CAP 4611 Assignment 2

Broc Weselmann (br142931)

Important: Submission Format [5 points]

Please make sure to follow the submission instructions posted on the course website and on the webcourses. Here is a direct link to the instructions. We will deduct marks if the submission format is incorrect, or if your handwriting is *at all* difficult to read – at least these 5 points, more for egregious issues. Be sure to include your name and student number. If you are working in a group of two, include both names and student numbers and only upload one submission. For more detailed instructions, see the link above.

1 K-Nearest Neighbours [15 points]

In the *citiesSmall* dataset (introduced in A1), 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. Also, please note that you will have to write code in the `main.py` file in order to do parts 2 and 3 of this question.

1. Write the `predict` function. Note that you can use the `utils.mode()` function in your implementation, and you can rely on this function to correctly handle potential ties when assigning a label to a training data point. Submit this code. [5 points]

   Answer:

```
def predict(self, X_hat):
    mode = utils.mode(self.y)
    y_hat = np.arange(X_hat.shape[0])
    distance_arr = euclidean_dist_squared(self.X, X_hat)

    # For each test point
    for i, test_point in enumerate(X_hat):
        # Tuple array (distance, class) for all training points
        training_points = []
        for j, training_point in enumerate(self.X):
            point_class = self.y[j]
            training_points.append((distance_arr[j, i], point_class))

        # Sorting by distance, kth closest points
        training_points.sort()
        kth_points = training_points[: self.k]

        # Getting nearest labels for kth closest points, finding prediction
```

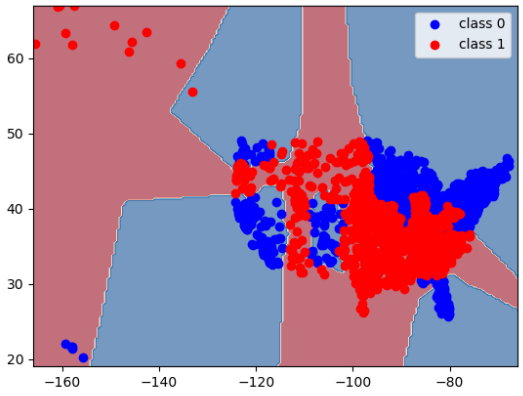
```
            k_nearest_labels = [point[1] for point in kth_points]
            numpy_k_nearest_labels = np.asarray(k_nearest_labels)
            prediction = utils.mode(numpy_k_nearest_labels)

            # Ties for even split of nearest labels
            if (
                numpy_k_nearest_labels.size % 2 == 0
                and prediction == numpy_k_nearest_labels.size / 2
            ):
                prediction = mode
            y_hat[i] = prediction
        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 error: 0.000
   k=1 Testing error: 0.065
   k=3 Training error: 0.028
   k=3 Testing error: 0.066
   k=10 Training error: 0.058
   k=10 Testing error: 0.085

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:



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

   Answer: The training error is 0 because the model finds the closest point in the training data (the point itself), computes that distance (0), and then assigns the value in $\hat{y}$ to the corresponding value in y. This results in $\hat{y}$ being the exact same as y, giving a training error of 0.

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: I would loop through all appropriate k values through the process of cross-validation. This will minimize the test error and tune the k value, a hyper-parameter.

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:

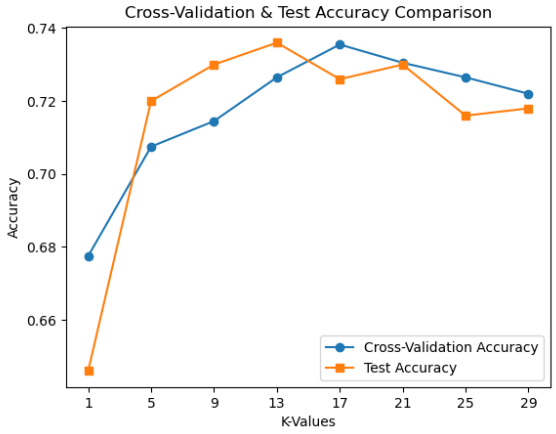
```python
# 10-fold cross-validation
cv_accs = 0.0
n = X.shape[0]
for fold in range(10):
    # Calculating partition size
    # From: https://g.co/gemini/share/1b623820fd6f
    mask_arr = np.ones(n, dtype=int)
    base_size, remainder = divmod(n, 10)
    start_index = fold * base_size + min(fold, remainder)
    partition_size = base_size + 1 if fold < remainder else base_size
    end_index = start_index + partition_size

    mask_arr[start_index:end_index] = 0
    train_mask = mask_arr.astype(bool)
    test_mask = ~train_mask

    cross_test = X[test_mask]
    cross_y_test = y[test_mask]
    cross_train = X[train_mask]
    cross_y_train = y[train_mask]

    model = KNN(k)
    model.fit(cross_train, cross_y_train)

    y_pred = model.predict(cross_test)
```

[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)`.

```
        err_test = np.mean(y_pred != cross_y_test)

        print(f"k={i} fold={fold}          Testing error: {err_test:.3f}")

        cv_accs += err_test
    cv_accs /= 10
```

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:

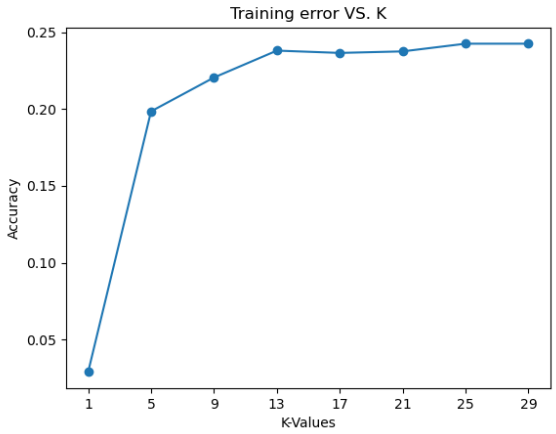


3. Which k would cross-validation choose in this case? Which k has the best test accuracy? Would the cross-validation k do okay (qualitatively) in terms of test accuracy? [2 points]

   Answer: $k = 17$ would have the best cross-validation score. The best test accuracy would be $k = 13$. The cross-validation would do relatively okay compared to the test accuracy.

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:



The value with the best training error, $k = 29$, does worse than selecting for the value with the best test error, $k = 17$.

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:

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

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: $\frac{6}{10}$

- Pr(not spam).

  Answer: $\frac{4}{10}$

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> $= 1 \mid$ spam).

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

- Pr(lottery $= 1 \mid$ spam).

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

- Pr(Venmo $= 0 \mid$ spam).

  Answer: $\frac{2}{6}$

- Pr(<your name> $= 1 \mid$ not spam).

  Answer: $\frac{3}{4}$

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

  Answer: $\frac{1}{4}$

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

  Answer: $\frac{3}{4}$

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:
$\Pr(\hat{x} \mid \text{spam}) = \Pr(\text{<your name>} = 1 \mid \text{spam}) * \Pr(\text{lottery} = 1 \mid \text{spam}) * \Pr(\text{Venmo} = 0 \mid \text{spam}) * \Pr(\text{spam})$
$= (\frac{1}{6}) * (\frac{5}{6}) * (\frac{2}{6}) * (\frac{6}{10})$
$= 0.0277$

$\Pr(\hat{x} \mid \text{not spam}) = \Pr(\text{<your name>} = 1 \mid \text{not spam}) * \Pr(\text{lottery} = 1 \mid \text{not spam}) * \Pr(\text{Venmo} = 0 \mid \text{not spam}) * \Pr(\text{not spam})$
$= (\frac{3}{4}) * (\frac{1}{4}) * (\frac{3}{4}) * (\frac{4}{10})$
$= 0.05625$

Since $\Pr(\hat{x} \mid \text{not spam}) > \Pr(\hat{x} \mid \text{spam})$, $\hat{x}$ would be classified as 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:

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

3.2 Exploring Bag-of-Words [2 points]

If you run `python main.py 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 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 = np.ones((d, k))
    for c in range(k):
        # Assign mask, create sub-matrix
        mask = y == c
        X_c = X[mask, :]

        # Sum up, divide, assign
        sum = np.sum(X_c, axis=0)
        p_xy[:, c] = sum / counts[c]

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

Naive Bayes training error: 0.200
Naive Bayes validation error: 0.188

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:

```
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 = np.ones((d, k))
    for c in range(k):
        # Assign mask, create sub-matrix
        mask = y == c
        X_c = X[mask, :]

        # Sum up, divide, assign
        sum = np.sum(X_c, axis=0)
        p_xy[:, c] = (sum + self.beta) / (counts[c] + (self.beta * k))

    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: I don't see a difference, no. $p(x_{ij} = 1 \mid y_i = 0)$ is not affected by the problem Laplace smoothing attempts to solve, which is when $p(x_{ij})$ does not exist in our dataset.

- 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: Yes, the number seems to be doubled compared with the $p(x_{ij} = 1 \mid y_i = 0)$ in the previous question. The training and validation errors also tripled - It seems like an increase of beta by a factor of 10000 messes with the probabilities immensely.

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_{ij} 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: $O(t^2) + O(n^2 + t)$

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: `RandomTree`, which uses `RandomStumpInfoGain`, removes features at random. `DecisionTree` takes into account all of the features, and can therefore overfit the training data by finding a specific decision tree to categorize all of the data. `RandomTree` cannot do this as it randomly removes features.

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: In `fit()`, where the depth is checked to be less than or equal to 1, there is also a check if the latest decision stump does nothing. When the depth is infinity, the model will create decision stumps until they are evaluated to do nothing, which prevents 'an infinite number of splitting rules.'

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:
    num_trees = None
    max_depth = None

    models = []

    def __init__(self, num_trees, max_depth):
        self.num_trees = num_trees
        self.max_depth = max_depth
        for i in range(num_trees):
            self.models.append(RandomTree(max_depth))

    def fit(self, X, y):
        # Creating, training trees
        for t in range(self.num_trees):
            self.models[t].fit(X, y)

    def predict(self, X_pred):
        n, d = X_pred.shape
        y = np.zeros(n)
```

[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)`.

```
        # Get predictions
        y_hat_trees = np.empty((n, self.num_trees))
        for t in range(self.num_trees):
            predictions = self.models[t].predict(X_pred)
            y_hat_trees[:, t] = predictions

        # Find mode of predictions, add to y
        for x in range(n):
            pred_array = y_hat_trees[x]
            y[x] = utils.mode(pred_array)

        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:
   n = 264, d = 10
   Decision tree info gain
   Training error: 0.011
   Testing error: 0.443
   Random tree info gain
   Training error: 0.159
   Testing error: 0.504
   Random Forest info gain
   Training error: 0.000
   Testing error: 0.212

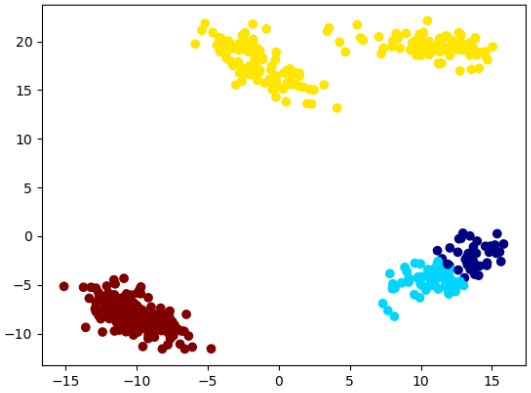
   Yes, the results are what I expected. Combining decision trees will improve the overall prediction, and with the amount of trees (50) and the potential depth of said trees, they do better at generalizing to the data.

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]

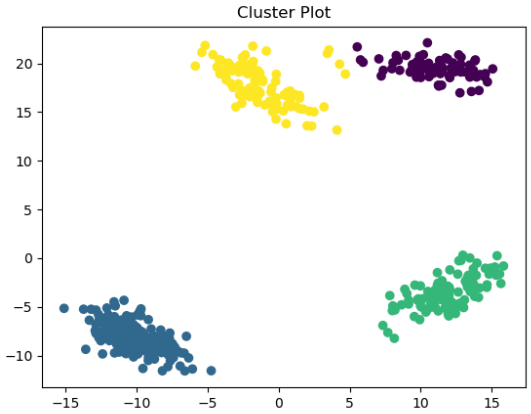
   Answer: With many trees in a RandomForest model, there is a very high probability that individual trees will overfit to the training data, resulting in a training error (for some of the training data) for many individual trees of 0. Collecting the mode of these responses will result in the prediction overfitting the training data.

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]

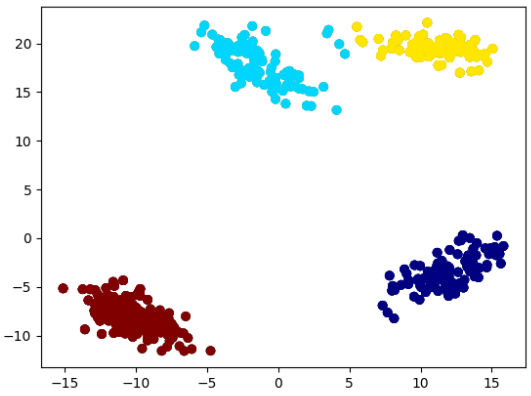
   Answer:

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

   While all of the models are randomly seeded and independent from one another, they either tend to huddle around the best value (around 3000, as shown below) or be way off (around 9000).

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 best reported error was 3071.468052653855.



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]

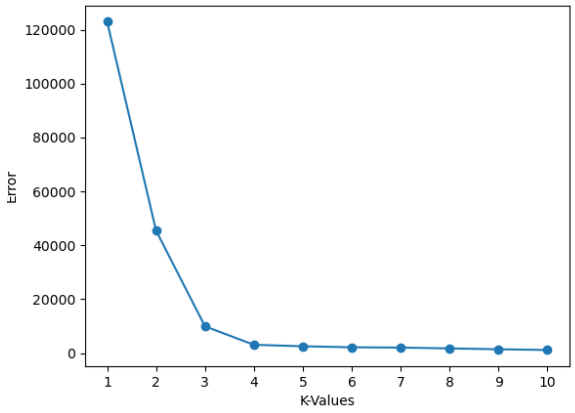
   Answer: Minimizing the error value in this case would be choosing a maximum value of k, so a k value of 10 or greater. This would arbitrarily increase the amount of clusters, when in actuality the data doesn't have that many unique clusters.

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

   Answer: Yes, as k is a hyper-parameter it must be fine-tuned via a validation set.

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: The 'elbow' of the above plot is either $k = 3$ or $k = 4$. After $k = 4$, the error 'bottoms' out and there is no use to changing it. This is reflected in the data set - There is no more than 4 unique clusters.

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: There may be more spam emails (True Positives) in the data distribution than there actually are (so, $p(spam)$ may be higher than it should), as the method of collection (people reporting emails as spam) is biased towards collecting many more spam emails than regular emails. Unless, you are Google, and decide to read people's emails. ;)

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

   Answer: Doing so will (typically) result in the model overfitting on the training data. Hyper-parameter optimization focuses on the 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: The optimization bias has an inverse relationship to n. As n decreases, the optimization bias increases, and vice versa.

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

   Answer: An advantage is that a large k value increases accuracy, and a disadvantage is that it will become more computationally expensive, especially if you are adjusting for more than one hyper-parameter.

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: In a medical application, it is preferable to have a false positive than a false negative. So, a medical imaging result showing that benign tumors are cancerous will be ruled out upon further inspection by human doctors and surgeons.

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

   Answer: $p(x_i)$ cancels out as it is the denominator on both sides of the equation and therefore a constant.

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 given the required labels (sick, not sick, etc.) in the training data and learn to fit y to those labels. Clustering models, or unsupervised learning models, are given

training data without any labels and learn to fit y to labels created by the model and its hyper-parameters.

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. See the first plot in Question #1, Class 0's 'island' in the middle of Class 1's region makes Class 1's region non-convex.