CPSC 340 Assignment 2 (due Friday February 3 at 11:55pm)

The assignment instructions are the same as Assignment 1, except you have the option to work in a group of 2. It is recommended that you work in groups as the assignment is quite long, but please only submit one assignment for the group.

Name(s) and Student ID(s):
Hao Jie Marcus Chua 27464635
ChengYu Deng 99681827

1 Naive Bayes

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

1.1 Naive Bayes by Hand

Consider the dataset below, which has 12 training examples and 3 features:

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

The feature in the first column is <your name> (whether the e-mail contained your name), in the second column is "pharmaceutical" (whether the e-mail contained this word), and the third column is "PayPal" (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:

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

1.1.1 Prior probabilities

Compute the estimates of the class prior probabilities (you don't need to show any work):

- $p(\text{spam})$.

  Answer: $\frac{7}{12}$

- $p(\text{not spam})$.

  Answer: $\frac{5}{12}$

1.1.2 Conditional probabilities

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

- $p(\text{<your name>} = 1 \mid \text{spam})$.

  Answer: $\frac{1}{7}$

- $p(\text{pharmaceutical} = 0 \mid \text{spam})$.

  Answer: $\frac{1}{7}$

- $p(\text{PayPal} = 1 \mid \text{spam})$.

  Answer: $\frac{4}{7}$

- $p(\text{<your name>} = 1 \mid \text{not spam})$.

  Answer: $\frac{4}{5}$

- $p(\text{pharmaceutical} = 0 \mid \text{not spam})$.

  Answer: $\frac{3}{5}$

- $p(\text{PayPal} = 1 \mid \text{not spam})$.

  Answer: $\frac{1}{5}$

1.1.3 Prediction

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:

$$
\begin{aligned}
&p(\text{spam} \mid \text{<your name>} = 1, \text{pharmaceutical} = 0, \text{PayPal} = 1) \\
&= \frac{p(\text{spam})p(\text{<your name>} = 1, \text{pharmaceutical} = 0, \text{PayPal} = 1 \mid \text{spam})}{p(\text{<your name>} = 1, \text{pharmaceutical} = 0, \text{PayPal} = 1)} \\
&= \frac{p(\text{spam})p(\text{<your name>} = 1 \mid \text{spam})p(\text{pharmaceutical} = 0 \mid \text{spam})p(\text{PayPal} = 1 \mid \text{spam})}{p(\text{<your name>} = 1, \text{pharmaceutical} = 0, \text{PayPal} = 1)} \\
&= \alpha p(\text{spam})p(\text{<your name>} = 1 \mid \text{spam})p(\text{pharmaceutical} = 0 \mid \text{spam})p(\text{PayPal} = 1 \mid \text{spam})
\end{aligned}
$$

$$= \alpha \cdot \frac{7}{12} \cdot \frac{1}{7} \cdot \frac{1}{7} \cdot \frac{4}{7}$$
$$= \frac{1}{147}\alpha$$
$$\approx 0.00680\alpha$$

$$p(\text{not spam} \mid <\text{your name}> = 1, \text{pharmaceutical} = 0, \text{PayPal} = 1)$$
$$= \frac{p(\text{not spam})p(<\text{your name}> = 1, \text{pharmaceutical} = 0, \text{PayPal} = 1 \mid \text{not spam})}{p(<\text{your name}> = 1, \text{pharmaceutical} = 0, \text{PayPal} = 1)}$$
$$= \frac{p(\text{not spam})p(<\text{your name}> = 1 \mid \text{not spam})p(\text{pharmaceutical} = 0 \mid \text{not spam})p(\text{PayPal} = 1 \mid \text{not spam})}{p(<\text{your name}> = 1, \text{pharmaceutical} = 0, \text{PayPal} = 1)}$$
$$= \alpha p(\text{not spam})p(<\text{your name}> = 1 \mid \text{not spam})p(\text{pharmaceutical} = 0 \mid \text{not spam})p(\text{PayPal} = 1 \mid \text{not spam})$$
$$= \alpha \cdot \frac{5}{12} \cdot \frac{4}{5} \cdot \frac{3}{5} \cdot \frac{1}{5}$$
$$= \frac{1}{25}\alpha$$
$$\approx 0.04\alpha$$

The most likely label for the test example is 'not spam'.

1.2 Bag of Words

If you run `python main.py 1.2`, 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.*'

1.3 Naive Bayes Implementation

If you run `python main.py 1.3` 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

```
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.zeros((d, k))
    # p_xy[j, c] = p(includes word j ^ newsgroup c|newsgroup c)

    # iterate over X_dataset (each row is one piece of data, each column is one word)
    rows, cols = X.shape
    for i in range(rows):  # for each row i,
        newsgroup = y[i]  # find corresponding newsgroup (y[i])
        for j in range(cols):  # for each bool value indexed j (in that row)
            if X[i][j] == 1:
                p_xy[j, newsgroup] += 1

    # iterate over p_xy
    rows, cols = p_xy.shape
    for j in range(rows):  # each row gives word info for j-th word
        for c in range(cols):  # for each newsgroup indexed c,
            p_xy[j, c] = p_xy[j, c] / counts[c]

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

1.4 Runtime of Naive Bayes for Discrete Data

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?

Answer: $O(tdk)$

2 K-Nearest Neighbours

In *citiesSmall* dataset, nearby points tend to receive the same class label because they are part of the same state. This indicates that a $k-$nearest neighbours classifier might be a better choice than a decision tree (while naive Bayes would probably work poorly on this dataset). The file *knn.py* has implemented the training function for a $k-$nearest neighbour classifier (which is to just memorize the data) but the predict function always just predicts 1.

2.1 KNN Prediction

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.

Hint: although it is not necessary, you may find it useful to pre-compute all the distances (using the `utils.euclidean_dist_squared`) and to use the numpy's `sort` and/or `argsort`.

1. Hand in the predict function.

   Answer:

```python
def predict(self, X_hat):
        """YOUR CODE HERE FOR Q2.1"""
        Y_hat = None  # output of the function
        distance = euclidean_dist_squared(self.X, X_hat)
        dis_index = np.argsort(np.array(distance), axis=0)
        y_temp = np.take(self.y, dis_index)
        y_temp = y_temp[:self.k, :]
        y_count = np.apply_along_axis(np.bincount, axis=0, arr=y_temp)
        Y_hat = np.argmax(y_count, axis=0)
        return Y_hat
```

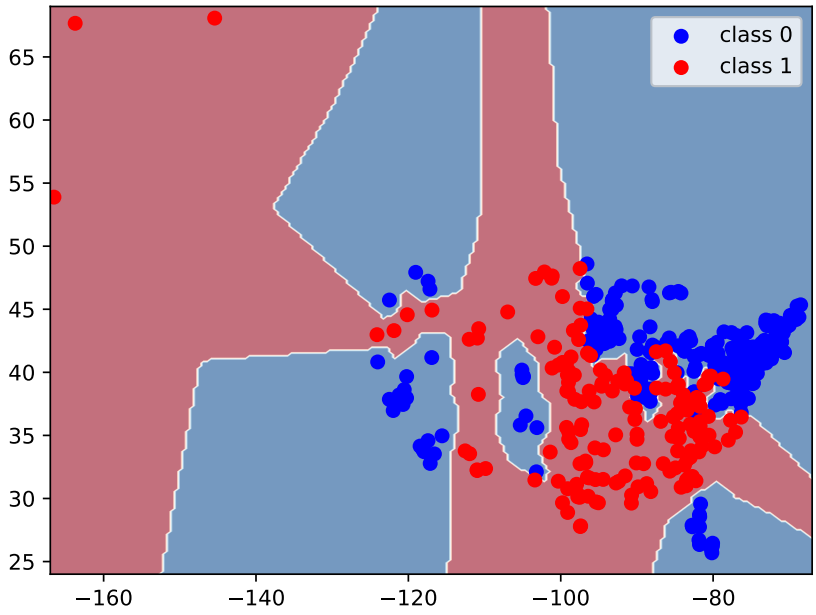
2. Report the training and test error obtained on the *citiesSmall* dataset for $k = 1$, $k = 2$, and $k = 3$.

   Answer:

	k=1	**k=2**	**k=3**
Training Error	0	0.035	0.0275
Testing Error	0.0645	0.092	0.066

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.

   Answer:



4. If we entered the coordinates of Vancouver into the predict function, would it be predicted to be in a blue state or a red state (for $k = 1$)?

   Answer: blue state

5. Why is the training error 0 for $k = 1$?

   Answer: For every data point, when it tries to find the 1 nearest neighbor, it will always find itself in the training set, which makes the training error equal to 0.

6. If you didn't have an explicit test set, how would you choose k?

   Answer: We can create a validation set from the training set and use the result on validation set to fine-tune our hyper-parameter.

Hint: when writing a function, it is typically a good practice to write one step of the code at a time and check if the code matches the output you expect. You can then proceed to the next step and at each step test is if the function behaves as you expect. You can also use a set of inputs where you know what the output should be in order to help you find any bugs. These are standard programming practices: it is not the job of the TAs or the instructor to find the location of a bug in a long program you've written without verifying the individual parts on their own.

2.2 Picking k in KNN

The file `data/ccdebt.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 bedrock principle: we don't want to look at the test data when we're picking k. Inside the `q2_2()` function of `main.py`, implement 10-fold cross-validation, evaluating on the `ks` set there ($k = \{1, 5, 9, \ldots, 29\}$), 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.

   Answer:

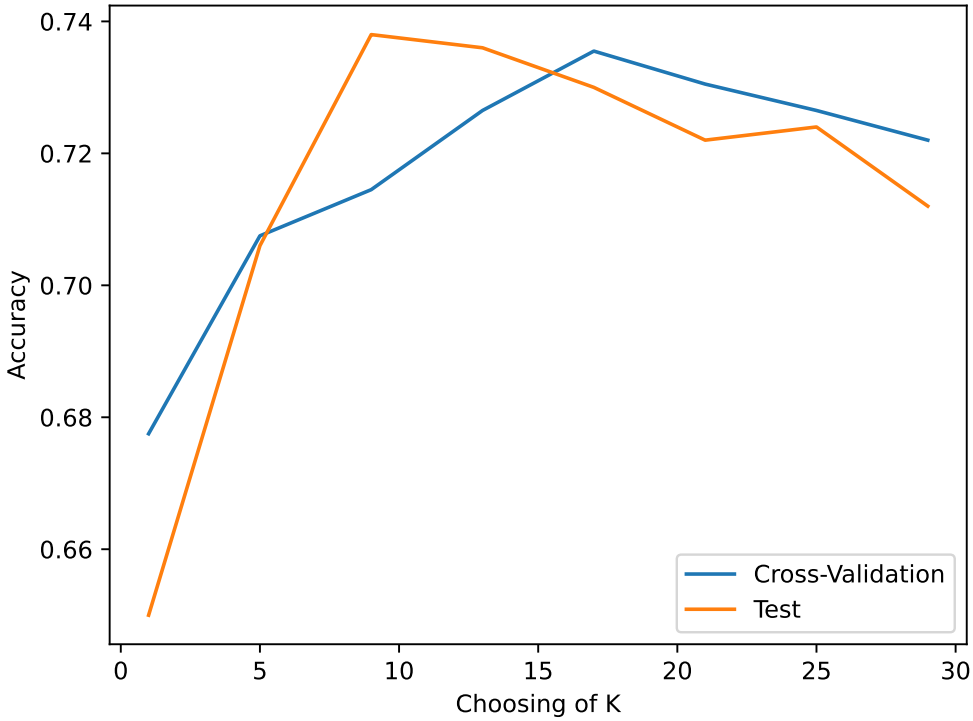
[1] If you haven't finished the code for question 2.1, or if you'd just prefer a slightly faster implementation, you can use scikit-learn's `KNeighborsClassifier` instead. The `fit` and `predict` methods are the same; the only difference for our purposes is that `KNN(k=3)` becomes `KNeighborsClassifier(n_neighbors=3)`.

```
cv_accs = np.zeros(8)
test_acc = np.zeros(8)
train_err = np.zeros(8)
num = 0
for k in ks:
    knn2_2 = KNeighborsClassifier(k)
    mean_acc = 0
    for i in range(10):
        maskX = np.ones(X.shape[0], dtype=bool)
        masky = np.ones(y.shape[0], dtype=bool)
        maskX[int(i / 10 * maskX.shape[0]):int((i + 1) / 10 *
                                               maskX.shape[0])] = 0
        masky[int(i / 10 * masky.shape[0]):int((i + 1) / 10 *
                                               masky.shape[0])] = 0
        X_train = X[maskX]
        y_train = y[masky]
        X_val = X[~maskX]
        y_val = y[~masky]
        knn2_2.fit(X_train, y_train)
        y_hat = knn2_2.predict(X_val)
        mean_acc = mean_acc + np.sum(y_hat == y_val) / y_val.size
    mean_acc = mean_acc / 10
    cv_accs[num] = mean_acc
    knn2_2.fit(X, y)
    test_hat = knn2_2.predict(X_test)
    test_acc[num] = np.sum(test_hat == y_test) / y_test.size
    knn2_2.fit(X, y)
    train_hat = knn2_2.predict(X)
    train_err[num] = 1 - np.sum(train_hat == y) / y.size
    num = num + 1
plt.plot(ks, cv_accs)
plt.plot(ks, test_acc)
plt.legend(['Cross-Validation', 'Test'])
plt.xlabel('Choosing of K')
plt.ylabel('Accuracy')
fname = Path("..", "figs", "q2_curve.pdf")
plt.savefig(fname)
print("\nFigure saved as '%s'" % fname)

plt.figure()
plt.plot(ks, train_err)
plt.legend('Train')
plt.xlabel('Choosing of K')
plt.ylabel('Error')
fname = Path("..", "figs", "q2_traincurve.pdf")
plt.savefig(fname)
print("\nFigure saved as '%s'" % fname)
```

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 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.

Answer:

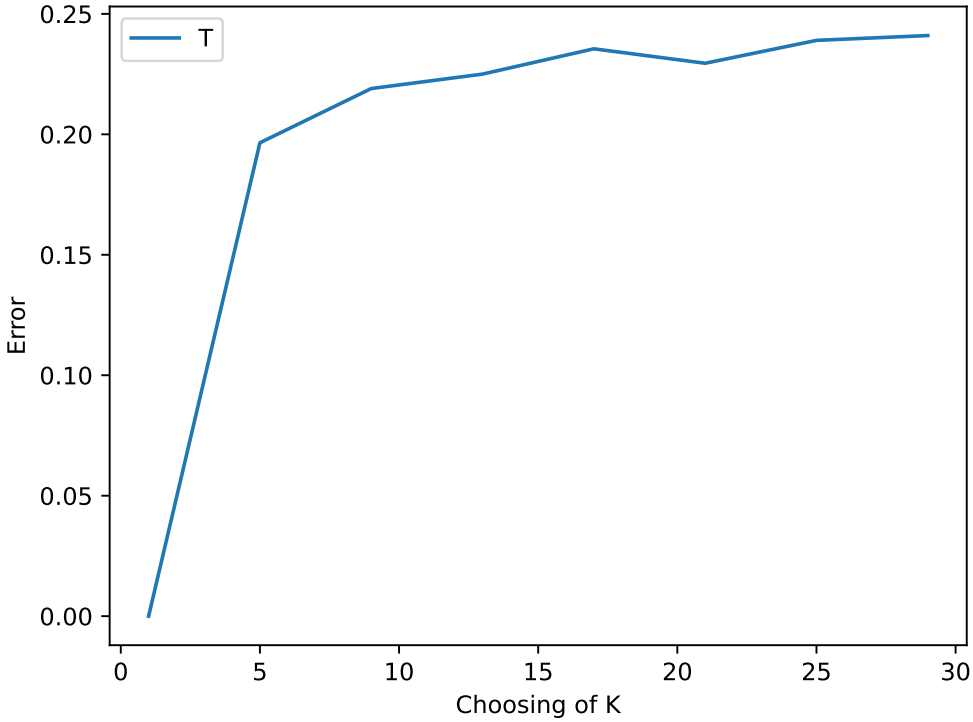


3. Which k would cross-validation choose in this case? Which k has the best test accuracy? Would the cross-validation k do okay (qualitatively) in terms of test accuracy?

   Answer:
   cross-validation choose: k=17
   best test accuracy: k=9
   Yes.

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?

   Answer:
   Not good. When k=1, training error always reaches its best to 0, but at that time, test error is not good.


T
0.25
0.20
0.15
0.10
0.05
0.00
Error
0
5
10
15
20
25
30
Choosing of K

3 Random Forests

3.1 Implementation

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 3` 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?

   Answer: The random tree model does not have a training error of 0 because it uses RandomStump and takes a bootstrap sample of the data before fitting. When a bootstrap sample is used, some specific data points will be duplicated, and some will be missing. As such, we cannot perfectly overfit the data anymore in such a way that all data points are 'memorised', hence training error is non-zero. This is contrasted to the 0 training error for decision tree due to overfitting where all the data points in training dataset were memorised.

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?

   Answer: Even with an infinite maximum depth, training functions terminate when the terminal nodes contain a minimum of one data point. So there will not be 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.

[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)`.

```
class RandomForest:
    num_trees = None
    max_depth = None
    trees = None
    y_matrix = None

    def __init__(self, num_trees, max_depth):
        self.num_trees = num_trees
        self.max_depth = max_depth

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

    def predict(self, X):
        rows = len(X)
        self.y_matrix = []
        for tree in self.trees:
            curr_tree_y_values = tree.predict(X)
            self.y_matrix.append(curr_tree_y_values)

        y = np.zeros(rows)
        for i in range(rows):
            all_y_values = []
            for y_row in self.y_matrix:
                all_y_values.append(y_row[i])
            y[i] = stats.mode(all_y_values)

        self.y_matrix = None
        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.

   Answer: Training error = 0.000, Testing error = 0.159 (fluctuates between 0.15-0.2).

   The training error for random forest is consistently 0, unlike that for the single RandomTree. This is as expected as explained in the answer for question 5. The training error for a single decision tree is also 0, but this is due to severe overfitting, a different reason from random forest.

The testing error for random forest is lower than that for random tree. This is also due to the voting ensemble method which improves predictions of multiple classifiers if the errors from individual random trees are independent.

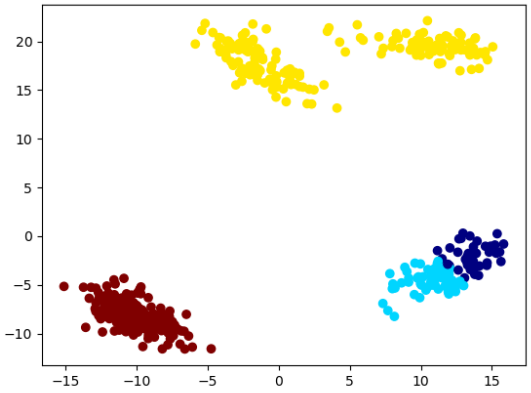
The testing error for random forest is also lower than that for a single decision tree. This is because of bootstrapping to generate different 'versions' of the dataset in random trees that roughly maintains trends in the dataset through fitting. With this general technique, random forest is able to more accurately predict data points than a decision tree.

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

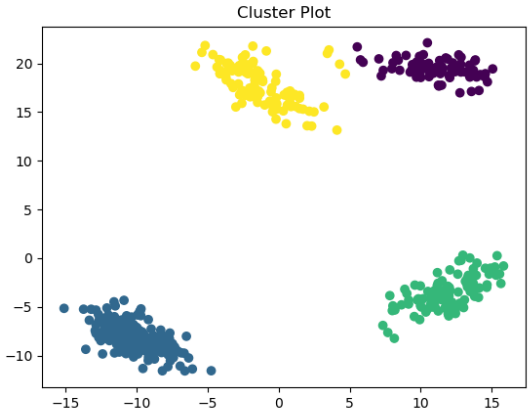
   Answer: The random forest typically has a training error of 0, even though it uses random trees with non-zero training error. This is because the random trees make independent errors when fitting on the training data. Hence, the voting mechanism of random forest can often successfully eliminate the specific overfitting of each classifier, resulting in a typical training error of 0.

4 K-Means Clustering

If you run `python main.py 4`, 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 due to the label switching problem.) But the 'correct' clustering (that was used to make the data) is something more like this:



4.1 Selecting among k-means Initializations

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. Hand in your code.

   Answer:

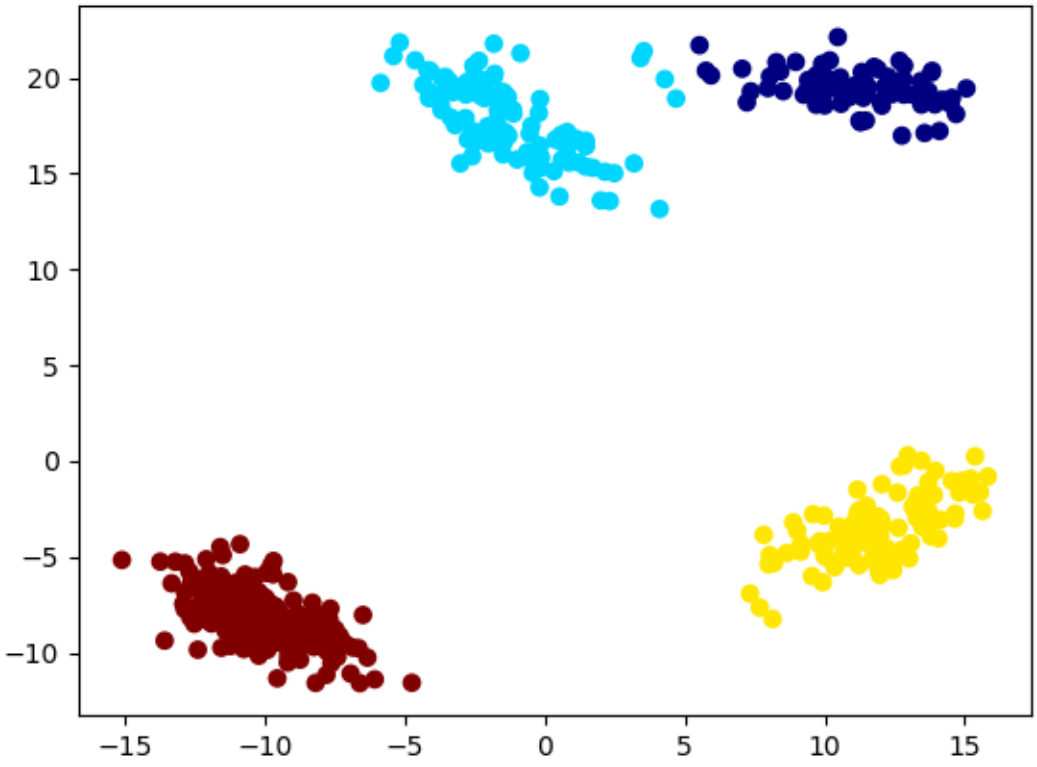
```python
def error(self, X, y, means):
        """YOUR CODE HERE FOR Q4.1"""
        n, d = X.shape
        err = 0
        errL1 = 0
        for i in range(n):
            err = err + (X[i, 0]-means[y[i], 0])**2 + (X[i, 1]-means[y[i], 1])**2
            errL1 = errL1 + abs(X[i, 0] - means[y[i], 0]) + abs(X[i, 1] -
                                                                means[y[i], 1])

        return err,errL1
```

2. Instead of printing the number of labels that change on each iteration, what trend do you observe if you print the value of this error after each iteration of the k-means algorithm?

   Answer: This error decreases after each iteration.

3. 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.

Answer:
Lowest Error: 3071.4680526538564


20
15
10
5
0
−5
−10
−15
−10
−5
0
5
10
15

4.2 Selecting k in k-means

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

1. Explain why the error function should not be used to choose k.

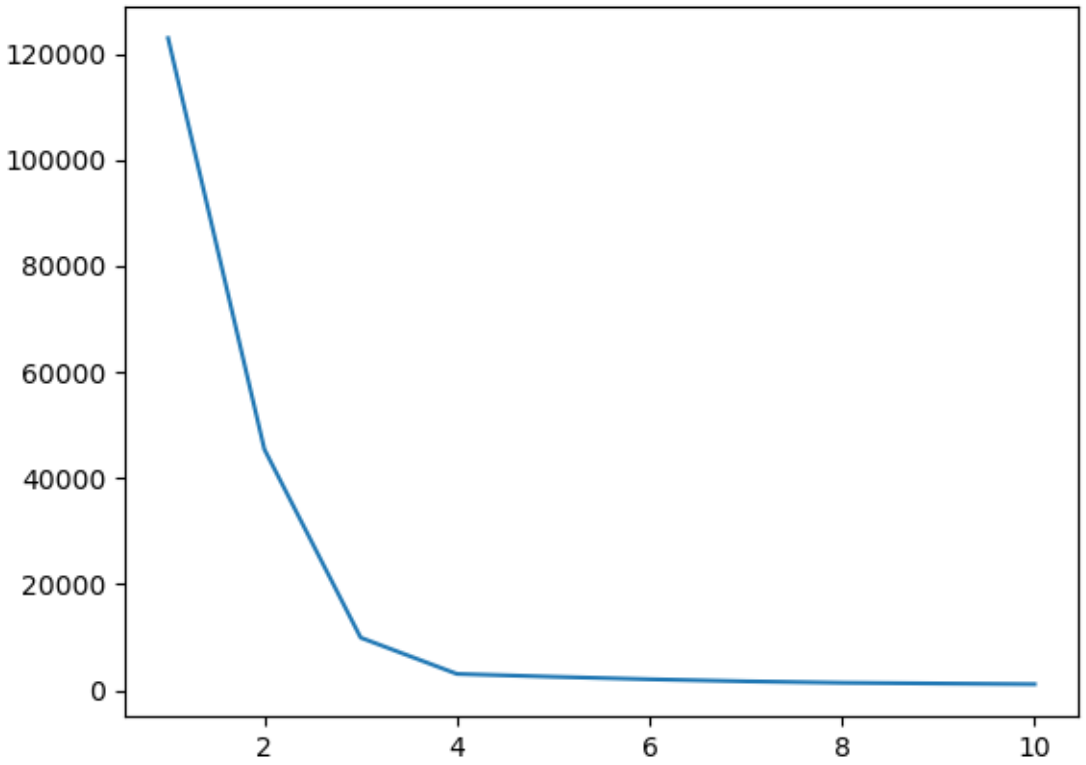
   Answer: The error function helps to determine the most suitable clustering for a fixed value of k. This is done by running k-means multiple times and choosing the lowest error, but only after k has been fixed. Running this error function with different k values will not help us determine the optimal k.

2. Explain why even evaluating the error function on test data still wouldn't be a suitable approach to choosing k.

   Answer: We should only use test set to verify our model once and should not touch it during whole training process. If we do so, it will affect generalization when we test the model using other data.

3. Hand in a plot of the minimum error found across 50 random initializations, as you vary k from 1 to 10.

   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.

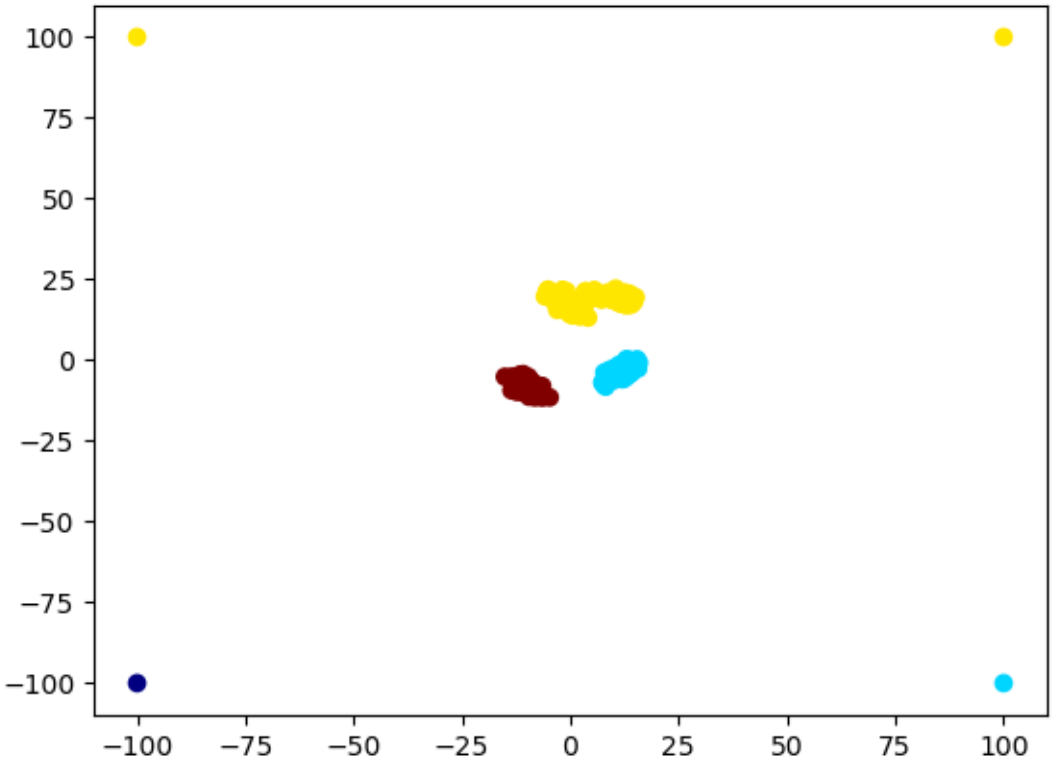
Answer: 3. The slope of curve changes dramatically at that point.

4.3 k-Medians

The data in *clusterData2.pkl* is the exact same as the above data, except it has 4 outliers that are very far away from the data.

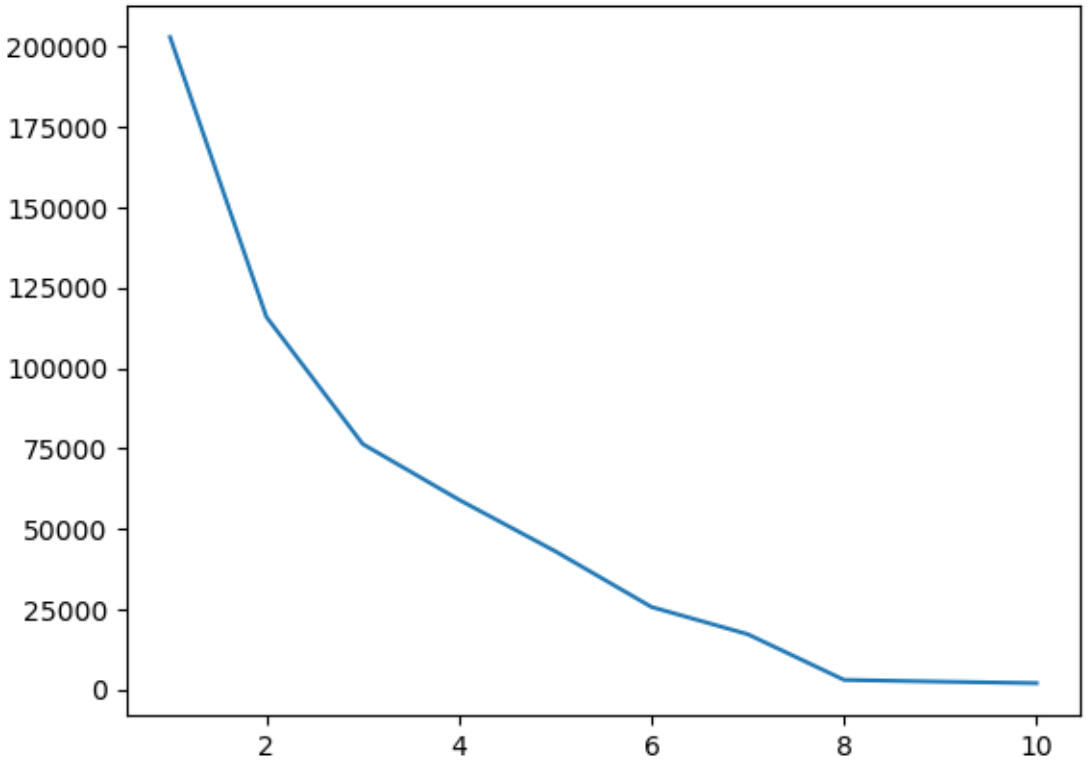
1. Visualize the clustering obtained by running k-means 50 times (with $k = 4$) on *clusterData2.pkl* and taking the one with the lowest error. Are you satisfied with the result?

   Answer:
   No. Since the data points are symmetric, clusters are expecting to have symmetry, which is not in the current clusters.



2. Hand in the elbow plot for this data. What values of k might be chosen by the elbow method for this dataset?

   Answer:
   k = 8

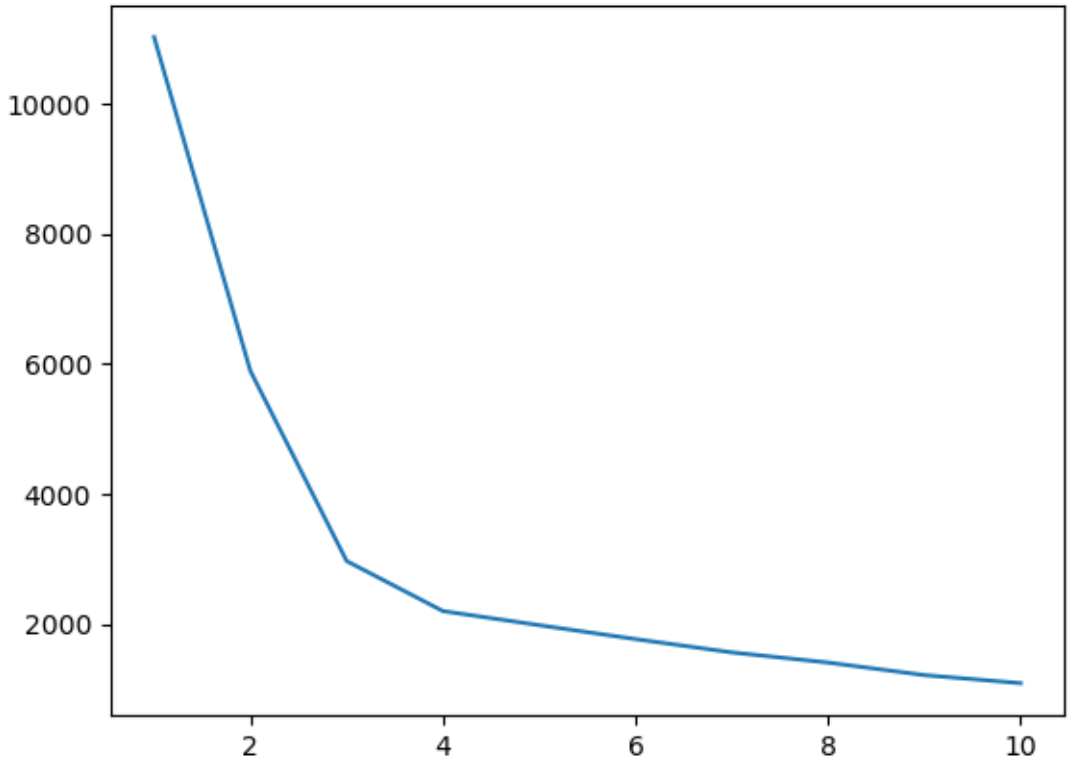


3. Instead of the squared distances between the examples x_i and their cluster centers, consider measuring the distance to the cluster centers in the L1-norm,

$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{d} |x_{ij} - w_{y_i j}|.$$

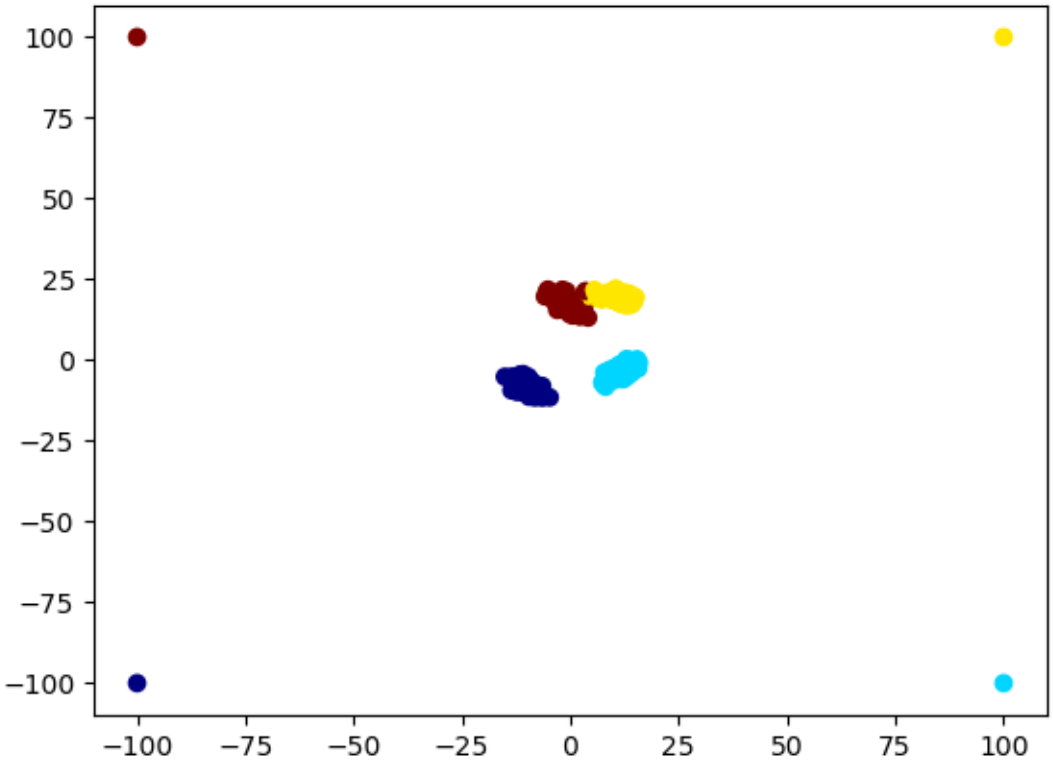
Hand in the elbow plot for k-means when we measure the error of the final model with the L1-norm. What value of k would be chosen by the elbow method?

Answer:
k = 3



4. The k-means algorithm tries to minimize the squared error and not the L1-norm error, so in the last question there is a mis-match between what the learning algorithm tries to minimize and how we measure the final error. We can try to directly minimize the L1-norm with the *k-medians* algorithm, which assigns examples to the nearest w_c in the L1-norm and to update the w_c by setting them to the "median" of the points assigned to the cluster (we define the d-dimensional median as the concatenation of the median of the points along each dimension). Implement the k-medians algorithm, and hand in your code and the the plot obtained by minimizing the L1-norm error across 50 random initializations of k-medians with $k = 4$.

   Answer:


100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

```python
class Kmedians:
    means = None

    def __init__(self, k):
        self.k = k

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

        means = np.zeros((self.k, d))
        for kk in range(self.k):
            i = np.random.randint(n)
            means[kk] = X[i]

        while True:
            # iterations of k-medians
            y_old = y

            # Compute L1 norm to each mean
            n, d = X.shape
            T, d = means.shape
            distance_matrix = np.zeros([n, T])
            for i in range(n):
                for j in range(T):
                    distance_matrix[i, j] = abs(X[i, 0] -
                                                means[j, 0]) + abs(X[i, 1] -
                                                                   means[j, 1])
            distance_matrix[np.isnan(distance_matrix)] = np.inf
            y = np.argmin(distance_matrix, axis=1)

            # Update means
            for kk in range(self.k):
                if np.any(
                        y == kk
                ):  # don't update the mean if no examples are assigned to it (one of several possible approaches)
                    means[kk] = np.median(X[y == kk], axis=0)

            changes = np.sum(y != y_old)
            # print('Running K-means, changes in cluster assignment = {}'.format(changes))

            # Stop if no point changed cluster
            if changes == 0:
                break

            # print(self.error(X, y, means))
        self.err, self.errL1 = self.error(X, y, means)
        self.means = means

    def predict(self, X_hat):
        means = self.means
        n, d = X_hat.shape
        T, d = means.shape
        distance_matrix = np.zeros([n, T])
        for i in range(n):
            for j in range(T):
                distance_matrix[i, j] = abs(X_hat[i, 0] -
                                            means[j, 0]) + abs(X_hat[i, 1] -
                                                               means[j, 1])
        distance_matrix[np.isnan(distance_matrix)] = np.inf
        return np.argmin(distance_matrix, axis=1)

    def error(self, X, y, means):
        """YOUR CODE HERE FOR Q4.1"""
        n, d = X.shape
        err = 0
        errL1 = 0
        for i in range(n):
            err = err + (X[i, 0] - means[y[i], 0])**2 + (X[i, 1] -
                                                         means[y[i], 1])**2
            errL1 = errL1 + abs(X[i, 0] - means[y[i], 0]) + abs(X[i, 1] -
                                                                means[y[i], 1])

        return err, errL1
```

5 Very-Short Answer Questions

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 feature transformation that you might do to address a "coupon collecting" problem in your data?

   Answer: Discretization i.e. binning, to turn numerical features into categorical features so that we can reduce the overall number of categories and obtain a representative for each feature category in our data.

2. What is one reason we would want to look at scatterplots of the data before doing supervised learning?

   Answer: Looking at a scatterplot helps us quickly determine the presence of any correlation at a glance through possible clustering or separability of our data points, which will help us ascertain how we can apply supervised learning on the dataset.

3. When we fit decision stumps, why do we only consider $>$ (for example) and not consider $<$ or $\geq$?

   Answer: Testing for $<$ will give equivalent rules as we are just swopping the left and right branches of the stump, and testing for $\geq$ is more or less equivalent to $>$ since only the data points equal to the threshold are shifted to the opposite side, making it redundant.

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

   Answer: Individuals or corporations that generate spam emails are constantly evolving the content of their emails to avoid being picked up by spam filters, hence future data to be predict is no longer sampled from the same distribution that the model was trained on.

5. What is the difference between a validation set and a test set?

   Answer: Validation set is used to tune the hyperparameters of a model, using part of our training data to approximate test error. Test set is used to evaluate the tuned model's performance on new data.

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

   Answer: Sub-optimal hyper-parameters will be chosen as we are choosing them for best performance in the training dataset specifically - but this will mean the hyper-parameter values chosen do not help to generalise as well on new data.

7. If you can fit one model in 10 sec., how long (in days) does it take to find the best among a set of 16 hyperparameter values using leave-one-out cross-validation on a dataset containing $n = 4320$ examples?

   Answer: 10 * 4320 * 16 = 691200 seconds = 8 days

8. Naïve Bayes makes the assumption that all features are conditionally independent given the class label. Why is this assumption necessary and what would happen without it?

   Answer: This assumption is necessary for the simplification of conditional probability values required for our calculations. Without the assumption, it will be extremely difficult to obtain the complicated conditional probabilities needed; or if we still go ahead with the assumption, we will get sub-optimal predictions.

9. Why is KNN considered a non-parametric method and what are two undesirable consequences of KNNs non-parametric design?

   Answer: The number of parameters for the KNN model grows with more data i.e. depends on 'n'. Undesirable consequences are that the model will require infinite memory as n approaches infinity, and has slow prediction time with high 'n'.

10. For any parametric model, how does increasing number of training examples n affect the two parts of the fundamental trade-off.

    Answer: With more training examples, the approximation error gets smaller. Training error should stay the same but it is inconclusive.

11. Suppose we want to classify whether segments of raw audio represent words or not. What is an easy way to make our classifier invariant to small translations of the raw audio?

    Answer: We can use data augmentation to add transformed versions of our raw audio training examples to the training set.

12. Both supervised learning and clustering models take in an input x_i and produce a label y_i. What is the key difference?

    Answer: For supervised learning, we are given the inputs and labels and we write a program that produces the labels from inputs. For clustering models, a form of unsupervised learning, we only have x_i values, without explicit labels y_i and we want to produce meaningful labels which can serve to cluster the inputs.

13. 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. K-means clusters are formed by the intersection of half-spaces, which are also convex sets themselves. But KNN uses a distance function to find the most suitable label for a data point, which does not guarantee convexity.