CPSC 340 Assignment 2 (see course page for due date)

Important: Submission Format [5 points]

Please make sure to follow the submission instructions posted on the course website. We will deduct marks if the submission format is incorrect, or if you're not using LaTeX and your handwriting is *at all* difficult to read – at least these 5 points, more for egregious issues. Compared to assignment 1, your name and student number are no longer necessary (though it's not a bad idea to include them just in case, especially if you're doing the assignment with a partner).

1 K-Nearest Neighbours [15 points]

In the *citiesSmall* dataset, nearby points tend to receive the same class label because they are part of the same U.S. state. For this problem, perhaps a k-nearest neighbours classifier might be a better choice than a decision tree. The file *knn.py* has implemented the training function for a k-nearest neighbour classifier (which is to just memorize the data).

Fill in the `predict` function in `knn.py` so that the model file implements the k-nearest neighbour prediction rule. You should use Euclidean distance, and may find numpy's `sort` and/or `argsort` functions useful. You can also use `utils.euclidean_dist_squared`, which computes the squared Euclidean distances between all pairs of points in two matrices.

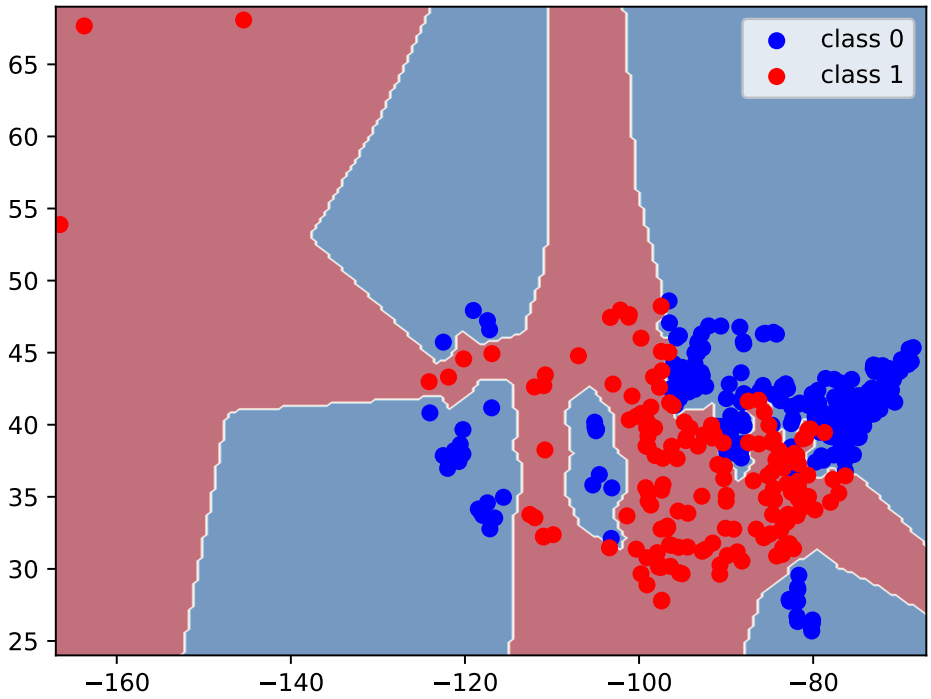
1. Write the `predict` function. Submit this code. [5 points]

```
def predict(self, X_hat):
    n,d = X_hat.shape
    y_hat = np.zeros(n)

    dist = euclidean_dist_squared(self.X, X_hat)

    for i in range(n):
        dist_x_hat_i = dist[:,i]
        order = np.argsort(dist_x_hat_i)
        select = order[:self.k]
        KNN_i = self.y[select]
        y_hat[i] = utils.mode(KNN_i)
    return y_hat

```

2. Report the training and test error obtained on the *citiesSmall* dataset for $k = 1$, $k = 3$, and $k = 10$. *Optionally*, try running a decision tree on this same train/test split; which gets better test accuracy? [4 points]

K	Training Err	Testing Err
1	0.00	0.065
3	0.028	0.066
10	0.072	0.097

3. Generate a plot with `utils.plot_classifier` on the *citiesSmall* dataset (plotting the training points) for $k = 1$, using your implementation of kNN. Include the plot here. To see if your implementation makes sense, you might want to check against the plot using `sklearn.neighbors.KNeighborsClassifier`. Remember that the assignment 1 code had examples of plotting with this function and saving the result, if that would be helpful. [2 points]



4. Why is the training error 0 for $k = 1$? [2 points]

   Answer: It's 0 because the algorithm remembers all the classifications of the training data, so it can correctly classify them.

5. Recall that we want to choose hyper-parameters so that the test error is (hopefully) minimized. How would you choose k? [2 points]

   Answer: In this case, we can see that when $K = 1$, we obtain the lowest training error and lowest error, so we should select $K = 1$.

   However, in general, it might not be the case. If we select $K = 1$, then the algorithm would predict exactly based on the training data. So if the test data doesn't have the exact distribution as the training data, we would have a large testing error. However, if we have K to be too large, it would just predict based on the mode of the sample. In practice, we may need to run several times with different K to pick the best one.

2 Picking k in kNN [15 points]

The file `data/ccdata.pkl` contains a subset of Statistics Canada's 2019 Survey of Financial Security; we're predicting whether a family regularly carries credit card debt, based on a bunch of demographic and financial information about them. (You might imagine social science researchers wanting to do something like this if they don't have debt information available – or various companies wanting to do it for less altruistic reasons.) If you're curious what the features are, you can look at the `'feat_descs'` entry in the dataset dictionary.

Anyway, now that we have our kNN algorithm working,[1] let's try choosing k on this data!

1. Remember the golden rule: we don't want to look at the test data when we're picking k. Inside the `q2()` function of `main.py`, implement 10-fold cross-validation, evaluating on the `ks` set there (1, 5, 9, ..., 29), and store the *mean* accuracy across folds for each k into a variable named `cv_accs`.

   Specifically, make sure you test on the first 10% of the data after training on the remaining 90%, then test on 10% to 20% and train on the remainder, etc – don't shuffle (so your results are consistent with ours; the data is already in random order). Implement this yourself, don't use scikit-learn or any other existing implementation of splitting. There are lots of ways you could do this, but one reasonably convenient way is to create a numpy "mask" array, maybe using `np.ones(n, dtype=bool)` for an all-`True` array of length `n`, and then setting the relevant entries to `False`. It also might be helpful to know that `~ary` flips a boolean array (`True` to `False` and vice-versa).

   Submit this code, following the general submission instructions to include your code in your results file. [5 points]

```
def q2():
    dataset = load_dataset("ccdebt.pkl")
    X = dataset["X"]
    y = dataset["y"]
    X_test = dataset["Xtest"]
    y_test = dataset["ytest"]
    n,d = X.shape

    ks = list(range(1, 30, 4))
    train = np.zeros((8,10))
    test = np.zeros((8,10))
    result = np.zeros((8,4))

    for  a in range(1,9,1):
        k_i = (a - 1) * 4 + 1
        result[a-1,0] = k_i

        #Compute testing error
        model = KNN(k = k_i)
        model.fit(X, y)
        y_hat = model.predict(X_test)
        err = np.mean(y_test != y_hat)
        result[a-1,3] = err

        #Compute errors with cross-validation
        for b in range(1,11,1):
```

[1] If you haven't finished the code for question 1, or if you'd just prefer a slightly faster implementation, you can use scikit-learn's `KNeighborsClassifier` instead. The `fit` and `predict` methods are the same; the only difference for our purposes is that `KNN(k=3)` becomes `KNeighborsClassifier(n_neighbors=3)`.

```
            low = int(0.1*(b-1)*n)
            high = int(0.1*b*n)

            #Generate Mask Array
            mask = np.ones(n, dtype=bool)
            mask[i:i+40] = False

            #Use Mask to Select Data
            X_test_i = X[~mask,:]
            y_test_i = y[~mask]
            X_i = X[mask,:]
            y_i = y[mask]

            #Compute Errors
            model = KNN(k = k_i)

            #Compute train error
            model.fit(X_i, y_i)
            y_hat = model.predict(X_i)
            err_train = np.mean(y_i != y_hat)
            train[a-1,b-1] = err_train

            # Compute test error
            y_hat_test = model.predict(X_test_i)
            err_test = np.mean(y_hat_test != y_test_i)
            test[a-1,b-1] = err_test


        result[a-1, 1] = np.mean(train[a-1,:])
        result[a-1, 2] = np.mean(test[a-1,:])
    cv_accs = result[:,2]
```

2. The point of cross-validation is to get a sense of what the test error for a particular value of k would be. Implement, similarly to the code you wrote for question 1.2, a loop to compute the test accuracy for each value of k above. Submit a plot the cross-validation and test accuracies as a function of k. Make sure your plot has axis labels and a legend. [5 points]

   Answer: Check Figure 2.2

3. Which k would cross-validation choose in this case? Which k has the best test accuracy? Would the cross-validation k do okay (qualitatively) in terms of test accuracy? [2 points]

   Answer: Cross-validation would pick $K = 17$. It also gives us a relatively low testing error, though not the lowert.

4. Separately, submit a plot of the training error as a function of k. How would the k with best training error do in terms of test error, qualitatively? [3 points]

   Answer: Check Figure 2.4

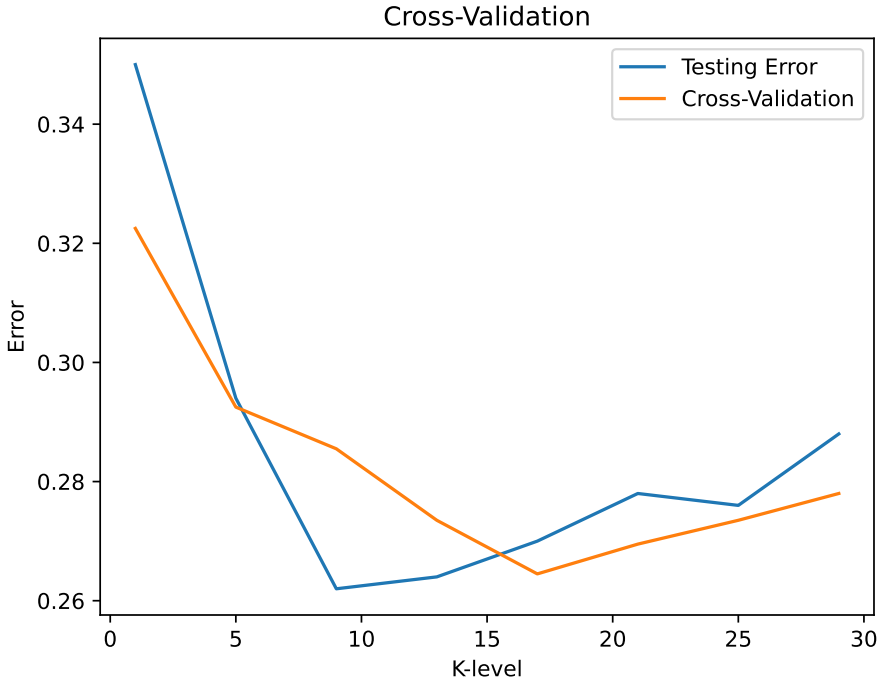


Figure 1: Figure 2.2

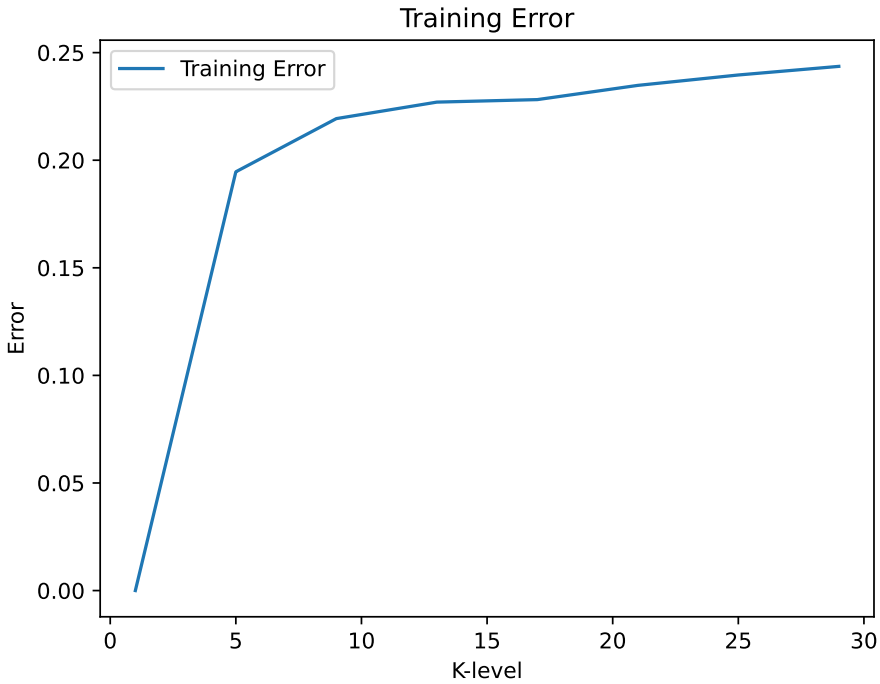


Figure 2: Figure 2.4

3 Naïve Bayes [17 points]

In this section we'll implement Naïve Bayes, a very fast classification method that is often surprisingly accurate for text data with simple representations like bag of words.

3.1 Naïve Bayes by Hand [5 points]

Consider the dataset below, which has 10 training examples and 3 features:

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{not spam} \\ \text{not spam} \\ \text{not spam} \\ \text{not spam} \end{bmatrix}.$$

The feature in the first column is <your name> (whether the e-mail contained your name), in the second column is "lottery" (whether the e-mail contained this word), and the third column is "Venmo" (whether the e-mail contained this word). Suppose you believe that a naive Bayes model would be appropriate for this dataset, and you want to classify the following test example:

$$\hat{x} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}.$$

3.1.1 Prior probabilities [1 points]

Compute the estimates of the class prior probabilities, which I also called the "baseline spam-ness" in class. (you don't need to show any work):

- Pr(spam).

  Answer: 60%

- Pr(not spam).

  Answer: 40%

3.1.2 Conditional probabilities [1 points]

Compute the estimates of the 6 conditional probabilities required by Naïve Bayes for this example (you don't need to show any work):

- Pr(<your name> = 1 | spam).

  Answer: $\frac{1}{6}$

- Pr(lottery = 1 | spam).

  Answer: $\frac{5}{6}$

- Pr(Venmo = 0 | spam).

  Answer: $\frac{2}{3}$

- Pr(<your name> = 1 | not spam).

  Answer: $\frac{1}{2}$

- Pr(lottery = 1 | not spam).

  Answer: $\frac{1}{2}$

- Pr(Venmo = 0 | not spam).

  Answer: 1

3.1.3 Prediction [2 points]

Under the naive Bayes model and your estimates of the above probabilities, what is the most likely label for the test example? **(Show your work.)**

Answer: Assuming all the X's are independent, we have

$$P(\hat{y} = 1|\hat{x}) \propto \prod_{i=1}^{3} P(\hat{x_i}|\hat{y} = 1)P(\hat{y} = 1) = \frac{1}{50} \tag{1}$$

On the other hand,

$$P(\hat{y} = 0|\hat{x}) \propto \prod_{i=1}^{3} P(\hat{x_i}|\hat{y} = 0)P(\hat{y} = 0) = 0 \tag{2}$$

Therefore, we have,

$$P(\hat{y} = 1|\hat{x}) > P(\hat{y} = 0|\hat{x}) \tag{3}$$

Then $\hat{y} = 1$

3.1.4 Simulating Laplace Smoothing with Data [1 points]

One way to think of Laplace smoothing is that you're augmenting the training set with extra counts. Consider the estimates of the conditional probabilities in this dataset when we use Laplace smoothing (with $\beta = 1$). Give a set of extra training examples where, if they were included in the training set, the "plain" estimation method (with no Laplace smoothing) would give the same estimates of the conditional probabilities as using the original dataset with Laplace smoothing. Present your answer in a reasonably easy-to-read format, for example the same format as the data set at the start of this question.

Answer: We can add an extra data set X' and y', such that

$$X' = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad y' = \begin{bmatrix} spam \\ spam \\ notspam \\ notspam \end{bmatrix}$$

3.2 Exploring Bag-of-Words [2 points]

If you run `python main.py 3.2`, it will load the following dataset:

1. `X`: A binary matrix. Each row corresponds to a newsgroup post, and each column corresponds to whether a particular word was used in the post. A value of 1 means that the word occured in the post.
2. `wordlist`: The set of words that correspond to each column.
3. `y`: A vector with values 0 through 3, with the value corresponding to the newsgroup that the post came from.

4. `groupnames`: The names of the four newsgroups.
5. `Xvalidate` and `yvalidate`: the word lists and newsgroup labels for additional newsgroup posts.

Answer the following:

1. Which word corresponds to column 73 of X? (This is index 72 in Python.)

   Answer: 'question'

2. Which words are present in training example 803 (Python index 802)?

   Answer: 'case' 'children' 'health' 'help' 'problem' 'program'

3. Which newsgroup name does training example 803 come from?

   Answer: 3

3.3 Naïve Bayes Implementation [4 points]

If you run `python main.py 3.3` it will load the newsgroups dataset, fit a basic naive Bayes model and report the validation error.

The `predict()` function of the naive Bayes classifier is already implemented. However, in `fit()` the calculation of the variable `p_xy` is incorrect (right now, it just sets all values to 1/2). Modify this function so that `p_xy` correctly computes the conditional probabilities of these values based on the frequencies in the data set. Submit your code. Report the training and validation errors that you obtain.

Answer: The training error is 0.200, and the validation error is 0.188.

The code is as following:

```
def fit(self, X, y):
    n, d = X.shape

    # Compute the number of class labels
    k = self.num_classes

    # Compute the probability of y
    counts = np.bincount(y)
    p_y = counts / n

    #Compute the conditional probabilities
    p_xy = np.zeros((d,k))
    for j in range(k):
        mask_j = np.where(y == j , True , False)
        X_j = X[mask_j]
        for i in range(d):
            p_xy[i,j] = np.mean(X_j[:,i])

    self.p_y = p_y
    self.p_xy = p_xy
```

3.4 Laplace Smoothing Implementation [4 points]

Laplace smoothing is one way to prevent failure cases of Naïve Bayes based on counting. Recall what you know from lecture to implement Laplace smoothing to your Naïve Bayes model.

- Modify the `NaiveBayesLaplace` class provided in `naive_bayes.py` and write its `fit()` method to implement Laplace smoothing. Submit this code.

```
def fit(self, X, y, beta):
    n, d = X.shape

    # Compute the number of class labels
    k = self.num_classes

    # Compute the probability of y
    counts = np.bincount(y)
    p_y = counts / n

    #Compute the conditional probabilities
    p_xy = np.zeros((d,k))
    for j in range(k):
        mask_j = np.where(y == j , True , False)
        X_j = X[mask_j]
        for i in range(d):
            p_xy[i,j] = (np.count_nonzero(X_j[:,i]) + beta)/(X_j.shape[0] + *beta)
    print(p_xy[:,0])

    self.p_y = p_y
    self.p_xy = p_xy
```

- Using the same data as the previous section, fit Naïve Bayes models with **and** without Laplace smoothing to the training data. Use $\beta = 1$ for Laplace smoothing. For each model, look at $p(x_{ij} = 1 \mid y_i = 0)$ across all j values (i.e. all features) in both models. Do you notice any difference? Explain.

  Answer: Without the Laplace smoothing, we have 12 zero entries in the conditional probabilities matrixs, which is problematice when we predict. After the Laplace smoothing, we have none zero entries.

- One more time, fit a Naïve Bayes model with Laplace smoothing using $\beta = 10000$. Look at $p(x_{ij} = 1 \mid y_i = 0)$. Do these numbers look like what you expect? Explain.

  Answer: All the numbers are close to 0.25. Since we add 10000 in the numerators and 20000 in the denominator, which are too large to our sample size(8121). So all the conditional probabilities will be driven close to $1/K$, and in this case we have $1/K = 1/4$.

3.5 Runtime of Naïve Bayes for Discrete Data [2 points]

For a given training example i, the predict function in the provided code computes the quantity

$$p(y_i \mid x_i) \propto p(y_i) \prod_{j=1}^{d} p(x_{ij} \mid y_i),$$

for each class y_i (and where the proportionality constant is not relevant). For many problems, a lot of the $p(x_{ij} \mid y_i)$ values may be very small. This can cause the above product to underflow. The standard fix for this is to compute the logarithm of this quantity and use that $\log(ab) = \log(a) + \log(b)$,

$$\log p(y_i \mid x_i) = \log p(y_i) + \sum_{j=1}^{d} \log p(x_{ij} \mid y_i) + (\text{log of the irrelevant proportionality constant}).$$

This turns the multiplications into additions and thus typically would not underflow.

Assume you have the following setup:

- The training set has n objects each with d features.
- The test set has t objects with d features.
- Each feature can have up to c discrete values (you can assume $c \leq n$).
- There are k class labels (you can assume $k \leq n$).

You can implement the training phase of a naive Bayes classifier in this setup in $O(kcd + nd)$, since you only need to do a constant amount of work for each x_{ij} value; usually $kc \ll n$ and so this is $O(nd)$. (You do not have to actually implement it in this way for the previous question, but you should think about how this could be done.) What is the cost of classifying t test examples with the model and this way of computing the predictions? It's preferable to leave your answer in terms of k and c if relevant.

Answer: For each of the t entries, we need to calculate k different conditional probabilities to compare. Within each calculation, we have d items to product. Therefore, all together, the computational cost would be O(tkd)

4 Random Forests [15 points]

The file `vowels.pkl` contains a supervised learning dataset where we are trying to predict which of the 11 "steady-state" English vowels that a speaker is trying to pronounce.

You are provided with a `RandomStump` class that differs from `DecisionStumpInfoGain` in that it only considers $\lfloor \sqrt{d} \rfloor$ randomly-chosen features.[2] You are also provided with a `RandomTree` class that is exactly the same as `DecisionTree` except that it uses `RandomStump` instead of `DecisionStump` and it takes a bootstrap sample of the data before fitting. In other words, `RandomTree` is the entity we discussed in class, which makes up a random forest.

If you run `python main.py 4` it will fit a deep `DecisionTree` using the information gain splitting criterion. You will notice that the model overfits badly.

1. Using the provided code, evaluate the `RandomTree` model of unlimited depth. Why doesn't the random tree model have a training error of 0? [2 points]

   Answer: Because in the `RandomTree` model, it's trained by the bootstrap data, but compared with the original data. Since the bootstrap data is not identical with the original data, there will be training errors.

2. For `RandomTree`, if you set the `max_depth` value to `np.inf`, why do the training functions terminate instead of making an infinite number of splitting rules? [2 points]

   Answer: Though we set the maximum depth to be infinity, but there is a certain level of depth that any depth beyond that would not change the result of the decision tree. That's where the model stops.

3. Complete the `RandomForest` class in `random_tree.py`. This class takes in hyperparameters `num_trees` and `max_depth` and fits `num_trees` random trees each with maximum depth `max_depth`. For prediction, have all trees predict and then take the mode. Submit this code. [5 points]

```
class RandomForest:
    def __init__(self, max_depth, num_trees):
        DecisionTree.__init__(
            self, max_depth=max_depth, stump_class=RandomStumpInfoGain
        )
        self.num_trees = num_trees

    def fit(self, X, y):
        #For simplicity, store 50 tree models
        self.trees = []
        for i in range(self.num_trees):
            tree = RandomTree(max_depth=self.max_depth)
            tree.fit(X, y)
            self.trees.append(tree)

    def predict(self, X):
        n = X.shape[0]
        y = np.zeros(n)

        #Generate a result matrix
        result = np.zeros((n, self.num_trees))

```

[2] The notation $\lfloor x \rfloor$ means the "floor" of x, or "x rounded down". You can compute this with `np.floor(x)` or `math.floor(x)`.

```
        #For each tree, generate a prediction vector, store it in a matrix
        for i, tree in enumerate(self.trees):
            y_i  = tree.predict( X)
            result[:,i] = y_i

        #Find the mode of the predictions
        for i in range(n):
            y[i] = utils.mode(result[i,:])

        return y
```

4. Using 50 trees, and a max depth of ∞, report the training and testing error. Compare this to what we got with a single `DecisionTree` and with a single `RandomTree`. Are the results what you expected? Discuss. [3 points]

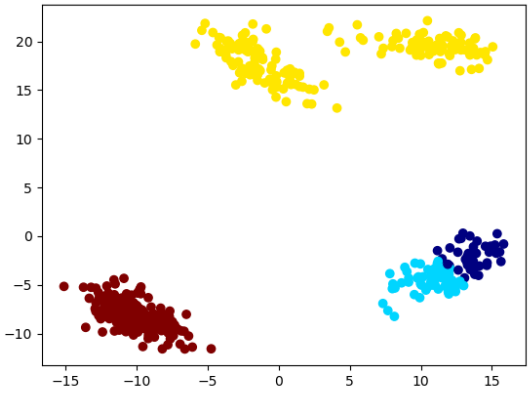
   Answer: The `RandomTree` model gives us 0 training error and 0.182 testing error. It is the result we expect since it has lower training error than a single random tree and a lower testing error than a single decision tree

5. Why does a random forest typically have a training error of 0, even though random trees typically have a training error greater than 0? [3 points]

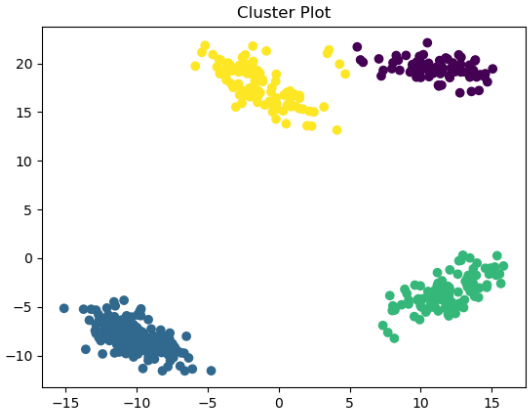
   Answer: Since even though each random tree use different bootstrap data, they are very familiar with the original data. Therefore, when we use a random forest with a lot of random trees, the mode of their perdiction will be the same as we trained from

5 Clustering [15 points]

If you run `python main.py 5`, it will load a dataset with two features and a very obvious clustering structure. It will then apply the k-means algorithm with a random initialization. The result of applying the algorithm will thus depend on the randomization, but a typical run might look like this:



(Note that the colours are arbitrary – this is the label switching issue.) But the "correct" clustering (that was used to make the data) is this:



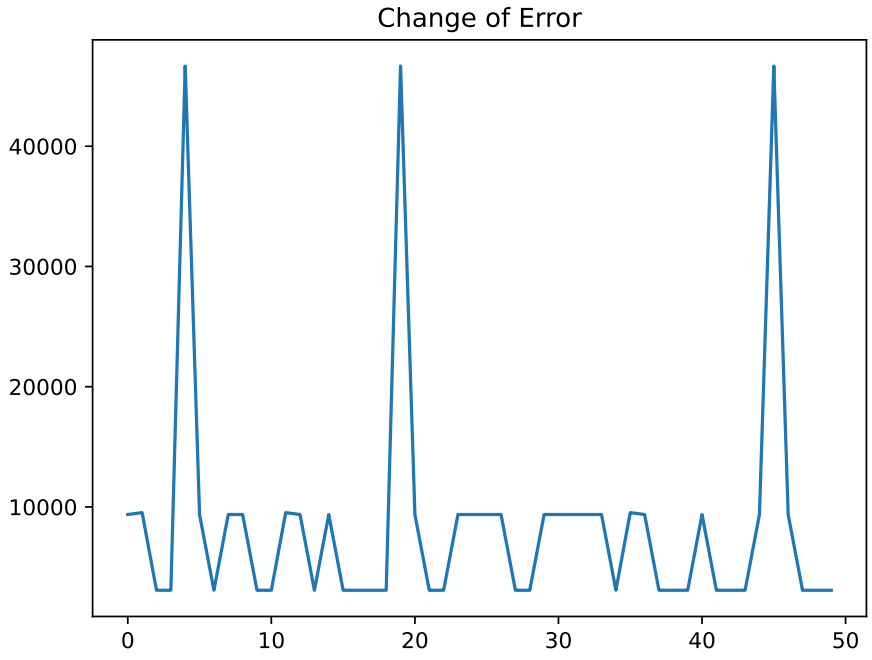
5.1 Selecting among k-means Initializations [7 points]

If you run the demo several times, it will find different clusterings. To select among clusterings for a *fixed* value of k, one strategy is to minimize the sum of squared distances between examples x_i and their means w_{y_i},

$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_2^2 = \sum_{i=1}^{n} \sum_{j=1}^{d} (x_{ij} - w_{y_i j})^2.$$

where y_i is the index of the closest mean to x_i. This is a natural criterion because the steps of k-means alternately optimize this objective function in terms of the w_c and the y_i values.

1. In the `kmeans.py` file, complete the `error()` method. `error()` takes as input the data used in fit (`X`), the indices of each examples' nearest mean (`y`), and the current value of means (`means`). It returns the



value of this above objective function. Submit this code. What trend do you observe if you print the value of this error after each iteration of the k-means algorithm? [4 points]

Answer: After each iteration, the error is lowered

```
def error(self, X, y, means):
    n = X.shape[0]
    err = np.int(0)
    for i in range(n):
        err += np.sum((X[i,:] - means[y[i],:]) ** 2)
    return err
```

2. Run k-means 50 times (with $k = 4$) and take the one with the lowest error. Report the lowest error obtained. Visualize the clustering obtained by this model, and submit your plot. [3 points]

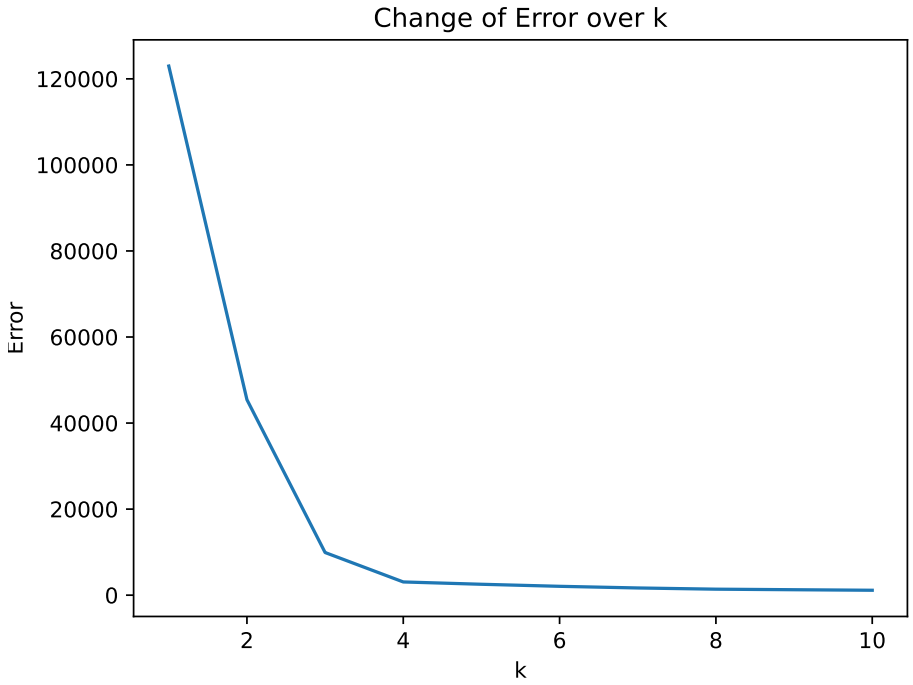
   Answer: The lowest error I got is 3071. We can see from figure 2 that the error of each time seems to be random within 3 steady states. Since each time the k-mean algorithm would update untill the categorises are steady, we could have 3 different ways of steady classification.

5.2 Selecting k in k-means [8 points]

We now turn to the task of choosing the number of clusters k.

1. Explain why we should not choose k by taking the value that minimizes the `error` value. [2 points]

   Answer: We could pick a very large k that minimize the error. For example, if we have k equal to our sample size, then we could potential have 0 error. But that doesn't give us a meaningful classification.



2. Is evaluating the `error` function on validation (or test) data a suitable approach to choosing k? [2 points]

   Answer: First of all, no matter what data we are working on, a larger k would always result in a smaller error. Secondly, this is a unsupervised learning, and we don't have a real y to compare with during the training period. Therefore, the error won't be different if we are using test data or not.

3. Hand in a plot of the minimum error found across 50 random initializations, as a function of k, taking k from 1 to 10. [2 points]

   Answer: See figure 3

4. The *elbow method* for choosing k consists of looking at the above plot and visually trying to choose the k that makes the sharpest "elbow" (the biggest change in slope). What values of k might be reasonable according to this method? Note: there is not a single correct answer here; it is somewhat open to interpretation and there is a range of reasonable answers. [2 points]

   Answer: Without calculating the real slope, the selection of $k = 3$ and $k = 4$ both seems plausible.

6 Very-Short Answer Questions [18 points]

Write a short one or two sentence answer to each of the questions below. Make sure your answer is clear and concise.

1. What is a reason that the the data may not be IID in the email spam filtering example from lecture?

   Answer: If one register on a shopping website, and then several brands send him spam email because of that, then the data of his email is not IID.

2. Why can't we (typically) use the training error to select a hyper-parameter?

   Answer: Because what we want from the algorithm it to accrately test/classify new data. However, training error can only shows how well the algorithm learns from the training data. If we select hyper-parameter by training error, we could always have a model that remembers and always reports the true output. However, it's useless. This problem can also be generalized as overfitting problem.

3. What is the effect of the training or validation set size n on the optimization bias, assuming we use a parametric model?

   Answer: As n increase, the bias would decrease. However, there would be a lowest bias that can't be eliminate by parametric model.

4. What is an advantage and a disadvantage of using a large k value in k-fold cross-validation?

   Answer: With k large, we would have more data when fitting the model. However, at the same time, we need to fit more models.

5. Recall that false positive in binary classification means $\hat{y}_i = 1$ while $\tilde{y}_i = 0$. Give an example of when increasing false positives is an acceptable risk.

   Answer: When testing for cancer or any kind of deadly disease, a false positive would be much more acceptable than a false negative.

6. Why can we ignore $p(x_i)$ when we use naive Bayes?

   Answer: Firstly, it won't affect the relative relationship between different $p(y_i = c|x_i)$. Secondly, it can sometimes be zero and meaningless.

7. For each of the three values below in a naive Bayes model, say whether it's better considered as a parameter or a hyper-parameter:

   (a) Our estimate of $p(y_i)$ for some y_i.

   Answer: Parameter. It's learned from sample.

   (b) Our estimate of $p(x_{ij} \mid y_i)$ for some x_{ij} and y_i.

   Answer: Parameter. It's learned from sample.

   (c) The value β in Laplace smoothing.

   Answer: Hyper-parameter. Since it's set previously.

8. Both supervised learning and clustering models take in an input x_i and produce a label y_i. What is the key difference between these types of models?

   Answer: In supervised learning, we will compare our result with the true y to improve the accuracy during the training. However, we don't have it in the clustering model(unsupervised learning).

9. In k-means clustering the clusters are guaranteed to be convex regions. Are the areas that are given the same label by kNN also convex?

Answer: No they are not. KNN would find a close neighborhohd and classify based on the mode. It doesn't guarantee the points on the segment connecting two elements in the same category would all be in the same one.