CPSC 340 Assignment 2

Talisha Griesbach (54645544), Alberto Escobar Mingo (92377860)

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 (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:

```
class KNN:
...

def predict(self, X_hat):
    """YOUR CODE HERE FOR Q1"""
    d = euclidean_dist_squared(X_hat, self.X)
    y_hat = []
    for row in d:
        nn = np.argsort(row)[0:self.k]
        y = np.array(self.y)
        y = y_hat.append(utils.mode(y[nn]))
    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]

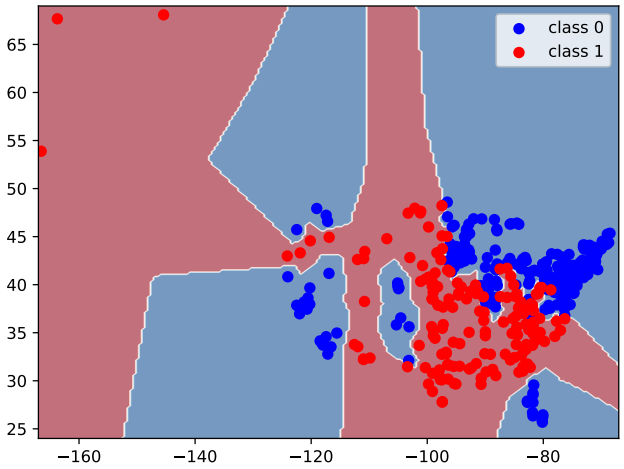
   Answer:
   k: 1
   Training error: 0.000
   Testing error: 0.0645

   k: 3
   Training error: 0.0275
   Testing error: 0.066

   k: 10
   Training error: 0.0575
   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: Each training point is its own neighbour, so error is zero. The model just memorizes the training data so when you pass training data in to calculate training error, it just matches the given point to the point it memorizes

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: We can split the training data into training and cross-validation data, then use cross-validation on various k values and select the k that has the lowest error from cross-validation.

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: See next page.

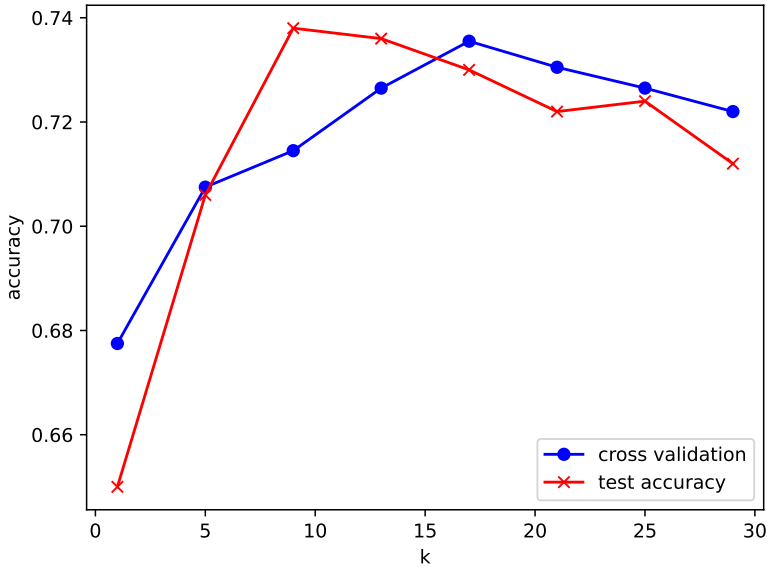
[1] If you haven't finished the code for question 1, or if you'd just prefer a slightly faster implementation, you can use scikit-learn's `KNeighborsClassifier` instead. The `fit` and `predict` methods are the same; the only difference for our purposes is that `KNN(k=3)` becomes `KNeighborsClassifier(n_neighbors=3)`.

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

    ks = list(range(1, 30, 4))
    """YOUR CODE HERE FOR Q2"""
    def splitDataSet(X, y, split=0.1):
        X = np.array(X)
        y = np.array(y)
        number_of_test_examples = int(len(y)*split)
        mask = np.zeros(len(y), dtype=bool)
        for i in range(number_of_test_examples):
            mask[i] = True
        X_cv = X[mask]
        y_cv = y[mask]
        X_train = X[~mask]
        y_train = y[~mask]
        return X_train, y_train, X_cv, y_cv

    def crossValidateKNN(X, y, k):
        accuracies = []
        for i in range(10):
            model = KNN(k)
            X_train, y_train, X_cv, y_cv = splitDataSet(X, y)
            model.fit(X_train, y_train)
            y_hat = model.predict(X_cv)
            accuracies.append(np.mean(y_hat == y_cv))

            # Append X_cv, and y_cv to X_train and y_train respectively
            X = np.concatenate([X_train, X_cv], axis=0)
            y = np.concatenate([y_train, y_cv], axis=0)
        return np.mean(accuracies)

    cv_accs = []
    test_accs = []
    training_error = []
    for k in ks:
        cv_accs.append(crossValidateKNN(X, y, k))
        model = KNN(k)
        model.fit(X, y)
        y_hat = model.predict(X_test)
        test_accs.append(np.mean(y_hat == y_test))

        y_hat = model.predict(X)
        training_error.append(np.mean(y_hat != y))
```

```
52
53      plt.plot(ks, cv_accs, label='cross validation', linestyle='-', color='blue',
        ↪  marker='o')
54      plt.plot(ks, test_accs, label='test accuracy', linestyle='-', color='red',
        ↪  marker='x')
55      plt.xlabel('k')
56      plt.ylabel('accuracy')
57      plt.legend()
58      fname = Path("..", "figs", "q2_cross validation accuracy vs test accuracy.pdf")
59      plt.savefig(fname)
60      plt.close()
61
62      plt.plot(ks, training_error, label='training error', linestyle='-', color='red',
        ↪  marker='x')
63      plt.xlabel('k')
64      plt.ylabel('training error')
65      fname = Path("..", "figs", "q2_training error for ks.pdf")
66      plt.savefig(fname)
67      plt.close()
68
```

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:

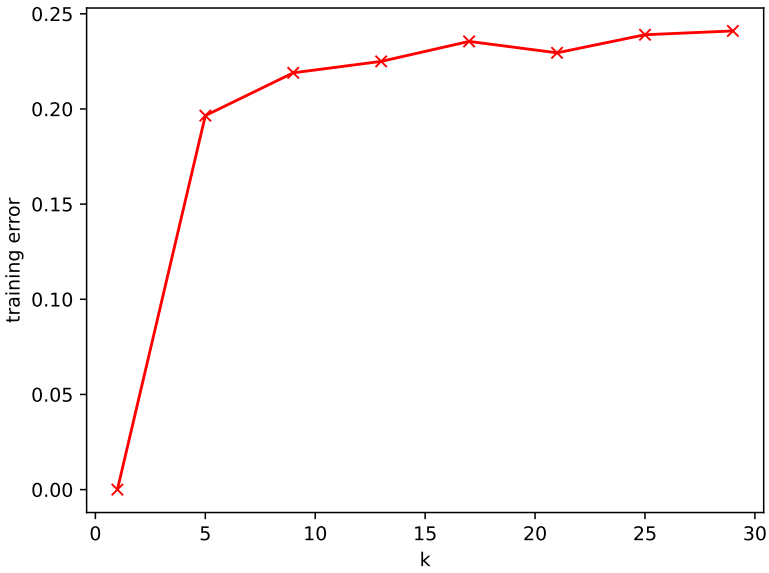


3. Which k would cross-validation choose in this case? Which k has the best test accuracy? Would the cross-validation k do okay (qualitatively) in terms of test accuracy? [2 points]

   Answer: The best k was chosen by finding the highest point in each data series cross validation and test accuracy. The best k with respect to cross validation data series is $k = 17$. The best k with respect to test accuracy data series is $k = 9$. For $k = 17$, if comparing the cross validation accuracy to the test accuracy, the difference in accuracy is okay and not too big of a difference. This is because $k = 17$ chosen from cross-validation is still near the peak maximum accuracy, and test sets are subjective too - different test sets yield different accuracy values.

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 best k in the training error plot is $k = 1$. This k value would result in a low accuracy based on the accuracy plot above (< 0.68), because $k = 1$ is over-fit to the training data.



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: Pr(spam) = 0.6

- Pr(not spam).

  Answer: Pr(spam) = 0.4

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: = 0.166...

- Pr(lottery = 1 | spam).

  Answer: = 0.833...

- Pr(Venmo = 0 | spam).

  Answer: = 0.333...

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

  Answer: = 0.75

- Pr(lottery = 1 | not spam).

  Answer: = 0.25

- Pr(Venmo = 0 | not spam)

  Answer: = 0.75

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: Most likely label for the test example is not spam. Work shown below:

$$\Pr(\text{spam} \mid \hat{x} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix})$$

$$= \Pr(\text{spam}) * \Pr(\text{<your name>} = 1 \mid \text{spam}) * \Pr(\text{lottery} = 1 \mid \text{spam}) * \Pr(\text{Venmo} = 0 \mid \text{spam})$$

$$= 0.6 * 0.166... * 0.833... * 0.333...$$

$$= 0.02777...$$

$$\Pr(\text{not spam} \mid \hat{x} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}) =$$

$$= \Pr(\text{not spam}) * \Pr(\text{<your name>} = 1 \mid \text{not spam}) * \Pr(\text{lottery} = 1 \mid \text{not spam}) * \Pr(\text{Venmo} = 0 \mid \text{not spam})$$

$$= 0.4 * 0.75 * 0.25 * 0.75$$

$$= 0.05625$$

Therefore, since

$$Pr(\text{not spam} \mid \hat{x} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}) > Pr(\text{spam} \mid \hat{x} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix})$$

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: since $\beta = 1$, you would add the the following new examples to the original data set:

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

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

```
class NaiveBayes:
    ...

    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

        """YOUR CODE HERE FOR Q3.3"""

        # 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))

        for c in range(k):
            indicies_c = np.array(y == c)
            X_c = X[indicies_c]
            p_xy[:, c] = np.mean(X_c, axis=0)

        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.

```
class NaiveBayesLaplace(NaiveBayes):
    def __init__(self, num_classes, beta=0):
        super().__init__(num_classes)
        self.beta = beta

    def fit(self, X, y):
        """YOUR CODE FOR Q3.4"""
        n, d = X.shape
        k = self.num_classes

        counts = np.bincount(y)
        p_y = (counts + self.beta) / (n + 2*self.beta)

        # 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))

        for c in range(k):
            indicies_c = np.array(y == c)
            X_c = X[indicies_c]
            p_xy[:, c] = (np.sum(X_c, axis=0) + self.beta) / (len(indicies_c) +
            ↪  2*self.beta)


        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: In the Naïve Bayes model, there were several instances where $p(x_{ij} = 1 \mid y_i = 0) = 0$. In the Naïve Bayes with Laplace-smoothing model, those instances where the probability is 0 in the original model, are now non-zero. This is because Laplace-smoothing adds extra examples to ensure that for all values of j, $p(x_{ij} = 1 \mid y_i = 0)$ will be non-zero.

- 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: When $\beta = 10000$, $p(x_{ij} = 1 \mid y_i = 0)$ is about approximate to each other for all features because all probabilities are summed with β in the denominator and βk in the denominator, which leads to $1/k = 0.25$, as $k = 4$. The Laplace smoothing has made the dataset "too smooth" and pretty much uniform. This makes sense as $\beta = 10000$ means to add 10000 samples where all the features are true for all class labels.

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: For a given test example, you would need to run the given formula for d features for k class labels. This means process one test example has a time complexity of $O(kd)$. If you have t test examples then this ends up being $O(tkd)$.

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: The Random Tree model has a non-zero training error because the feature it fits on is from a subset of all the features in the data set (boostrapped)- not the original data set which the training error is based off of, and because the features chosen for each split is random from the subset as well. Therefore, it does not take into account the "most optimal splits" and does not use every single feature available to fit at every stump.

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: It terminates eventually because the Random Tree fit method has a condition that if a `stump_model` has a `j_best == None`, it stops splitting for that leaf. The `j_best` of a stump will equal `None` if a stump is given a data set that has only one example, or if there is no splitting done (i.e. do nothing, and it does not purify the subset of data further).

[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)`.

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:
    """
    YOUR CODE HERE FOR Q4
    Hint: start with the constructor __init__(), which takes the hyperparameters.
    Hint: you can instantiate objects inside fit().
    Make sure predict() is able to handle multiple examples.
    """

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

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


    def predict(self, X_pred):
        tree_y_hat = []
        for randomTree in self.trees:
            tree_y_hat.append(randomTree.predict(X_pred))
        tree_y_hat = np.array(tree_y_hat)
        y_hat = []
        for i in range(len(tree_y_hat[0])):
            y_hat.append(utils.mode(tree_y_hat[:,i]))
        return y_hat
```

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:
   Training error: 0.000
   Testing error: 0.178

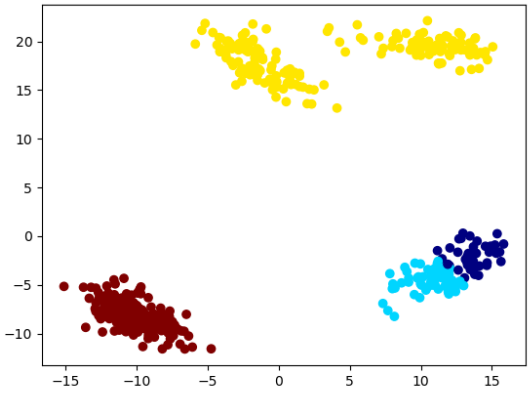
   The results are expected. Compared to the training and testing error of a Random Tree, which is 0.159 and 0.420 respectively, the Random Forest did much better. Compared to the Decision Tree errors of 0.000 and 0.367 respectively, the Random Forest also had a training error of 0.000 but without overfitting, and did better in the testing error. Following the concept of wisdom of the crowds, and by using ensemble methods to prevent overfitting, having many random trees and taking the majority predicted label of the forest, the Random Forest outperforms the Random Tree model.

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]

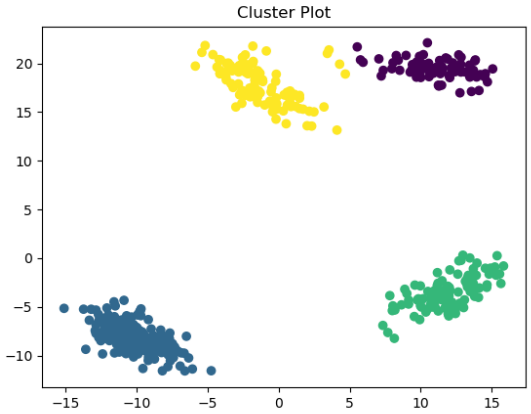
Answer: A forest will fit the data almost perfectly, by averaging the predictions of each random tree and correct any mistakes of individual trees. The forest is also not overfitting since it is not possible to overfit the data on the individual random trees with bootstrap data. Using this ensemble method you achieve wisdom of the crowds.

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]

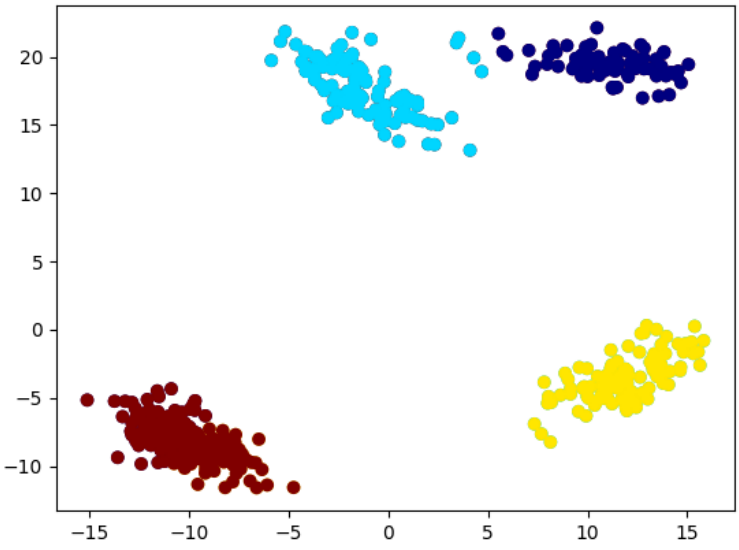
   Answer: The error decreases with every iteration in the k-means algorithm, see code below:

```python
class Kmeans:
    ...

    def error(self, X, y, means):
        """YOUR CODE HERE FOR Q5.1"""
        if len(X) != len(y):
            raise RuntimeError()
        sum = 0
        for i in range(len(y)):
            sum = sum + np.sum((X[i]-means[y[i]])**2)
        return sum
```

```python
@handle("5.1")
def q5_1():
    X = load_dataset("clusterData.pkl")["X"]

    """YOUR CODE HERE FOR Q5.1. Also modify kmeans.py/Kmeans"""
    error = np.inf
    for i in range(50):
        model = Kmeans(k=4)
        model.fit(X)
        y = model.predict(X)
        error_i = model.error(X,y,model.means)
        if (error_i < error):
            error = error_i
            plt.scatter(X[:, 0], X[:, 1], c=y, cmap="jet")

    print(f"Best error obtained: {error}")
    fname = Path("..", "figs", "kmeans_best_run.png")
    plt.savefig(fname)
    print(f"Figure saved as {fname}")
```

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: Best error obtained: 3071.4680526538546



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]

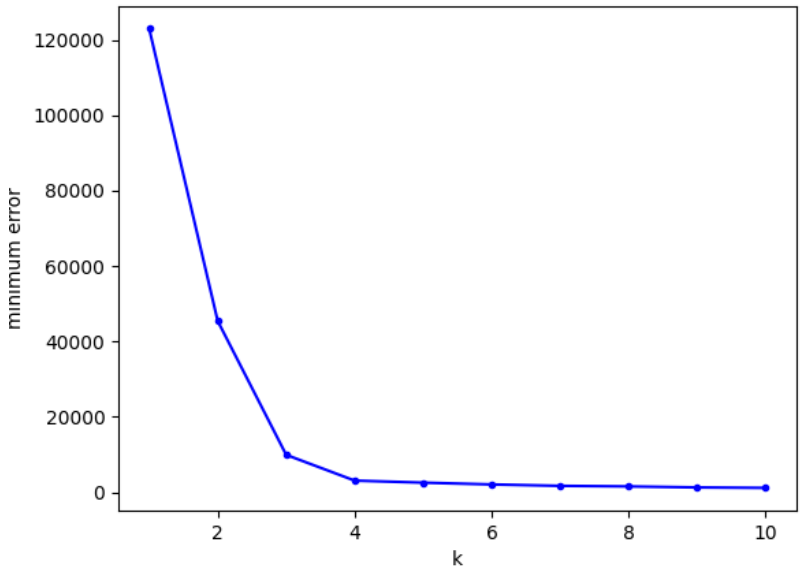
   Answer: We should not choose k by taking the value that minimizes the error value because minimum error values are often impacted by luck and is a coincidence when the number of running k-means to find this minimum value is very high. This is a classic case of optimization bias, which is when we get "lucky" with choosing a value that performs well by chance. This value is also very sensitive to initialization randomization, not necessarily the value of k.

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

   Answer: No, never evaluate anything using test data as that violates the golden rule, and the optimization bias from the previous answer still stands when choosing k from repeatedly testing the model against any data set. Furthermore, the "error" of a K-Means, an unsupervised learning model, does not always reflect the accuracy or correctness of the model in the real world. In our function, the error is the distances within the cluster, and minimizing this value does not necessarily mean they are supposed to be within one cluster. Alternatively, use the elbow method or silhouette method to determine k.

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 pointiest part of the graph above occurs at k = 3. Another reasonable value is k = 4 because it is the first k value where the minimum error begins to level off in the plateau region.

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: One reason that the data may not be IID in the email spam filtering example is that some features are not independent because they are related in the context of the email. This results in the case that the presence of a feature may be dependent on the presence of another feature.

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

   Answer: Training error is not a good gauge for true error as it is prone to over-fitting when used to select a hyper-parameter (e.g. you select a specific hyper-parameter value because is gave a training error of zero due to coincidence of running 10000 iterations).

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 data set becomes more representative of the population, and when it is used to fit and train a parametric model, will decrease the optimization bias resulting in an accurate model that is less likely to be overfit or underfit. However, there is an upper-limit for the benefits as n increases - the training error does not decrease further past a plateau point.

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

   Answer:
   The advantage of using a large k in k-fold cross-validation will result in more accurate model performance metrics as metrics will be calculated and averaged k times.

   The disadvantage of using a large k in k-fold cross-validation is that training the model will become more computationally intense since you will be essentially training/testing the model k times.

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: An increase in false positive rate is acceptable if the cost of a false positive is significantly lower compared to the cost of a false negative. An example of this would be cancer detection where a false positive would lead to more testing to confirm the diagnosis but a false negative could mean overseeing a cancer growth that could become terminal if not addressed immediately.

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

   Answer: Since $p(x_i)$ is the denominator of both sides of the inequality, it is a constant factor which does not affect the relative probability of each class compared to one another. When the denominator is cancelled (i.e. ignored), the inequality is still mathematically correct.

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: This is a parameter as it is estimated by looking at the data set we give the model to train on.

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

   Answer: This is a parameter as it is estimated by looking at the data set we give the model to train on.

   (c) The value β in Laplace smoothing.

   Answer: This is a hyper-parameter as it is not determined from the training data set and can be optimized to improve the accuracy of the model.

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: The key difference between supervised learning and clustering is that in supervised learning, the model is trained by using a labeled data set and predicts the label of test examples using the information it learned, with the same labels that were given in the data set. Meanwhile, clustering the model is trained on a unlabeled data set, the model has to determine the grouping of the data into clusters, without any predefined labels.

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, unlike k-means where it results in convex regions by design, the areas in kNN can be non-convex, this can be seen if the value if k is small when defining a kNN model.