**Vincent Vo (19239169) & CPSC ID: s2j1b**

# 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 | 0.002 | 0.146 |
| KNN | 0.131 | 0.204 |
| Naive Bayes | 0.250 | 0.228 |

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

Answer: Note that our data set is quite relatively large. It is a shape of (5818, 200). Based from the results above, items as though Random Forest is the best classifier for modelling this type of dataset. The reason being is because, several of our methods are non parametric methods where Model gets more complicated as you get more data. KNN can work great but it can suffer the curse of dimensionality and also suffers from high prediction and memory cost. The huge memory costs are derived from the cosine-distance calculations. On the other hand, Naive Bayes assumes that all the features must be independent from one another. With regard to real life data we've extracted, it's almost impossible that we get a set of features that are completely independent or one another. This leads us to to why Random forests is the best model for this data set because ensembling or averaging works best to leads us to better results. Averaging allows us to consider the independent errors and improves the predictions of other classifers if errors are independent, while also fortunately being able to handle large data with numerous amount of features.

## 1.2 Code

Include the code you have written for each particular classifier.

1. Random Forest

```python
class DecisionStumpGiniIndex(DecisionStumpErrorRate):
    #Question

    def fit(self, X, y, split_features=None, thresholds=None):
        # you may get RuntimeWarning: invalid value encountered Warning but that's ok.
        # Also, it may takes between 5 to 10 min to see the result.
        N, D = X.shape

        # Get an array with the number of 0's, number of 1's, etc.
        count = np.bincount(y)

        # Get the index of the largest value in count.
        # Thus, y_mode is the mode (most popular value) of y
        y_mode = np.argmax(count)

        self.splitSat = None
        self.splitNot = None
        self.splitVariable = y_mode
        self.splitValue = None

        #Array of probailities and plug into Gini Impurity
        p = count/np.sum(count)
        #This is the minimum error so far.
        giniIndexSoFar = Gini_impurity(p)

        # Loop over features looking for the best split
        for d in range(D):
            for n in range(N):
                # Choose value to equate to
                value = X[n, d]

                # Find most likely class for each split and their counts/total
                y_R = mode(y[X[:, d] > value])
                n_R = 1 # Base case for Right counts
                n_R = np.bincount(y_R)

                y_L = mode(y[X[:, d] <= value])
                n_L= 1 # Base case for Left counts
                n_L = np.bincount(y_L)

                #Total counts
                n_Total = n_R + n_L

                #Data
                data_Total = np.sum(n_Total)
                data_R = np.sum(n_R)
                data_L = np.sum(n_L)

                #probability of the right side and left side
                p_R = n_R/data_R
                p_L = n_L/data_L

                # Compute Gini Index
                giniIndex = ((data_L/data_Total)*Gini_impurity(p_L))+
((data_R/data_Total)*Gini_impurity(p_R))

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


"""**Decision Tree**"""
```

```python
def create_splits(self, X):
    N, D = X.shape
    kmeans = Kmeans(5)
    splits = np.zeros((D * kmeans.k,))
    for d in range(D):
        current = X[:,d]
        current = np.array(current, ndmin=2).T
        kmeans.fit(current)
        for j in range(kmeans.k):
            splits[j] = kmeans.means[j,]
    self.splits = splits
    return splits
```

2. KNN

```python
"""
Implementation of k-nearest neighbours classifier
"""

import numpy as np
from scipy import stats
import utils

class KNN:

    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X = X # just memorize the trianing data
        self.y = y

    def predict(self, Xtest):
        X = self.X
        y = self.y
        n = X.shape[0]
        t = Xtest.shape[0]
        k = min(self.k, n)

        # Compute cosine_distance distances between X and Xtest
        dist2 = self.cosine_distance(X, Xtest)

        # yhat is a vector of size t with integer elements
        yhat = np.ones(t, dtype=np.uint8)
        for i in range(t):
            # sort the distances to other points
            inds = np.argsort(dist2[:,i])

            # compute mode of k closest training pts
            yhat[i] = stats.mode(y[inds[:k]])[0][0]

        return yhat

    def cosine_distance(self,X1,X2):
        x1_norm = np.linalg.norm(X1)
        x2_norm = np.linalg.norm(X2)

        cosine_sim = (np.dot(X1, X2.T))/(x1_norm * x2_norm)
        return 1 - cosine_sim
```

3. Naive Bayes

```
import numpy as np

class NaiveBayes:

    def __init__(self):
        pass

    def fit(self, X, y):
        N, D = X.shape

        C = 2 #Because is there is 2 classes (Binary)
        # Compute the probability of each class i.e p(y==c)
        counts = np.bincount(y)
        p_y = counts / N

        #Dataset of mean, variance, sd
        mu = np.zeros((C,D))
        variances = np.zeros((C,D))
        sd = np.zeros((C,D))

        for d in range(D):
            for c in range(C):
                n_c = counts[c] # Number of y values that have class c
                mu[c,d] = np.mean(X[y==c,d]) #Mean of the dth coloum with class c
                variances[c,d] = np.var(X[y==c,d]) # variance ___
                sd[c,d] = np.sqrt(variances[c,d])

        self.variances = variances
        self.mu = mu
        self.sd = sd
        self.p_y = p_y

    def predict(self, X):
        N, D = X.shape
        p_y = self.p_y
        mu = self.mu
        variances = self.variances
        sd = self.sd
        # p_xy = self.p_xy
        # p_y = self.p_y

        y_pred = np.zeros(N)

        for n in range(N):
            probs = p_y.copy()
            for d in range(D):
                if X[n, d] != 0:
                    probs += ((0.5*((X[n,d]-mu[:,d])/sd[:,d])**2)+np.log(sd[:,d]*np.sqrt(2*np.pi)))
                else:
                    probs += 1-(1/2*((X[n,d] - mu[:,d])/(sd[:,d]))**2 +
np.log(np.sqrt(sd[:,d])*np.sqrt(2*np.pi)))
            probs = -1 * probs
            y_pred[n] = np.argmax(probs)

        return y_pred
```

# 2 Stacking

## 2.1 Result

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

Answer: The training error was 0.004 and the testing error was 0.156

## 2.2 Code

Include all the code you have written for stacking classifier

```python
import numpy as np
import utils
from random_forest import RandomForest
from knn import KNN
from naive_bayes import NaiveBayes
from sklearn.ensemble import RandomForestClassifier


class Stacking():

    def __init__(self):
        pass

    def fit(self, X, y):
        N,D = X.shape
        randomForestModel = RandomForest(num_trees=15, max_depth=np.inf)
        randomForestModel.fit(X,y)
        rf_predicted = randomForestModel.predict(X)

        naiveBayesModel = NaiveBayes()
        naiveBayesModel.fit(X,y)
        nb_predicted = naiveBayesModel.predict(X)

        knnModel = KNN(3)
        knn.fit(X, y)
        knn_predicted = knnModel.predict(X)

        self.rf_predicted = rf_predicted
        self.nb_predicted = nb_predicted
        self.knn_predicted = knn_predicted

        self.y = y

    def predict(self, X):
        y = self.y
        N, D = X.shape
        rf_predicted = self.rf_predicted
        nb_predicted = self.nb_predicted
        knn_predicted = self.knn_predicted

        stacked_predicted = (np.vstack((rf_predicted, nb_predicted, knn_predicted))).T

        decisionTreemodel = DecisionTree(max_depth=2)
        decisionTreemodel.fit(stacked_predicted, y)
        y_pred = decisionTreemodel.predict(stacked_predicted)

        return y_pred
```

# Template for Question 2

## 1 Team

| | |
|---|---|
| Team Members | *all team member names and csids here* |
| Kaggle Team Name | *your Kaggle team name here* |

## 2 Solution Summary

*In no more than several paragraphs summarize the approach you took to address the problem.*

## 3 Experiments

*In this section report, in less than two pages, describe in technical terms the training procedures you used, including how you went about feature selection, hyperparameter value selection, training, and so forth. Plots related to hyperparameter sweeps and other reportable aspects of your training procedure would be appreciated.*

## 4 Results

| Team Name | Kaggle Phase 1 Score | Kaggle Phase 2 Score |
|---|---|---|
| *the name of your team* | *your Phase 1 Kaggle score* | *your Phase 2 Kaggle score* |

## 5 Conclusion

*Describe what you learned and what you would have done were you to have been given more time in a few paragraphs.*

# Appendix

## Gini Index

In this example, we have a dataset with two features x and y. Each data entry belongs to either the blue class or green class.

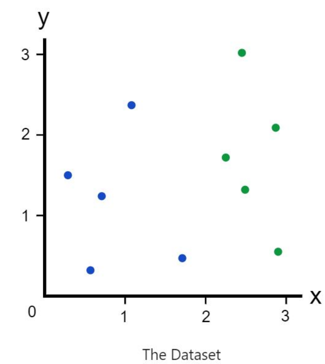


Figure 1: A given dataset with two features

lets define
$p_b$ = probability of blue class,
$p_g$ = probability of green class.

To compute the Gini impurity before splitting,
$G(p_g^{NoSplit}, p_b^{NoSplit}) = p_g^{NoSplit}(1 - p_g^{NoSplit}) + p_b^{NoSplit}(1 - p_b^{NoSplit}) = \frac{5}{10}(1 - \frac{5}{10})$,
$G(p_g^{NoSplit}, p_b^{NoSplit}) = \frac{1}{2}$.

In order to find the best split in the x-axis, we should search over the set of possible splits. One arbitrary choice is shown in figure 2.

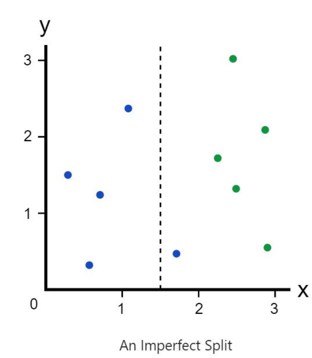


Figure 2: split feature x, where x = 1.5.

let's compute the Gini impurity for the right side.
$G(p_g^r, p_b^r) = p_g^r(1 - p_g^r) + p_b^r(1 - p_b^r)$, where $p_g^r = \frac{5}{6}$ , $p_b^r = \frac{1}{6}$,

$G(p^r_g, p^r_b) = \frac{10}{36}$

And for the left side,

$G(p^l_g, p^l_b) = p^l_g(1 - p^l_g) + p^l_b(1 - p^l_b)$, where $p^l_g = \frac{0}{4}$ , $p^l_b = \frac{4}{4}$.
$G(p^l_g, p^l_b) = 0$.

In the next step, we compute the Gini index for the current split, as follosw

$Gini\ Index = \frac{N_l}{N_t} * G(p^l_g, p^l_b) + \frac{N_r}{N_t} * G(p^r_g, p^r_b) = \frac{4}{10} * 0 + \frac{6}{10} * \frac{10}{36} = \frac{1}{6}$.

Another possible split is demonstrated in the figure 3.

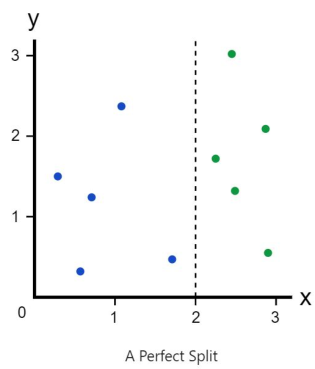


Figure 3: split feature x, where x = 2.

To compute the Gini impurity for the right side, we have

$G(p^r_g, p^r_b) = p^r_g(1 - p^r_g) + p^r_b(1 - p^r_b)$, where $p^r_g = \frac{5}{5}$ , $p^r_b = \frac{0}{5}$
$G(p^r_g, p^r_b) = 0$.

And the Gini impurity for the left side is

$G(p^l_g, p^l_b) = p^l_g(1 - p^l_g) + p^l_b(1 - p^l_b)$, where $p^l_g = \frac{0}{5}$ , $p^l_b = \frac{5}{5}$
$G(p^l_g, p^l_b) = 0$.

In the next step, we compute the Gini index for the current split:

$Gini\ Index = \frac{N_l}{N_t} * G(p^l_g, p^l_b) + \frac{N_r}{N_t} * G(p^r_g, p^r_b) = \frac{5}{10} * 0 + \frac{5}{10} * 0 = 0$.

In the end, we find the minimum Gini index between the splits. The minimum Gini index for this example is when x = 2. Also, the minimum Gini index is less than Gini impurity of no split. Therefore, we select x = 2 as the splitting rule.