Personal Information

Name: Chengzhi Fan
Student ID: 67883553
CS ID: p2k2b

Template for Question 1

1 Individual Classifiers

1.1 Result

Report the train and test result for each classifier in the given table. You should use the following hyperparameters,

1. Random Forest: no max cap on depth, and a forest size of 15 trees.
2. KNN: $k = 3$.
3. Continuous Naive Bayes: has no hyperparameters.

Model	Your Train Error (%)	Your Test Error (%)
Random Forest	0.3	13.8
KNN	7.8	13.4
Naive Bayes	19.6	17.6

Explain in one paragraph why you think a particular classifier works better on this dataset.

Answer:
KNN classifier works the best on this dataset.

The Naive Bayes runs faster on this dataset, but it's both train and test errors are considerably higher than the other two. For Random Forest, it has relatively good test error, but it runs significantly slow. Only KNN has the lowest test error and performs fast as well.

1.2 Code

Include the code you have written for each particular classifier.

1. Random Forest

   Answer:

```python

def Gini_impurity(p):
        return np.sum(p*(1-p))

class DecisionStumpGiniIndex(DecisionStumpErrorRate):

    def fit(self, X, y, split_features=None, thresholds=None):
        N, D = X.shape
        # Address the trivial case where we do not split.
        count = np.bincount(y)
        p = count / np.sum(count)  # Convert counts to probabilities.
        minGiniIndex = Gini_impurity(p)

        self.splitVariable = None
        self.splitValue = None
        self.splitSat = np.argmax(count)
        self.splitNot = None

        # Check if labels are not all equal.
        if np.unique(y).size <= 1:
            return

        if split_features is None:
            split_features = range(D)
        for d in split_features:
            column = np.unique(X[: d]) if (thresholds is None) else np.unique(thresholds[:, d])
            for value in column:
                # Count number of class labels where the feature is greater than threshold.
                y_vals = y[X[:, d] > value]
                # Avoid "RuntimeWarning: invalid value encountered".
                if len(y_vals) != 0 and len(y_vals) != N:
                    count1 = np.bincount(y_vals, minlength=len(count))
                    count0 = count - count1

                    # Compute Gini Index.
                    p1 = count1 / np.sum(count1)
                    p0 = count0 / np.sum(count0)
                    G1 = Gini_impurity(p1)
                    G0 = Gini_impurity(p0)
                    prob1 = np.sum(X[:, d] > value) / N
                    prob0 = 1 - prob1

                    giniIndex = prob0 * G0 + prob1 * G1

                    # Compare to minimum error so far.
                    if giniIndex < minGiniIndex:
                        # This is the lowest Gini Index, store this value.
                        minGiniIndex = giniIndex
                        self.splitVariable = d
                        self.splitValue = value
                        self.splitSat = np.argmax(count1)
                        self.splitNot = np.argmax(count0)

```

```python
    '''
    def create_splits(self, X):
        K = 15
        N, D = X.shape
        thresholds = np.zeros((K, D))

        for j in range(D):
            model = Kmeans(k=K)
            column = X[:, j:j+1]
            # I modified Kmeans.fit() to return its means.
            means = model.fit(column)
            thresholds[:, j:j+1] = means

        self.thresholds = thresholds

```

2. KNN

Answer:

```python

def cosine_distance(self, X1, X2):
    "Insert your code here"

    # We will first compute cosine similarity, then transform it to cosine distance.
    # cosine_similarity(a, b) = (a . b) / (|a| * |b|)

    # numerator:    a . b
    numeratorMatrix = X1 @ (X2.T)

    # denominator:  |a| * |b|
    norms1 = np.linalg.norm(X1, axis=1).reshape((X1.shape[0], 1))
    norms2 = np.linalg.norm(X2, axis=1).reshape((X2.shape[0], 1))
    denominatorMatrix = norms1 @ (norms2.T)

    # if there is any zero row in X1 or X2, set the distance between any zero row in X1 to all the rows X2
    # to zero or vice verca. So we can play a trick here by setting denominator to inf in the case above.
    denominatorMatrix = np.where(denominatorMatrix == 0, np.inf, denominatorMatrix)

    # transform cosine similarity to cosine distance
    cosineSimilarity = (numeratorMatrix / denominatorMatrix)
    cosineDistance = 1 - cosineSimilarity
    return cosineDistance

```

3. Naive Bayes

Answer:

```python

class NaiveBayes:
    def __init__(self):
        pass

    def fit(self, X, y):
        # Important: First clean dataset by removing its duplicate data. Otherwise, the duplicate data will affect
        # the expected shape of normal distribution (mean stdDev), thus increases training and testing errors.
        Xy = np.hstack((X, y.reshape((len(y), 1))))
        Xy_unique = np.unique(Xy, axis=0)
        N, D = Xy_unique.shape
        X = Xy_unique[:, 0:D-1]
        y = Xy_unique[:, D-1].astype(y.dtype)

        # Compute the probability of each class i.e p(y==c).
        N, D = X.shape
        counts = np.bincount(y)
        p_y = counts / N

        # Compute the number of class labels.
        C = len(counts)

        # Let x_dc represent data entry from X with feature d and class c.
        # Compute the mean and stdDev for each x_dc.
        meanMatrix = np.zeros((D, C), float)
        stdDevMatrix = np.zeros((D, C), float)
        for d in range(D):
            for c in range(C):
                x_dc = X[:, d][(y == c)]
                meanMatrix[d, c] = np.mean(x_dc)
                stdDevMatrix[d, c] = np.std(x_dc)

        self.c = C
        self.p_y = p_y
        self.meanMatrix = meanMatrix
        self.stdDevMatrix = stdDevMatrix

    def predict(self, X):
        C = self.c
        p_y = self.p_y
        mean = self.meanMatrix
        stdDev = self.stdDevMatrix

        N, D = X.shape
        y_pred = np.zeros(N, int)
        for i in range(N):
            # make predictions p(y) in log space.
            log_p_y = np.log(p_y)
            # make predictions p(x|y) in log space.
            x = np.array([X[i],]*C).T
            log_p_xy = np.sum((-0.5 * (((x - mean) / stdDev) ** 2) - np.log(stdDev * np.sqrt(2 * np.pi))), axis=0)
            # p(x|y) * p(y) in log space
            result = log_p_xy + log_p_y

            y_pred[i] = np.argmax(result)

        return y_pred

```

2 Stacking

2.1 Result

Report the test error and training error of the stacking classifier.

Answer:

Stacking
Training error: 0.078
Testing error: 0.119
Runtime: 0:01:22.775136

2.2 Code

Include all the code you have written for stacking classifier

Answer:

```python

class Stacking():

    def __init__(self):
        # classifiers
        self.randomForest = RandomForest(max_depth=np.inf, num_trees=15)
        self.knn = KNN(k=3)
        self.naiveBayes = NaiveBayes()

        # meta-classifier
        self.decisionTree = DecisionTree(max_depth=np.inf)  # stump_class=DecisionStumpErrorRate by default.

    def fit(self, X, y):
        self.randomForest.fit(X, y)
        self.knn.fit(X, y)
        self.naiveBayes.fit(X, y)

    def predict(self, X):
        pred0 = self.randomForest.predict(X)
        pred1 = self.knn.predict(X)
        pred2 = self.naiveBayes.predict(X)

        # Use predictions of the classifiers to form X y for meta-classifier.
        X = np.vstack((pred0, pred1, pred2)).T
        y = stats.mode(X, axis=1)[0].flatten()

        # Train meta-classifier and return predictions.
        self.decisionTree.fit(X, y)
        return self.decisionTree.predict(X)

```