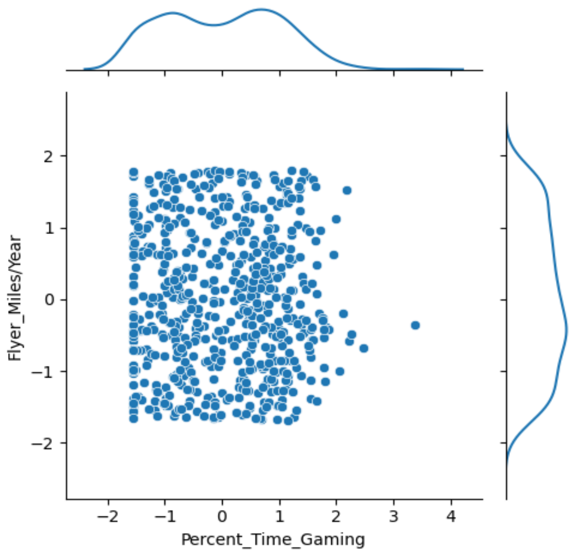
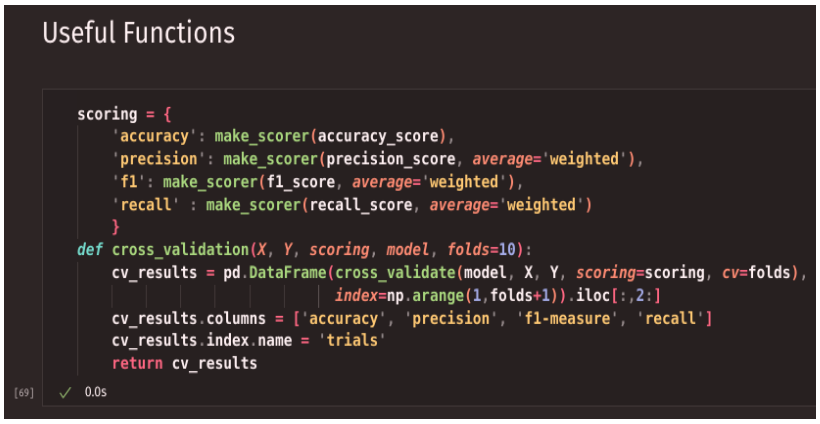


Figure 4: Flyer Miles/Year vs Percentage Time Gaming

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



Useful 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_validation(X, Y, scoring, model, folds=10):
    cv_results = pd.DataFrame(cross_validate(model, X, Y, scoring=scoring, cv=folds),
                              index=np.arange(1,folds+1)).iloc[:,2:]
    cv_results.columns = ['accuracy', 'precision', 'f1-measure', 'recall']
    cv_results.index.name = 'trials'
    return cv_results
[69]  ✓ 0.0s
```

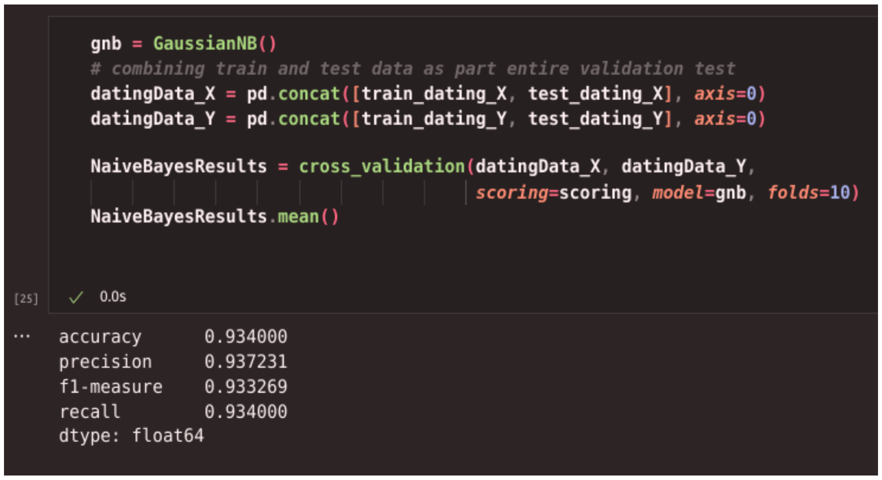
Figure 5: Utility Functions

The following analysis will involve model selection and validation using only the **training dataset**, and then conclude with a final model comparison where the selected models are first trained on the **training set**, and then evaluated using the **test set**.

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()
# combining train and test data as part entire validation test
datingData_X = pd.concat([train_dating_X, test_dating_X], axis=0)
datingData_Y = pd.concat([train_dating_Y, test_dating_Y], axis=0)

NaiveBayesResults = cross_validation(datingData_X, datingData_Y,
                                     scoring=scoring, model=gnb, folds=10)
NaiveBayesResults.mean()

[25]  ✓ 0.0s

...  accuracy      0.934000
     precision     0.937231
     f1-measure    0.933269
     recall        0.934000
     dtype: float64
```

Figure 6: Naive Bayes Cross Validation Results

K-NN Classification

Model Selection

We evaluate the accuracy for k-NN models using different **k** values. The train/test split uses the training data and is shuffled for each of 1000 epochs during which we record model test accuracy for models with values of k $\in [1, 30]$. At the end of each epoch, we increment the count of the k value associated with the highest accuracy score during that epoch. The final distribution shows that $k = 3$ has the highest accuracy an overwhelming majority of the time, and thus we choose $k = 3$ as our hyperparameter.

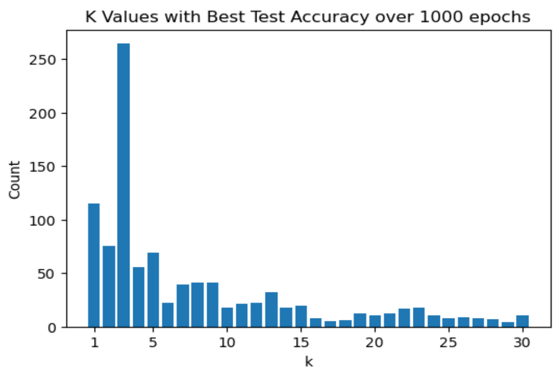
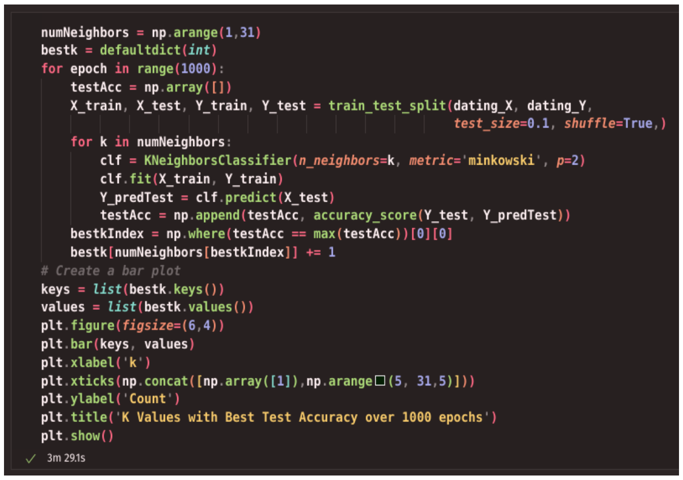


Figure 7: Best K Hyperparameter Selection Barplot

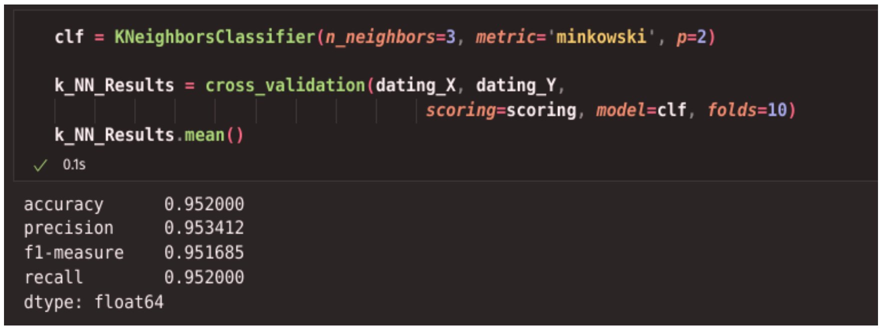


```
numNeighbors = np.arange(1,31)
bestk = defaultdict(int)
for epoch in range(1000):
    testAcc = np.array([])
    X_train, X_test, Y_train, Y_test = train_test_split(dating_X, dating_Y,
                                                        test_size=0.1, shuffle=True,)
    for k in numNeighbors:
        clf = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)
        clf.fit(X_train, Y_train)
        Y_predTest = clf.predict(X_test)
        testAcc = np.append(testAcc, accuracy_score(Y_test, Y_predTest))
    bestkIndex = np.where(testAcc == max(testAcc))[0][0]
    bestk[numNeighbors[bestkIndex]] += 1
# Create a bar plot
keys = list(bestk.keys())
values = list(bestk.values())
plt.figure(figsize=(6,4))
plt.bar(keys, values)
plt.xlabel('k')
plt.xticks(np.concat([np.array([1]),np.arange□(5, 31,5)]))
plt.ylabel('Count')
plt.title('K Values with Best Test Accuracy over 1000 epochs')
plt.show()
✓ 3m 29.1s
```

Figure 8: Best K Hyperparameter Selection Code

Validation

Having selecting our model, we perform a 10-fold cross validation and determine the associated average performance metrics.



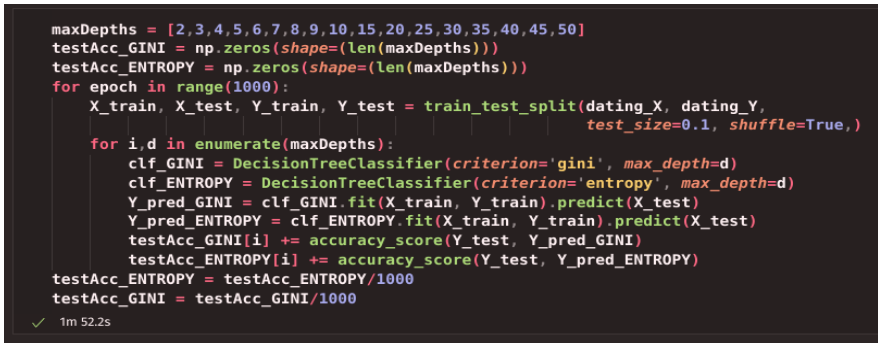
```
clf = KNeighborsClassifier(n_neighbors=3, metric='minkowski', p=2)

k_NN_Results = cross_validation(dating_X, dating_Y,
                                 scoring=scoring, model=clf, folds=10)
k_NN_Results.mean()
✓ 0.1s

accuracy      0.952000
precision     0.953412
f1-measure    0.951685
recall        0.952000
dtype: float64
```

Figure 9: k-NN Cross Validation Results (k=3)

Decision Tree Classification

Model Selection

Using a similar approach as with K-NN classification, we shuffle the data into new train/test splits for each of 1000 epochs. During each epoch, we create decision trees of different depths, for both entropy and gini impurity measures. For each impurity and depth, we maintain a running total of the test accuracies and average these values across the 1000 epochs at the end of the test.



```
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)))
for epoch in range(1000):
    X_train, X_test, Y_train, Y_test = train_test_split(dating_X, dating_Y,
                                                        test_size=0.1, shuffle=True,)
    for i,d in enumerate(maxDepths):
        clf_GINI = DecisionTreeClassifier(criterion='gini', max_depth=d)
        clf_ENTROPY = DecisionTreeClassifier(criterion='entropy', max_depth=d)
        Y_pred_GINI = clf_GINI.fit(X_train, Y_train).predict(X_test)
        Y_pred_ENTROPY = clf_ENTROPY.fit(X_train, Y_train).predict(X_test)
        testAcc_GINI[i] += accuracy_score(Y_test, Y_pred_GINI)
        testAcc_ENTROPY[i] += accuracy_score(Y_test, Y_pred_ENTROPY)
testAcc_ENTROPY = testAcc_ENTROPY/1000
testAcc_GINI = testAcc_GINI/1000
✓ 1m 52.2s
```

Figure 10: Decision Tree Model Selection Code

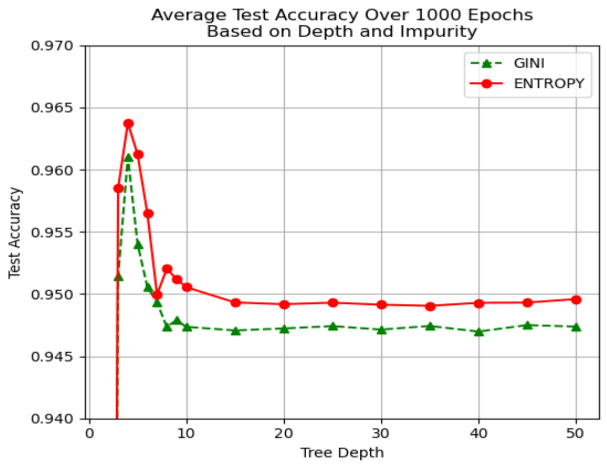


Figure 11: Accuracy by Impurity Measure and Depth

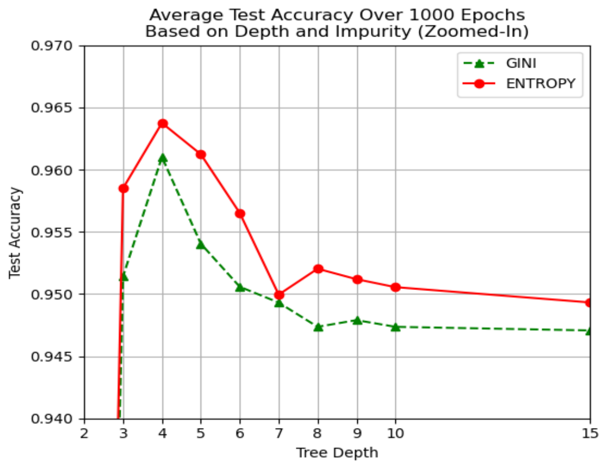
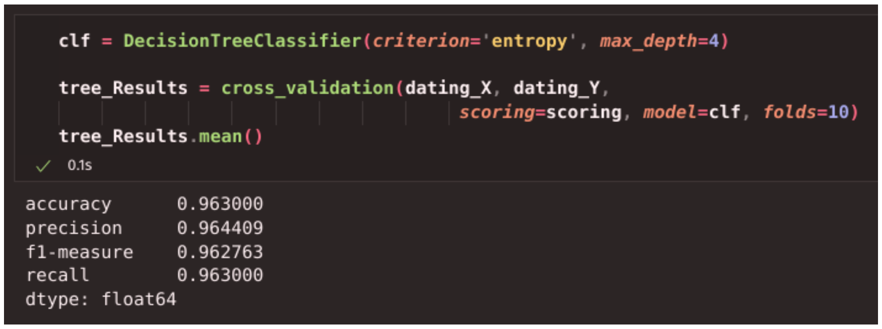


Figure 12: Accuracy by Impurity Measure and Depth (Zoomed-In)

Validation

We see that entropy impurity with a max tree depth of 4 gives the highest test accuracy score.



```
clf = DecisionTreeClassifier(criterion='entropy', max_depth=4)

tree_Results = cross_validation(dating_X, dating_Y,
                                scoring=scoring, model=clf, folds=10)
tree_Results.mean()
✓ 0.1s
```

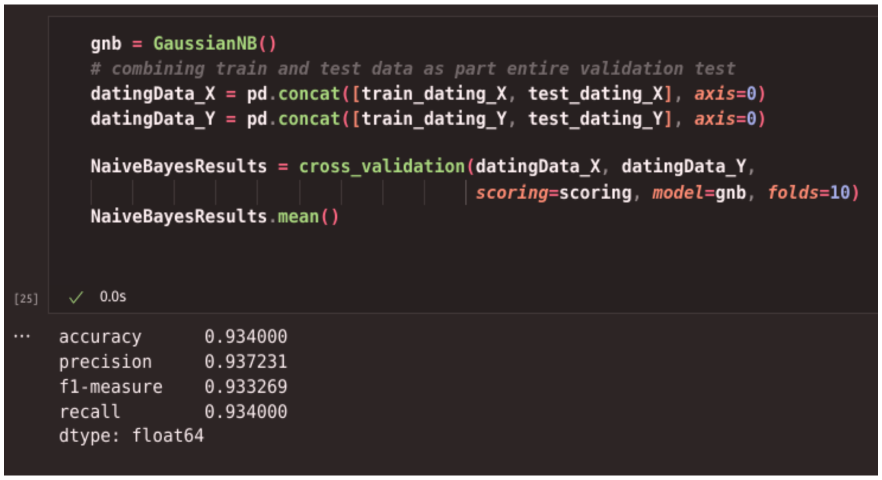
```
accuracy      0.963000
precision     0.964409
f1-measure    0.962763
recall        0.963000
dtype: float64
```

Figure 13: Decision Tree Validation (Entropy Impurity & Depth = 4)

Test Set Validation

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()
# combining train and test data as part entire validation test
datingData_X = pd.concat([train_dating_X, test_dating_X], axis=0)
datingData_Y = pd.concat([train_dating_Y, test_dating_Y], axis=0)

NaiveBayesResults = cross_validation(datingData_X, datingData_Y,
                                     scoring=scoring, model=gnb, folds=10)
NaiveBayesResults.mean()
[25] ✓ 0.0s
```

```
...  accuracy      0.934000
     precision     0.937231
     f1-measure    0.933269
     recall        0.934000
     dtype: float64
```

Figure 14: Naive Bayes Cross Validation Results