CSCI4150U: Data Mining
Lab 03

Syed Naqvi
100590852

October 22, 2024

Preprocessing:

We start by standardizing the features and visualizing their distributions.

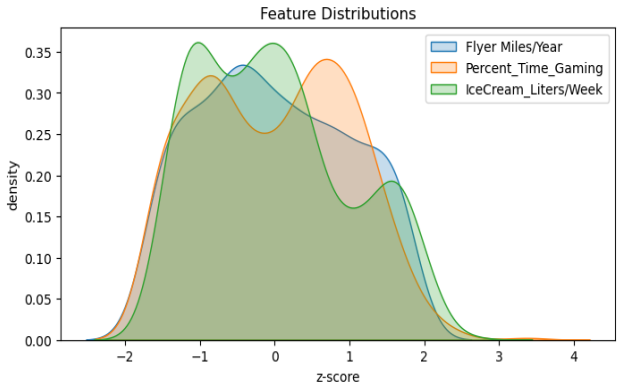

Figure 1: Standardized Distributions

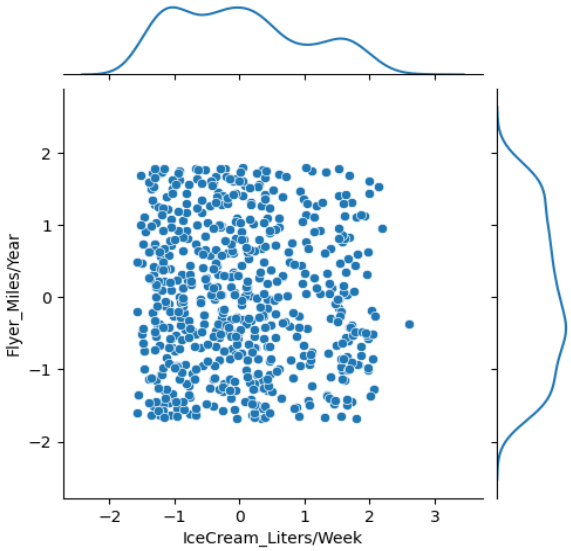

Figure 2: Flyer Miles/Year vs Ice Cream Liters/week

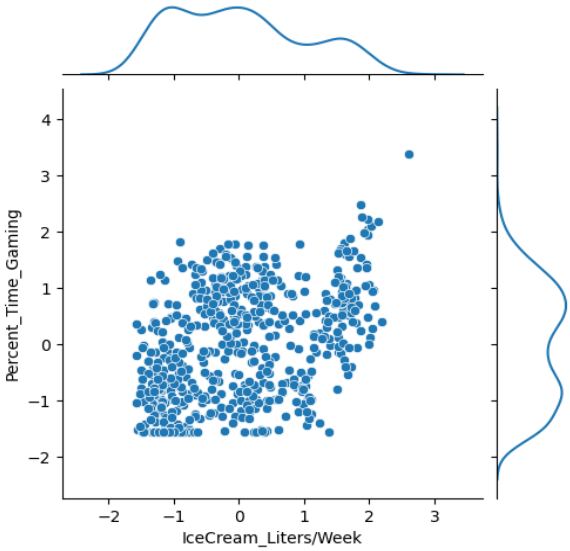

Figure 3: Percentage Time Gaming vs Ice Cream Liters/week

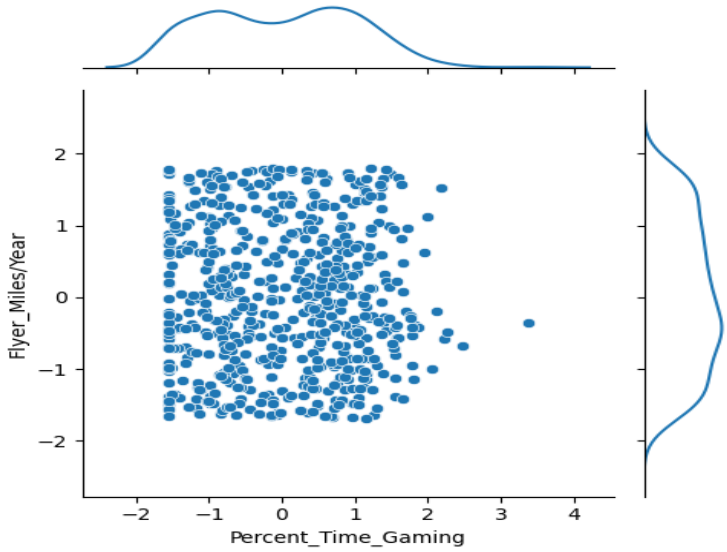

Figure 4: Flyer Miles/Year vs Percentage Time Gaming

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

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 the roughly normal feature distributions and week/nonexistent overall feature pair correlations.

```
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 test accuracy for k-NN models using different values of **k**, shuffling the dataset for each **k** value and repeating this process over 100 epochs to determine the best hyperparameter value. Our tests help us determine the best value for k should be 5.

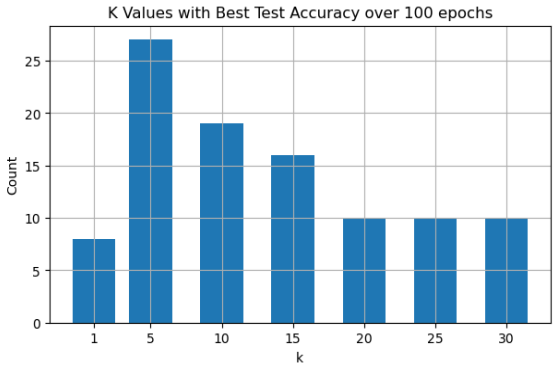

Figure 7: Best K Hyperparameter Selection Barplot

Model Selection

```
numNeighbors = [1, 5, 10, 15, 20, 25, 30]
bestk = defaultdict(int)

for epoch in range(100):
    testAcc = np.array([])
    for k in numNeighbors:
        X_train, X_test, Y_train, Y_test = train_test_split(dating_X, dating_Y,
                                                            test_size=0.1, shuffle=True,)
        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=(7,4))
plt.bar(keys, values, width=3)
plt.xlabel('k')
ticks = np.array([1])
plt.xticks(np.append(ticks, np.arange(5, 35, 5)))
plt.ylabel('Count')
plt.title('Best K Values with Best Test Accuracy over 100 epochs')
plt.grid()
plt.show()
```

```
[102]  ✓ 5.7s
```

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.

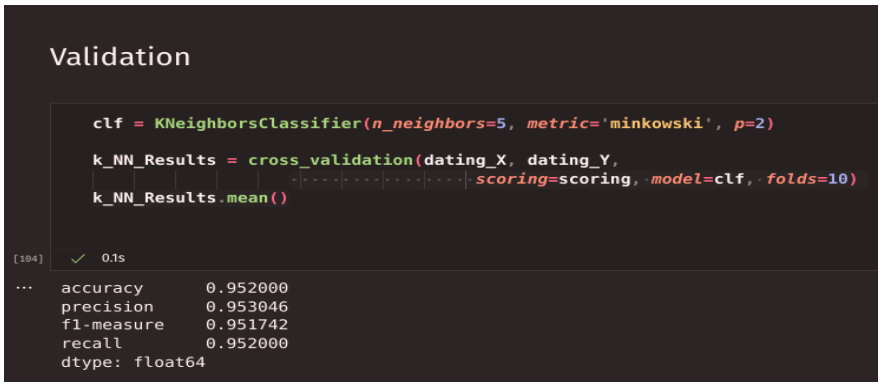

Figure 9: k-NN Cross Validation Results (k=5)

2.