CPSC 340 Assignment 2 (due 2021-10-01 at 11:59pm)

We are providing solutions because supervised learning is easier than unsupervised learning, and so we think having solutions available can help you learn. However, the solution file is meant for you alone and we do not give permission to share these solution files with anyone. Both distributing solution files to other people or using solution files provided to you by other people are considered academic misconduct. Please see UBC's policy on this topic if you are not familiar with it:
http://www.calendar.ubc.ca/vancouver/index.cfm?tree=3,54,111,959
http://www.calendar.ubc.ca/vancouver/index.cfm?tree=3,54,111,960

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]

   Answer:

```python
def predict(self, X_hat):
    t, d = X_hat.shape
    y_hat = np.zeros(t, dtype=np.uint8)

    # Compute n-by-t squared distance matrix. Sometimes called "D".
    distance_matrix = euclidean_dist_squared(self.X, X_hat)

    # Iterate through rows of distance matrix
    for i in range(t):
        distances_from_x_hat_i = distance_matrix[:, i]
        nn_idxs = np.argsort(distances_from_x_hat_i)
        knn_idxs = nn_idxs[:self.k]
        y_knns = self.y[knn_idxs]
        y_hat[i] = utils.mode(y_knns)
```

```
15
16        return y_hat
```

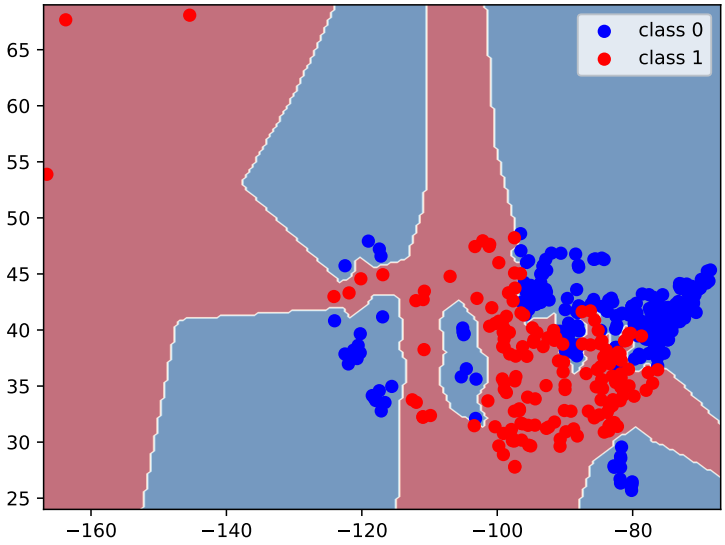
2. Report the training and test error obtained on the *citiesSmall* dataset for $k = 1$, $k = 3$, and $k = 10$. *Optionally*, try running a decision tree on this same train/test split; which gets better test accuracy? [4 points]

   Answer:

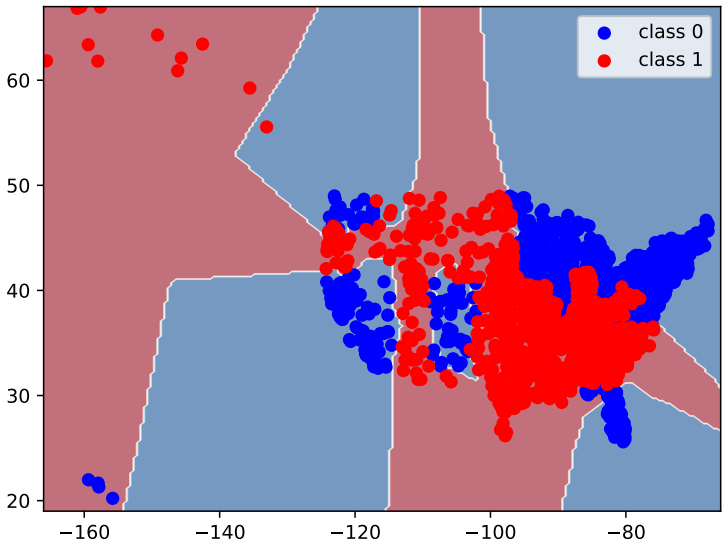
   - $k = 1$: training is 0, test is 0.065.
   - $k = 3$: training is 0.028, test is 0.066.
   - $k = 10$: training is 0072, test is 0.097.

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]

   Answer: The plot should look like this:



   If you plotted the test points instead, you'd get



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

   Answer: Every training example is 1-nearest neighbour of itself, so when you are predicting you just copy the labels from the training data. (Unless you have duplicate training examples, 1-nearest neighbour always obtains a training error of 0: this is why reporting low training errors is meaningless.)

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: The training error strictly goes down as k decreases, so you can't use the training error to choose k. Instead, you should split your data into a training and validation set, or use cross-validation – as we're just about to do in the next question!

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]

   Answer:

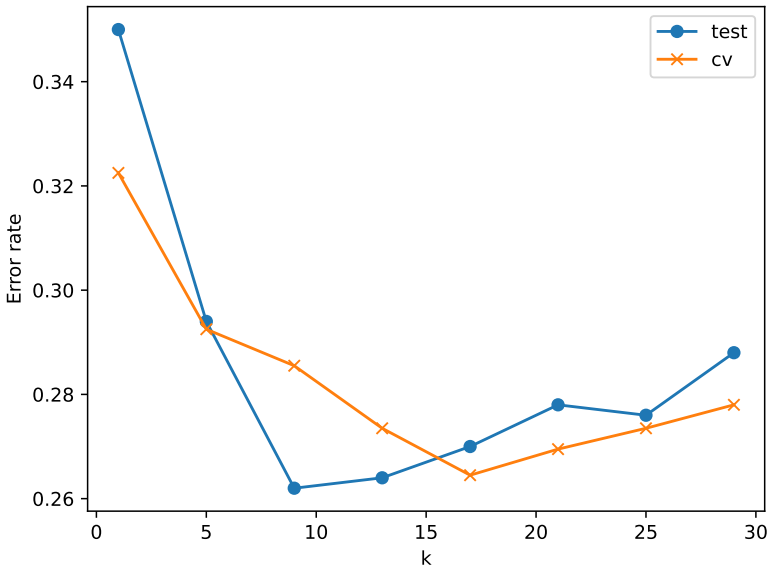
```
cv_accs = []
n = X.shape[0]
n_folds = 10
for k in ks:
    fold_accs = []
    for fold in range(n_folds):
        in_train = np.ones(n, dtype=bool)
        in_train[fold * n // n_folds:(fold + 1) * n // n_folds] = False

        model = KNN(k=k)
        model.fit(X[in_train], y[in_train])
        preds = model.predict(X[~in_train])
        acc = np.mean(preds != y[~in_train])
        fold_accs.append(acc)
    cv_accs.append(np.mean(fold_accs))
```

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: Here it is in terms of error rate; accuracies would be one minus this. (Either is fine to submit on this problem.)

[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)`.

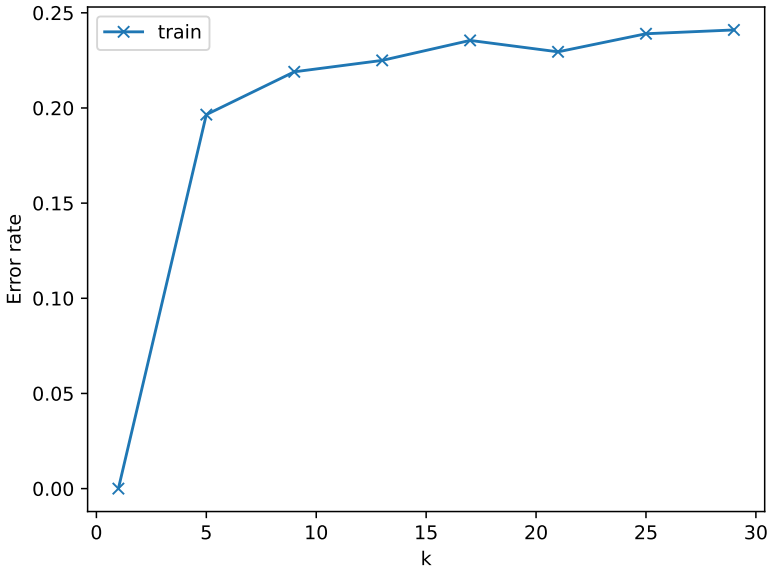


3. Which k would cross-validation choose in this case? Which k has the best test accuracy? Would the cross-validation k do okay (qualitatively) in terms of test accuracy? [2 points]

   Answer: CV would choose $k = 17$, while the optimal k of those we tried on the test set is actually $k = 9$ (which CV thought was pretty bad). The CV choice of k is not terrible, though.

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: It would (as always) pick $k = 1$, which is quite bad on this dataset.



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:

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

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: 7/10.

- Pr(not spam).

  Answer: 3/10.

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: 2/7.

- Pr(lottery = 1 | spam).

  Answer: 5/7.

- Pr(Venmo = 0 | spam).

  Answer: 3/7.

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

  Answer: 2/3.

- $\Pr(\text{lottery} = 1 \mid \text{not spam})$.

  Answer: 1/3.

- $\Pr(\text{Venmo} = 0 \mid \text{not spam})$.

  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:

$$\begin{aligned}
\Pr(\text{spam} \mid x_1 = 1, x_2 = 1, x_3 = 0) &\propto \Pr(x_1 = 1, x_2 = 1, x_3 = 0 \mid \text{spam}) \Pr(\text{spam}) \\
&= \Pr(x_1 = 1 \mid \text{spam}) \Pr(x_2 = 1 \mid \text{spam}) \Pr(x_3 = 0 \mid \text{spam}) \Pr(\text{spam}) \\
&= \frac{2}{7}\frac{5}{7}\frac{3}{7}\frac{7}{10} \\
&= \frac{3}{49} \approx 0.061 \\
\Pr(\text{not spam} \mid x_1 = 1, x_2 = 1, x_3 = 0) &\propto \Pr(x_1 = 1, x_2 = 1, x_3 = 0 \mid \text{not spam}) \Pr(\text{not spam}) \\
&= \Pr(x_1 = 1 \mid \text{not spam}) \Pr(x_2 = 1 \mid \text{not spam}) \Pr(x_3 = 0 \mid \text{not spam}) \Pr(\text{not spam}) \\
&= \frac{2}{3}\frac{1}{3}1\frac{3}{10} \\
&= \frac{2}{30} \approx 0.067.
\end{aligned}$$

Since $\Pr(\text{not spam} \mid x_1 = 1, x_2 = 1, x_3 = 0)$ is proportional to a (slightly) bigger number, and the proportionality constants are the same ($p(x_1 = 1, x_2 = 1, x_3 = 0)$), we would predict "not spam."

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: You could add the following examples:

$$X_\beta = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad y_\beta = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}.$$

3.2 Exploring Bag-of-Words [2 points]

If you run `python main.py -q 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: "talk.*"

3.3 Naïve Bayes Implementation [4 points]

If you run `python main.py -q 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:

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

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

    # Compute the probability of each class i.e p(y==c), aka "baseline -ness"
    counts = np.bincount(y)
    p_y = counts / n

    # Compute the conditional probabilities i.e.
    # p(x_ij=1 | y_i==c) as p_xy[j, c]
    # p(x_ij=0 | y_i==c) as 1 - p_xy[j, c]
    p_xy = 0.5 * np.ones((d, k))

    # Use counting to populate entries of p_xy
    for c in range(k):
        X_in_c = X[y == c, :]
        p_xy[:, c] = np.mean(X_in_c, axis=0)

    self.p_y = p_y
    self.p_xy = p_xy
    self.not_p_xy = 1 - p_xy
```

The training error is approximately 0.20. The validation error is approximately 0.19.

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.

  Answer:

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

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

    # Compute the probability of each class i.e p(y==c), aka "baseline -ness"
    counts = np.bincount(y)
    p_y = counts / n
    # If we want to use Laplace smoothing for the priors, it'd be:
    # p_y = (counts + self.beta) / (n + self.num_classes * self.beta)

    # Compute the conditional probabilities with Laplace
    p_xy = 0.5 * np.ones((d, k))
    for c in range(k):
        X_in_c = X[y == c]
        n_in_c = X_in_c.shape[0]
        counts = np.sum(X_in_c, axis=0)
        p_xy[:, c] = (counts + self.beta) / (n_in_c + 2 * self.beta)
    not_p_xy = 1 - p_xy  # still works

    self.p_y = p_y
    self.p_xy = p_xy
    self.not_p_xy = not_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: 12 zeros appear without Laplace smoothing, but there are no zeros with Laplace smoothing.

- 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 of the probabilities are shunted heavily towards $\frac{1}{2}$, as expected, since we're regularizing extremely heavily towards that point.

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 $\log(ab) = \log(a) + \log(b)$,

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

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 examples, the dominant cost is computing $p(x_{ij}|y_i)$ for all d values of j and all k class labels. You can do this with three "for" loops (as in the naive Bayes `predict` function in the given code): one looping over the examples t, one looping over the features d, and one looping over the class labels k. Since each of the loops does a constant amount of work, the total time is $O(tdk)$. Note that this is much slower than using a depth m decision tree in the common case that $m \ll dk$. (However, the training and testing phases can be much faster if the examples are sparse, meaning that most values of x_{ij} are zero.)

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 -q 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: Even if you get 0 training error on the bootstrap sample that the random tree is being trained on, this likely does not correspond to 0 training error on the original training set.

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: At some point, we reach one of the termination criteria for every stump: (a) we have a single class left in the label vector or (b) we have only a single example left.

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]

   Answer:

```
class RandomForest:
    def __init__(self, num_trees, max_depth):
        self.num_trees = num_trees
        self.max_depth = max_depth
        self.trees = [RandomTree(max_depth=self.max_depth) for _ in
        ↪  range(num_trees)]

    def fit(self, X, y):
        for tree in self.trees:
            tree.fit(X, y)

    def predict(self, X_pred):
        t, d = X_pred.shape
        # get matrix of predictions of each tree on each test point
        pred_matrix = np.zeros([t, self.num_trees])
        for c, tree in enumerate(self.trees):
            y_pred_for_tree = tree.predict(X_pred)  # shape [t]
            pred_matrix[:, c] = y_pred_for_tree  # set as column for tree

        # the following block could equivalently just be
        #   return scipy.stats.mode(pred_matrix, axis=1).mode
        y_pred = np.zeros(t, dtype=np.uint8)
        for i in range(t):
```

[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)`.

```
            y_pred[i] = utils.mode(pred_matrix[i, :])
        return y_pred
```

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]

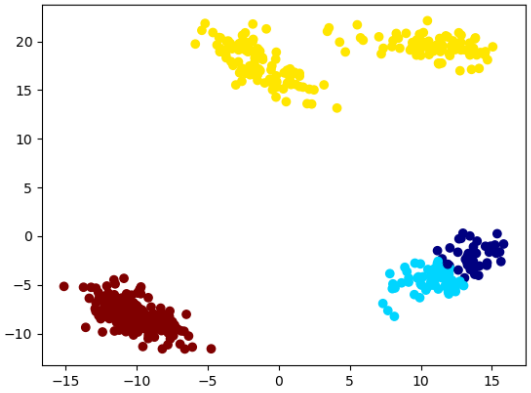
   Answer: Train error is 0, test error is 0.186 (with some variability there, of course). This is better test error than a single tree – I got 0.367 for an info gain tree, and about 0.492 for a single random tree.

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]

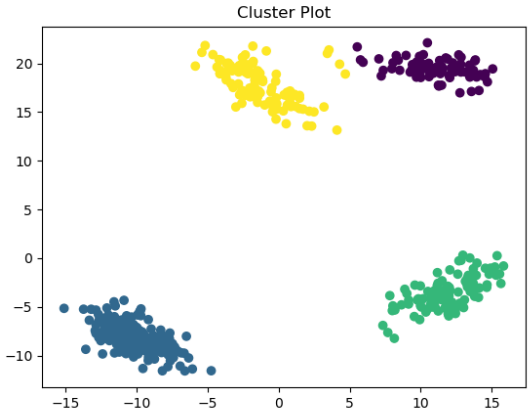
   Answer: Each tree with infinite max depth will perfectly classify whatever data used for actual fitting, i.e. the tree-specific bootstrapped data. If each training example is in more than half the trees, there will be majority correct predictions across trees in the forest. Then the forest will predict the correct label for each training example.

5 Clustering [15 points]

If you run `python main.py -q 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

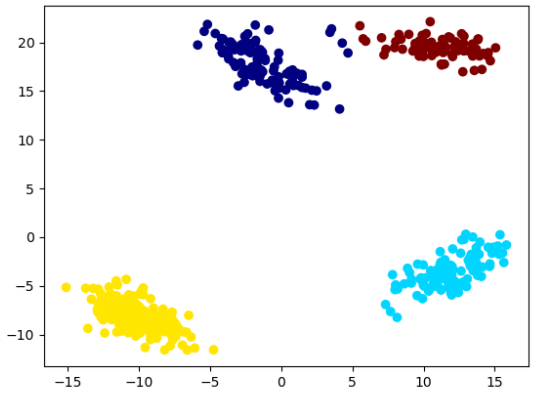
value of this above objective function. Submit this code. What trend do you observe if you print the value of this error after each iteration of the k-means algorithm? [4 points]

Answer: The error decreases monotonically over iterations.

```
def error(self, X, y, means):
    n, d = X.shape
    err = 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: I get 3071 as error. The clustering tends to be correct.



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: The error value always prefers the largest possible k, sort of like choosing the depth of a decision tree with training error.

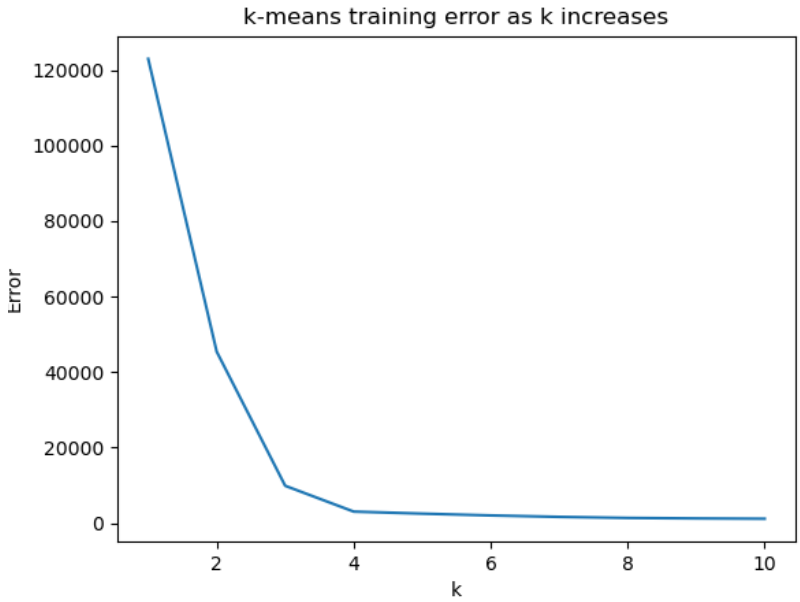
2. Is evaluating the `error` function on validation (or test) data a suitable approach to choosing k? [2 points]

   Answer: Since you have more clusters as k increases, the closest mean is still likely to be closer on new data with large values of k. So even on validation data, this objective function would likely prefer the largest value of k.

   (We wrote "test" here when we really meant "validation"; if you answered "no, because it breaks the golden rule" then you still get full points, although that's not what we intended for this question.)

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:



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: This will change based on a person's interpretation, but reasonable value might be 3 or 4.

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: Many answers are possible. For example, the spammers might modify their e-mail over time (in response to getting filtered out).

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

   Answer: Hyper-parameters (typically) control model complexity, and more complex models (typically) have lower training error (so this just leads to picking the most complex model we try rather than one that is likely to have a small test error).

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 increases, the optimization bias decreases.

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

   Answer: Large k values let you use more data (to fit the model), but are more expensive (since you need to fit k 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: Many answers are possible. In general, when we really don't want false negative, e.g. COVID-19 test, detecting cancer, etc.

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

   Answer: It's the same for all classes, so doesn't affect our decision.

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.

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

   (c) The value β in Laplace smoothing.

   Answer: The probabilities are parameters and β is a hyper-parameter.

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're given specific y_i values during training (so they have a meaning in the real world).

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; you could have two points with the same label that have other points in between with different labels.