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

Pedram Amani - 73993008
Henry Xu - 40728164

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: `https://numpy.org/doc/stable/reference/generated/numpy.argsort.html`

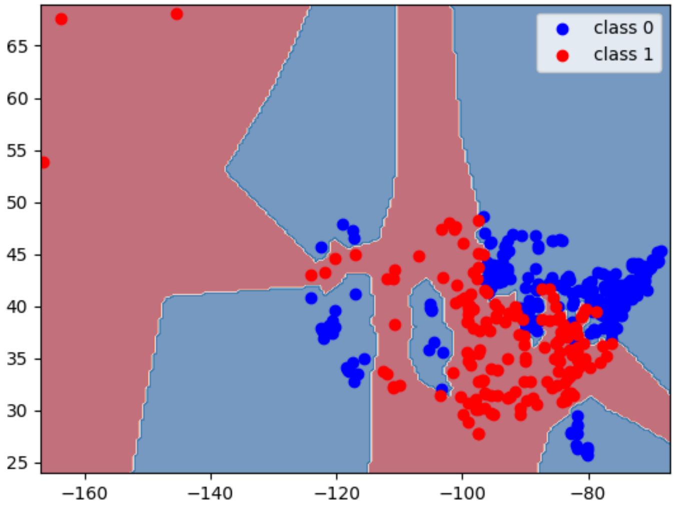
```
def predict(self, X_hat):
    n = len(X_hat)
    y_hat = np.zeros(n, dtype=np.int8)
    dist = utils.euclidean_dist_squared(self.X, X_hat)
    for i in range(n):
        top_k = np.argsort(dist[:, i])[:self.k]
        y_hat[i] = utils.mode(self.y[top_k])
    return y_hat
```

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:
   $k = 1 : E_{train} = 0.0\%,\ E_{test} = 6.45\%$
   $k = 3 : E_{train} = 2.75\%,\ E_{test} = 6.60\%$
   $k = 10 : E_{train} = 7.25\%,\ E_{test} = 9.70\%$

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: Because in the training data the closest city to a city is itself.

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: Compare results with a few more models of higher k and plot E_{test} versus k. Choose the region where E_{test} is lowest and finally choose k, erring on the side of a higher k (less complex model).

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`.

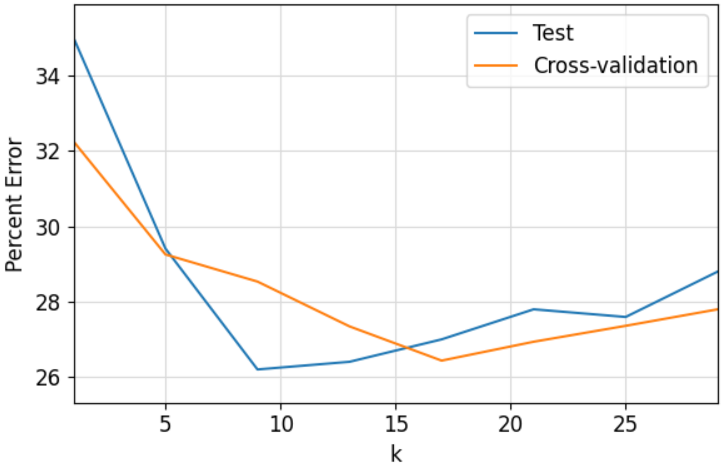
   Specifically, make sure you test on the first 10% of the data after training on the remaining 90%, then test on 10% to 20% after training on the remainder, etc – don't shuffle (so your results are consistent with ours; the data is already in random order). Implement this yourself, don't use scikit-learn 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]

```
ks = list(range(1, 30, 4))
cv_accs = np.zeros_like(ks, dtype=np.float16)
n = len(X)
n_fold = 10

for i, k in enumerate(ks):
    model = KNN(k)

    for j in range(n_fold):
        mask = np.zeros(n, dtype=np.bool8)
        mask[j * n // n_fold: (j+1) * n // n_fold] = 1
        i_validate, = np.nonzero(mask)
        i_train, = np.nonzero(~mask)
        X_train, y_train = X[i_train], y[i_train]
        X_validate, y_validate = X[i_validate], y[i_validate]

        model.fit(X_train, y_train)
        cv_accs[i] += np.mean(model.predict(X_validate) != y_validate)
    cv_accs[i] *= 100. / n_fold
```

2. The point of cross-validation is to get a sense of what the test accuracy 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 of the cross-validation and test accuracies as a function of k. Make sure your plot has axis labels and a legend. [5 points]

[1] If you haven't finish 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)`.

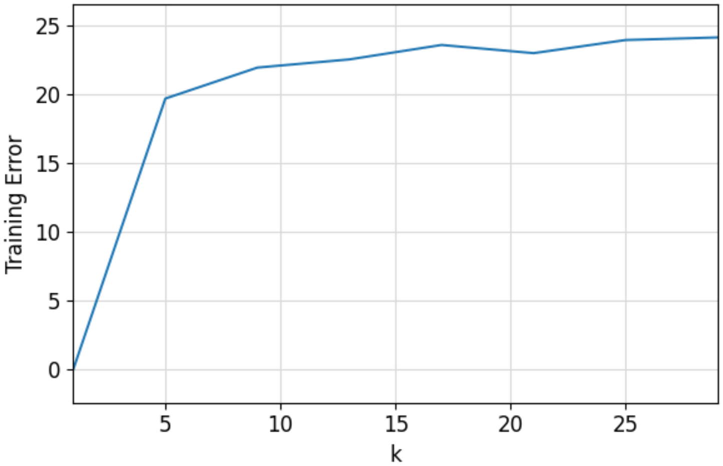


3. Which k would cross-validation choose in this case? Which k has the best test accuracy? Would the cross-validation k do okay (qualitatively) in terms of test accuracy? [2 points]

   Answer: $k = 17$ has the lowest cross-validation error $E = 27\%$ and $k = 9$ has the lowest test error $E = 26.2$. The cross-validation model only performs $\approx 3\%$ worse than the best-performing model and it is less complex.

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

   Answer: $k = 1$ trivially has the lowest $E = 0$. And out of the tested k values it is the worst-performing on the test data with an error of $E = 35\%$, or $\approx 34\%$ worse than the best-performing model.



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(lottery = 1 | not spam).

  Answer: 1/3

- 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:

$$\begin{aligned}
&\Pr(\text{spam} \mid \text{<your name>} = 1,\ \text{lottery} = 1,\ \text{Venmo} = 0) \\
&\propto\ \Pr(\text{<your name>} = 1,\ \text{lottery} = 1,\ \text{Venmo} = 0 \mid \text{spam}) \cdot \Pr(\text{spam}) \\
&\approx\ \Pr(\text{<your name>} = 1 \mid \text{spam}) \cdot \Pr(\text{lottery} = 1 \mid \text{spam}) \cdot \Pr(\text{Venmo} = 0 \mid \text{spam}) \cdot \Pr(\text{spam}) \\
&= \frac{2}{7} \cdot \frac{5}{7} \cdot \frac{3}{7} \cdot \frac{7}{10} \approx 0.0612
\end{aligned}$$

$$\begin{aligned}
&\Pr(\text{not spam} \mid \text{<your name>} = 1,\ \text{lottery} = 1,\ \text{Venmo} = 0) \\
&\propto\ \Pr(\text{<your name>} = 1,\ \text{lottery} = 1,\ \text{Venmo} = 0 \mid \text{not spam}) \cdot \Pr(\text{not spam}) \\
&\approx\ \Pr(\text{<your name>} = 1 \mid \text{not spam}) \cdot \Pr(\text{lottery} = 1 \mid \text{not spam}) \cdot \Pr(\text{Venmo} = 0 \mid \text{not spam}) \cdot \Pr(\text{not spam}) \\
&= \frac{2}{3} \cdot \frac{1}{3} \cdot 1 \cdot \frac{3}{10} \approx 0.0667
\end{aligned}$$

$$\Pr(\text{spam} \mid \text{<your name>} = 1,\ \text{lottery} = 1,\ \text{Venmo} = 0) = \frac{0.0612}{0.0612 + 0.0667} = 0.478$$

Therefore, the most likely label is 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:

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

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: Training error: 0.200, Validation error: 0.188

```
p_xy = np.array([np.sum(X[np.where(y == i)], axis=0) for i in range(len(counts))])
p_xy = np.divide(p_xy.T, counts)  # divide by number of posts in each group
```

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: `https://numpy.org/doc/stable/reference/generated/numpy.unique.html`

```
def fit(self, X, y):
    d, k = len(X[0]), len(np.unique(y))
    X = np.concatenate((X, np.zeros((self.beta * k, d)), np.ones((self.beta * k, d))),
                                                        axis=0)
    y_extra = [np.arange(k) for _ in range(2 * self.beta)]
    y = np.concatenate((y, *y_extra))
    super().fit(X, y)
```

- 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: Compared to the probabilities without smoothing, no 0s or 1s appear. In fact, 0s are replaced with $\frac{1}{s+2}$ and 1s are replaced with $1 - \frac{1}{s+2}$ as a consequence of Laplace smoothing where s is the number of posts in a given newsgroup.

- 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: No, all probability values are close to 0.5. Since $\beta = 10000$ is much larger than the number of features (words), a probability fraction is effectively reduced to $\frac{\beta}{2\beta} = 0.5$.

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 $O(nd)$, since you only need to do a constant amount of work for each $X(i, j)$ value. (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?

Answer: $O(tkd)$. For each of t examples, we need to calculate k conditional probabilities. Calculating a conditional probability requires multiplying d probabilities from the training phase. Assuming it takes $O(1)$ to find one probability among cd, the total cost is $O(tkd)$.

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 -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: Because of bootstrap sampling, the model overfits the sampled data and not all of the training data. The random nature of sampling is reflected in the varying training error between different runs.

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: Because the sampled training data will eventually be split upto a few examples with the same label, requiring no further splitting. While the depth is finite, it is higher than the regular decision tree depth since not all features are made available at a given stump.

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: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mode.html

```
class RandomForest:
    def __init__(self, max_depth, num_trees):
        self.max_depth = max_depth
        self.num_trees = num_trees
        self.trees = []

    def fit(self, X, y):
        self.trees = [RandomTree(self.max_depth) for _ in range(self.num_trees)]

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

    def predict(self, X_hat):
        y_hats = np.array([tree.predict(X_hat) for tree in self.trees])
        modes, _ = scipy.stats.mode(y_hats)
        return modes
```

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:
   DecisionTree: $E_{train} = 0$, $E_{test} = 0.367$
   RandomTree: $E_{train} = 0.182$, $E_{test} = 0.466$
   RandomForest: $E_{train} = 0$, $E_{test} = 0.212$

[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).

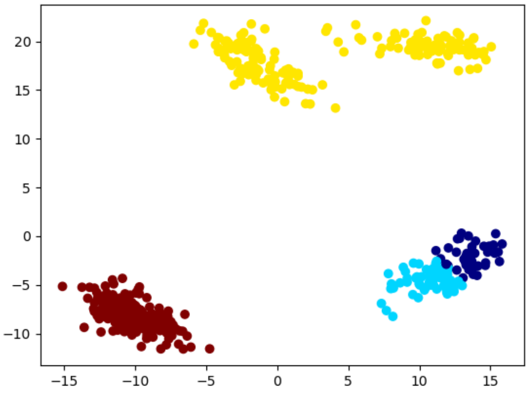
Yes, the results are expected. `RandomForest` performs better than the other models on test data. This is simply because it is less likely for the majority of 50 independent `RandomTree` to agree on an incorrect classification.

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]

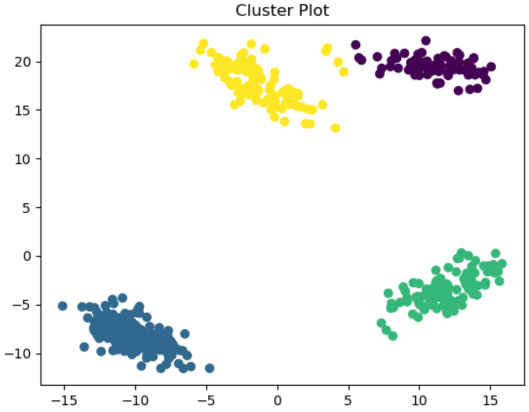
   Answer: The `RandomForest` model predicts the most frequent output among 50 `RandomTree` models. Due to bagging, each model is trained on a random sample of the training data. So while a single `RandomTree` model does not have access to the entire training set, it is very likely that a collection of 50 models do. Therefore, in this particular comparison, `RandomForest` is more likely to overfit.

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

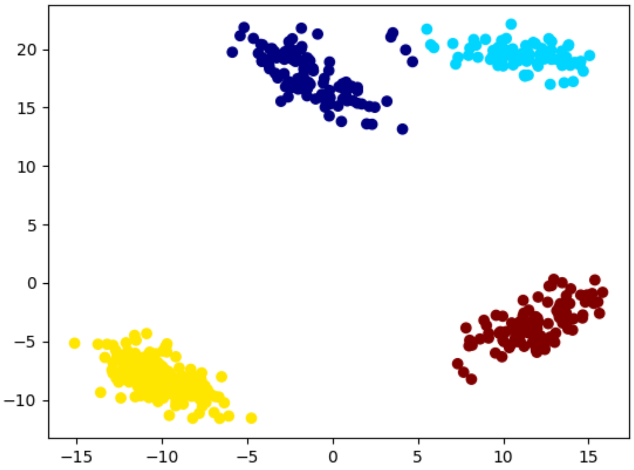
value of this above objective function. Submit this code. What trend do you observe if you print the value of this error after each iteration of the k-means algorithm? [4 points]

Answer: The trend I observed is $\approx [255, 62, 3.26, 3.08, 3.07] \cdot 10^3$ which indicates that the error rapidly converges to a locally minimum value.

```
def error(self, X, y, means):
    return np.sum((X - means[y.astype(int)]) ** 2)
```

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 was ≈ 3071, with the clustering below.



5.2 Selecting k in k-means [8 points]

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

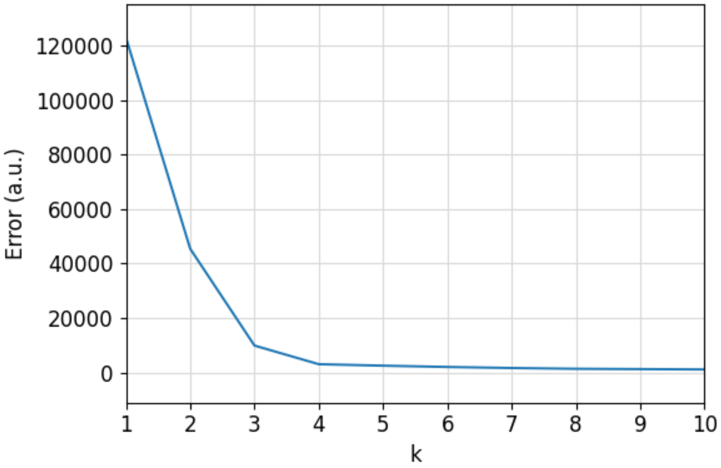
1. Explain why we should not choose k by taking the value that minimizes the `error` value. [2 points]

   Answer: Because choosing a higher value of k would lower the error, even thought it is overfitting. For example, the extreme case of $k = n$ would yield an error of 0.

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

   Answer: Not really. Generally, we would like to choose model hyper-parameters before testing to avoid overfitting to the test data.

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]



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: $k = 3$ seems to have the sharpest change in slope. Presumably, this is when the top two clusters are grouped together.

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 data may not be IID in the email spam filtering example from lecture?

   Answer: In the example given in class, emails are randomly sampled from a large pool and labeled by users. The population of users that agree to label the data is most definitely a biased sampling of all users (for example, they could be more agreeable than average). This bias will persist in the training and test data. And so the IID assumption breaks when the biased model is deployed on unbiased examples.

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

   Answer: Because a more complex model, typically yields a lower training error. This usually means the model is overfitting to the training data and would perform badly when tested on new data (i.e. would have a high E_{approx}).

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

   Answer: A model with fixed complexity is less likely to overfit to a larger training set. Similarly, a larger validation set would make it less likely to overfit hyper-parameters.

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

   Answer: The advantage is that the mean score for a set of hyper-parameters gets more accurate with higher k. A clear disadvantage is the increasing computation cost which scales linearly with k.

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 treating victims of a lethal drug, it is better to treat a non-user (false positive) than to not treat a user (false negative).

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

   Answer: Since we are interested in the probability of an example belonging to a particular class, the relative probabilities matter. In other words, in calculating $p(y \mid x_i)$, the $p(x_i)$ cancel out.

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

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

   Answer: Parameter

   (c) The value β in Laplace smoothing.

   Answer: 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: Supervised learning models are provided with y_i in the training phase, and so we expect $\hat{y}_i \in \{y_i\}$. Clustering models are not provided with y_i, and must "discover" the classification groups $\{y_i\}$.

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. A clear counter-example is our answer to Question 1.3. The process of finding the regions is more complicated in kNN and depends on the labels of training data.