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

Important: Submission Format [5 points]

Please make sure to follow the submission instructions posted on the course website. We will deduct marks if the submission format is incorrect, or if you're not using LaTeX and your handwriting is *at all* difficult to read – at least these 5 points, more for egregious issues. Compared to assignment 1, your name and student number are no longer necessary (though it's not a bad idea to include them just in case, especially if you're doing the assignment with a partner).

1 K-Nearest Neighbours [15 points]

In the *citiesSmall* dataset, nearby points tend to receive the same class label because they are part of the same U.S. state. For this problem, perhaps a k-nearest neighbours classifier might be a better choice than a decision tree. The file *knn.py* has implemented the training function for a k-nearest neighbour classifier (which is to just memorize the data).

Fill in the `predict` function in `knn.py` so that the model file implements the k-nearest neighbour prediction rule. You should use Euclidean distance, and may find numpy's `sort` and/or `argsort` functions useful. You can also use `utils.euclidean_dist_squared`, which computes the squared Euclidean distances between all pairs of points in two matrices.

1. Write the `predict` function. Submit this code. [5 points]

   Answer:

```
    def predict(self, X_hat):

        n_hat, d_hat = X_hat.shape

        distance = utils.euclidean_dist_squared(self.X, X_hat)

        closestIndices = np.argsort(distance, axis=0)

        trainI, testI = closestIndices.shape

        print(closestIndices.shape)

        y_hat = np.zeros(n_hat, dtype=int)
        for j in range(testI):      #iterating through each of the points
            kNN = np.zeros(self.k)
            for i in range(self.k):
                kNN[i] = self.y[closestIndices[i][j]]
            mostCommonLabel = utils.mode(kNN)
            y_hat[j] = mostCommonLabel

        return y_hat
```

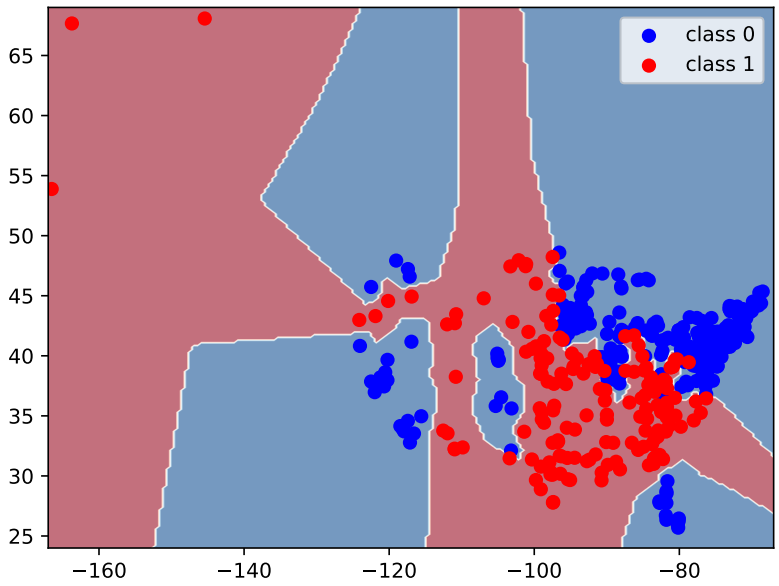
2. Report the training and test error obtained on the *citiesSmall* dataset for $k = 1$, $k = 3$, and $k = 10$. *Optionally*, try running a decision tree on this same train/test split; which gets better test accuracy? [4 points]

   Answer:
   $k = 1$: Training error - 0.0, test error - 0.0645
   $k = 3$: Training error - 0.0275, test error - 0.066
   $k = 10$: Training error - 0.0725, test error - 0.097

   references: `https://stats.stackexchange.com/questions/367010/training-error-in-knn-classifier-when-k-367015`

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 closest neighbour to a training data point is itself, so you will always choose the correct label.

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 choose the k that would give me the smallest cross-validation error.

2 Picking k in kNN [15 points]

The file `data/ccdata.pkl` contains a subset of Statistics Canada's 2019 Survey of Financial Security; we're predicting whether a family regularly carries credit card debt, based on a bunch of demographic and financial information about them. (You might imagine social science researchers wanting to do something like this if they don't have debt information available – or various companies wanting to do it for less altruistic reasons.) If you're curious what the features are, you can look at the `'feat_descs'` entry in the dataset dictionary.

Anyway, now that we have our kNN algorithm working,[1] let's try choosing k on this data!

1. Remember the golden rule: we don't want to look at the test data when we're picking k. Inside the `q2()` function of `main.py`, implement 10-fold cross-validation, evaluating on the `ks` set there (1, 5, 9, ..., 29), and store the *mean* accuracy across folds for each k into a variable named `cv_accs`.

   Specifically, make sure you test on the first 10% of the data after training on the remaining 90%, then test on 10% to 20% after training on the remainder, etc – don't shuffle (so your results are consistent with ours; the data is already in random order). Implement this yourself, don't use scikit-learn or any other existing implementation of splitting. There are lots of ways you could do this, but one reasonably convenient way is to create a numpy "mask" array, maybe using `np.ones(n, dtype=bool)` for an all-`True` array of length `n`, and then setting the relevant entries to `False`. It also might be helpful to know that `~ary` flips a boolean array (`True` to `False` and vice-versa).

   Submit this code, following the general submission instructions to include your code in your results file. [5 points]

   Answer:

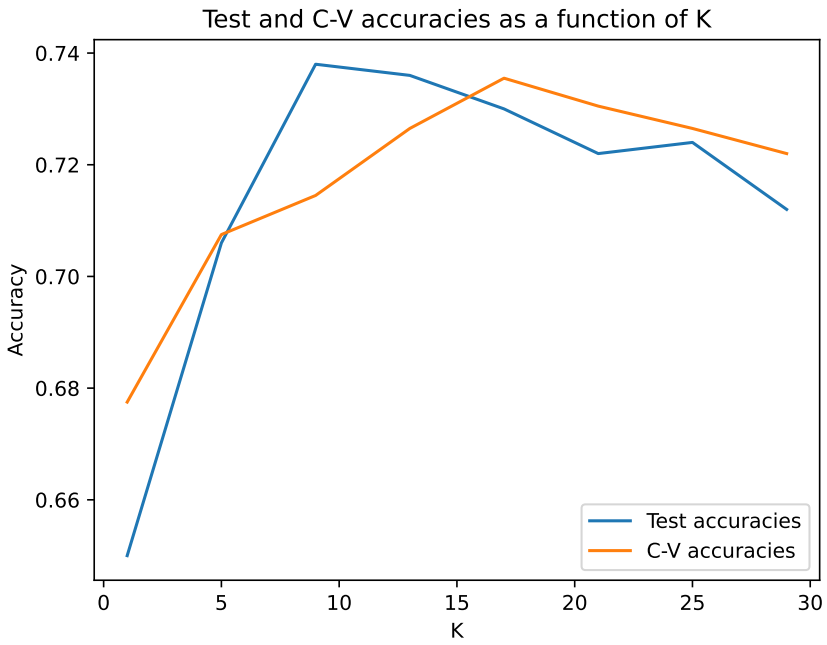
```
"""YOUR CODE HERE FOR Q2"""
n, d = X.shape
mask = np.zeros(n, dtype=bool)
num_examples = int(n/10)
cv_accs = np.zeros(len(ks))
mask_start = 0

for i in range(len(ks)):
    model = KNN(ks[i])
    curr_k_accs = np.zeros(10)
    for j in range(10):
        for k in range(mask_start, mask_start+num_examples):
            # creating the mask
            mask[k] = ~mask[k]
        # resetting the starting position of where to flip the bits
        mask_start = mask_start + num_examples

        # applying the mask
        X_cv_test = X[mask]
        y_cv_test = y[mask]
        mask = ~mask
        X_cv_train = X[mask]
        y_cv_train = y[mask]
```

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

```
            # resetting the mask
            mask = np.zeros(n, dtype=bool)

            # training
            model.fit(X_cv_train, y_cv_train)
            # predicting
            y_cv_pred = model.predict(X_cv_test)

            # computing error and accuracy
            error = np.mean(y_cv_pred != y_cv_test)
            curr_k_accs[j] = 1-error

        cv_accs[i] = np.mean(curr_k_accs)

        # resetting start position for next iteration of k
        mask_start = 0

    print(cv_accs)
```

2. The point of cross-validation is to get a sense of what the test accuracy for a particular value of k would be. Implement, similarly to the code you wrote for question 1.2, a loop to compute the test accuracy for each value of k above. Submit a plot of the cross-validation and test accuracies as a function of k. Make sure your plot has axis labels and a legend. [5 points]

   Answer:

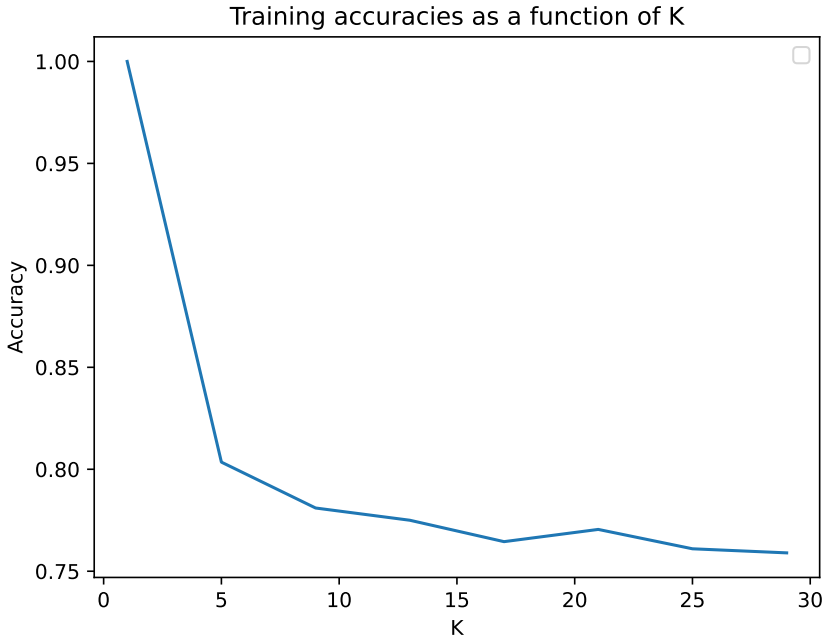


3. Which k would cross-validation choose in this case? Which k has the best test accuracy? Would the cross-validation k do okay (qualitatively) in terms of test accuracy? [2 points]

   Answer: Cross-validation would choose $k = 17$. $k = 9$ has the best test accuracy. The k chosen by cross-validation wouldn't do too bad in terms of test accuracy.

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

   Answer: The best k for training accuracy ($k = 1$) would have the worst performance in terms of test accuracy.



3 Naïve Bayes [17 points]

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

3.1 Naïve Bayes by Hand [5 points]

Consider the dataset below, which has 10 training examples and 3 features:

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

The feature in the first column is <your name> (whether the e-mail contained your name), in the second column is "lottery" (whether the e-mail contained this word), and the third column is "Venmo" (whether the e-mail contained this word). Suppose you believe that a naive Bayes model would be appropriate for this dataset, and you want to classify the following test example:

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

3.1.1 Prior probabilities [1 points]

Compute the estimates of the class prior probabilities, which I also called the "baseline spam-ness" in class. (you don't need to show any work):

- Pr(spam).

  Answer: $\frac{7}{10}$
- Pr(not spam).

  Answer: $\frac{3}{10}$

3.1.2 Conditional probabilities [1 points]

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

- Pr(<your name> $= 1 \mid$ spam).

  Answer: $\frac{2}{7}$
- Pr(lottery $= 1 \mid$ spam).

  Answer: $\frac{5}{7}$
- Pr(Venmo $= 0 \mid$ spam).

  Answer: $\frac{3}{7}$
- Pr(<your name> $= 1 \mid$ not spam).

  Answer: $\frac{2}{3}$

- Pr(lottery = 1 | not spam).

  Answer: $\frac{1}{3}$

- Pr(Venmo = 0 | not spam).

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

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: Probability of spam:
$P(\text{spam}|\text{<your name>} = 1,\ \text{lottery} = 1,\ \text{venmo} = 0) = \frac{P(\text{<your name>=1, lottery=1, venmo=0|spam})*P(\text{spam})}{P(\text{<your name>=1, lottery=1, venmo=0})}$
Probability of not-spam (ham):
$P(\text{not-spam}|\text{<your name>} = 1,\ \text{lottery} = 1,\ \text{venmo} = 0) =$
$\frac{P(\text{<your name>=1, lottery=1, venmo=0|not-spam})*P(\text{not-spam})}{P(\text{<your name>=1, lottery=1, venmo=0})}$
Both of them are using the same denominator, so we only need to compare numerators. Using Naive Bayes, the numerators become:
for spam:
$P(\text{<your name>} = 1,\ \text{lottery} = 1,\ \text{venmo} = 0|\text{spam}) * P(\text{spam} = 1) =$
$P(\text{<your name>} = 1|\text{spam}) * P(\text{lottery} = 1|\text{spam}) * P(\text{venmo} = 0|\text{spam}) * P(\text{spam} = 1)$
For not spam:
$P(\text{<your name>} = 1,\ \text{lottery} = 1,\ \text{venmo} = 0|\text{not-spam}) * P(\text{not-spam} = 1) =$
$P(\text{<your name>} = 1|\text{not-spam}) * P(\text{lottery} = 1|\text{not-spam}) * P(\text{venmo} = 0|\text{not-spam}) * P(\text{not-spam} = 1)$
These two become (for spam):
$\frac{2}{7} * \frac{5}{7} * \frac{3}{7} * \frac{7}{10} = \frac{3}{49}$
for not spam:
$\frac{2}{3} * \frac{1}{3} * \frac{3}{3} * \frac{3}{10} = \frac{1}{15}$

Since $\frac{1}{15} > \frac{3}{49}$, the most likely label would be "not-spam" for our test example.

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:

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

3.2 Exploring Bag-of-Words [2 points]

If you run `python main.py -q 3.2`, it will load the following dataset:

1. `X`: A binary matrix. Each row corresponds to a newsgroup post, and each column corresponds to whether a particular word was used in the post. A value of 1 means that the word occured in the post.
2. `wordlist`: The set of words that correspond to each column.
3. `y`: A vector with values 0 through 3, with the value corresponding to the newsgroup that the post came from.
4. `groupnames`: The names of the four newsgroups.
5. `Xvalidate` and `yvalidate`: the word lists and newsgroup labels for additional newsgroup posts.

Answer the following:

1. Which word corresponds to column 73 of X? (This is index 72 in Python.)

   Answer: question

2. Which words are present in training example 803 (Python index 802)?

   Answer: case,children,health,help,problem,program

3. Which newsgroup name does training example 803 come from?

   Answer: talk

3.3 Naïve Bayes Implementation [4 points]

If you run `python main.py -q 3.3` it will load the newsgroups dataset, fit a basic naive Bayes model and report the validation error.

The `predict()` function of the naive Bayes classifier is already implemented. However, in `fit()` the calculation of the variable `p_xy` is incorrect (right now, it just sets all values to 1/2). Modify this function so that `p_xy` correctly computes the conditional probabilities of these values based on the frequencies in the data set. Submit your code. Report the training and validation errors that you obtain.

Answer: Training Error: 0.200
Validaiton 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

        """YOUR CODE HERE FOR Q3.3"""
        #raise NotImplementedError()

        p_xy = np.zeros((d, k))

        for i in range(4):
            numurator = X[np.where(y == i)[0]].sum(axis=0)
            denominator = counts[i]
            p_xy[:,i] =  numurator/denominator
```

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

  Answer:

```
 def fit(self, X, y):
"""YOUR CODE FOR Q3.4"""
#raise NotImplementedError()

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

p_xy = np.zeros((d, k))

for i in range(4):
    numurator = X[np.where(y == i)[0]].sum(axis=0)+ self.beta
    denominator = counts[i] + self.beta*k
    p_xy[:,i] =  numurator/denominator

self.p_y = p_y
self.p_xy = p_xy
self.not_p_xy = 1-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: When Laplace smoothing was used, lower training error and validation error were obtained compare to when Naive Bayes without Laplace Smoothing were used. This is showing the Laplace Smoothing helps against over-fitting.

- 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: Both training error and validation error did not change even if the beta was increased. This is expected because we add values to both denominator and numerator as we calculate the probabilities with Laplace smoothing, so it balances out.

3.5 Runtime of Naïve Bayes for Discrete Data [2 points]

For a given training example i, the predict function in the provided code computes the quantity

$$p(y_i \mid x_i) \propto p(y_i) \prod_{j=1}^{d} p(x_{ij} \mid y_i),$$

for each class y_i (and where the proportionality constant is not relevant). For many problems, a lot of the $p(x_{ij} \mid y_i)$ values may be very small. This can cause the above product to underflow. The standard fix for this is to compute the logarithm of this quantity and use that $\log(ab) = \log(a) + \log(b)$,

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

This turns the multiplications into additions and thus typically would not underflow.

Assume you have the following setup:

- The training set has n objects each with d features.
- The test set has t objects with d features.
- Each feature can have up to c discrete values (you can assume $c \leq n$).
- There are k class labels (you can assume $k \leq n$)

You can implement the training phase of a naive Bayes classifier in this setup in $O(nd)$, since you only need to do a constant amount of work for each $X(i, j)$ value. (You do not have to actually implement it in this way for the previous question, but you should think about how this could be done.) What is the cost of classifying t test examples with the model and this way of computing the predictions?

Answer: The cost of classifying t test examples is $O(tdk)$ as it has 3 loops with the constant work. Each one loops over t examples, d features, and k class labels.

4 Random Forests [15 points]

The file `vowels.pkl` contains a supervised learning dataset where we are trying to predict which of the 11 "steady-state" English vowels that a speaker is trying to pronounce.

You are provided with a `RandomStump` class that differs from `DecisionStumpInfoGain` in that it only considers $\lfloor \sqrt{d} \rfloor$ randomly-chosen features.[2] You are also provided with a `RandomTree` class that is exactly the same as `DecisionTree` except that it uses `RandomStump` instead of `DecisionStump` and it takes a bootstrap sample of the data before fitting. In other words, `RandomTree` is the entity we discussed in class, which makes up a random forest.

If you run `python main.py -q 4` it will fit a deep `DecisionTree` using the information gain splitting criterion. You will notice that the model overfits badly.

1. Using the provided code, evaluate the `RandomTree` model of unlimited depth. Why doesn't the random tree model have a training error of 0? [2 points]

   Answer: Random tree model does not have a training error of 0 as a bootstrap sample does not include some training examples

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: The tree will be able to classify the examples perfectly after the certain depth, and it does not split anymore after that.

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]

```
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, max_depth, num_trees):
        self.max_depth = max_depth
        self.num_trees = num_trees


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

    def predict(self, X_hat):

        n, d = X_hat.shape
        tree_prediction = np.zeros((n, self.num_trees))
```

[2] The notation $\lfloor x \rfloor$ means the "floor" of x, or "x rounded down". You can compute this with `np.floor(x)` or `math.floor(x)`.

```
        y_hat = np.zeros(n)

        for i in range(self.num_trees):
             tree_prediction[:,i] = self.random_forest[i].predict(X_hat)

        for i in range(n):
            y_hat[i] = utils.mode(tree_prediction[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]

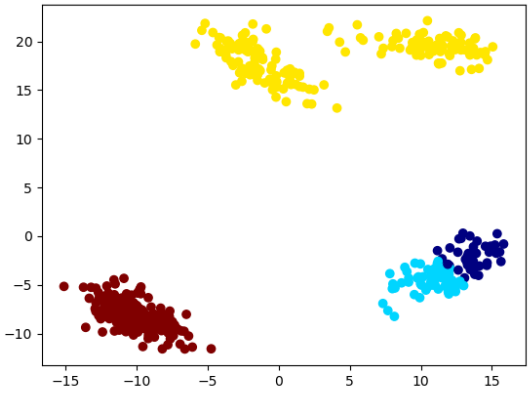
   Answer: Training error of 0 and testing error of 0.192 are obtained. The training error is lower than when other methods were used, which is what I expected. We can see that using random forest would help improving over-fitting.

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]

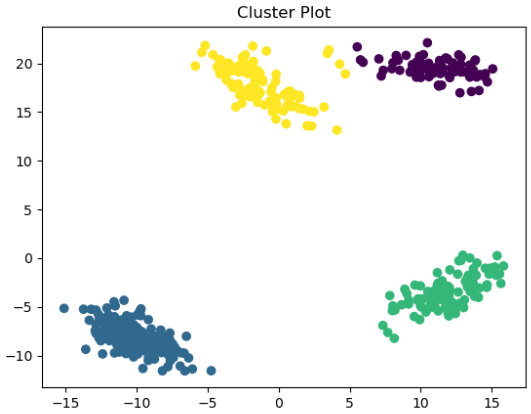
   Answer: All the training examples are in more than half of the trees, and because the depth is infinity, they can be classified perfectly.Therefore, more than half of the trees predict the label perfectly, which leads the random forest having the training error of 0

5 Clustering [15 points]

If you run `python main.py -q 5`, it will load a dataset with two features and a very obvious clustering structure. It will then apply the k-means algorithm with a random initialization. The result of applying the algorithm will thus depend on the randomization, but a typical run might look like this:



(Note that the colours are arbitrary – this is the label switching issue.) But the "correct" clustering (that was used to make the data) is this: ;lkj



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]

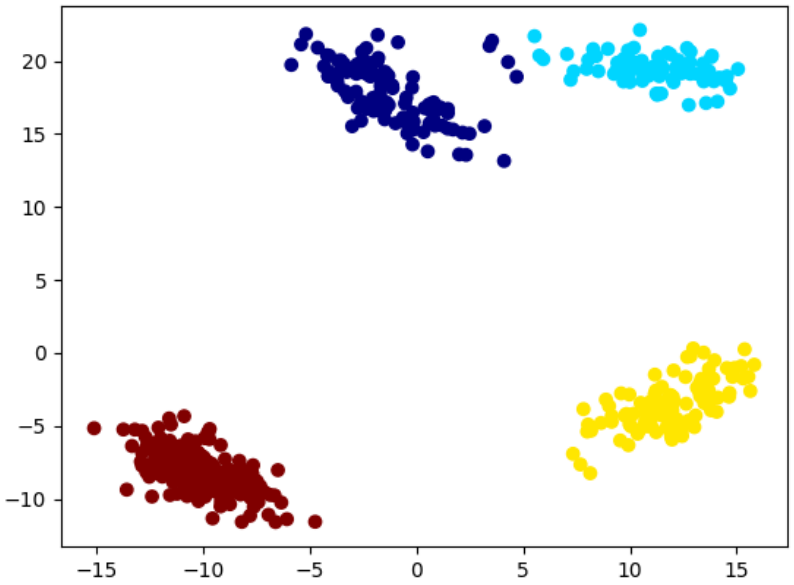
Answer: reference: https://www.youtube.com/watch?v=QXOkPvFM6NU

```
def error(self, X, y, means):
    """YOUR CODE HERE FOR Q5.1"""
    n, d = X.shape
    err = 0

    for i in range(n):
        curr_mean_index = y[i]
        for j in range(d):
            err += (X[i][j] - means[curr_mean_index][j]) ** 2

    return err
```

The error starts off quite high (around 8000), drops quite quickly, and then mostly plateaus (still slowly decreasing).

2. Run k-means 50 times (with $k = 4$) and take the one with the lowest error. Report the lowest error obtained. Visualize the clustering obtained by this model, and submit your plot. [3 points]

   Answer: The lowest error obtained was 3071.4680526538564.



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]

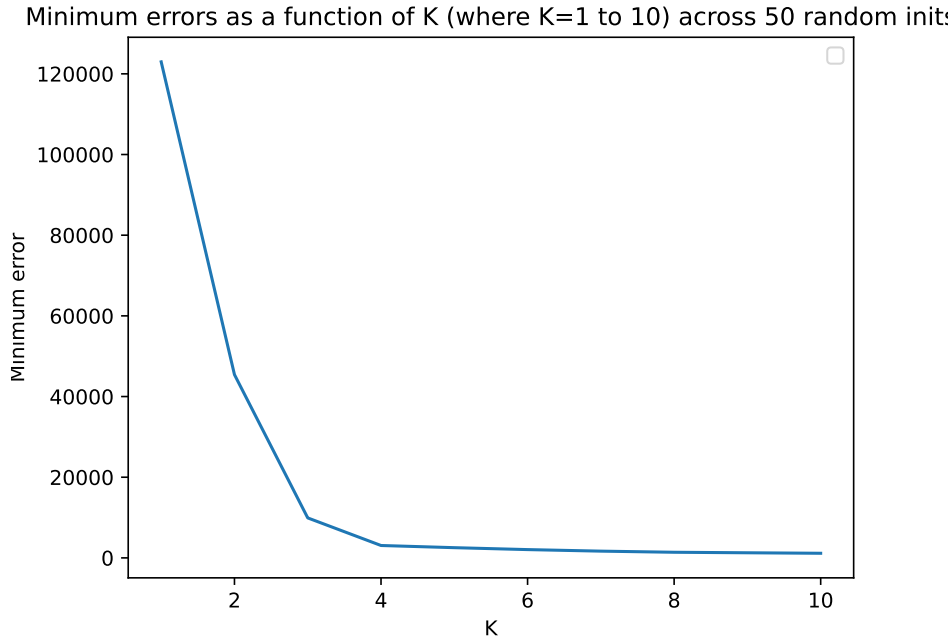
   Answer: Choosing a k that minimizes the error value would mean that the model can fit really well to

that specific dataset, so it may not work well with other data. In other words, we would be overfitting.

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

   Answer: It's still a valid approach because while we're not looking to minimize the error, we are still choosing based off of what the "elbow" point of the error is.

3. Hand in a plot of the minimum error found across 50 random initializations, as a function of k, taking k from 1 to 10. [2 points]



4. The *elbow method* for choosing k consists of looking at the above plot and visually trying to choose the k that makes the sharpest "elbow" (the biggest change in slope). What values of k might be reasonable according to this method? Note: there is not a single correct answer here; it is somewhat open to interpretation and there is a range of reasonable answers. [2 points]

   Answer: I think $k = 3$ or $k = 4$ would be reasonable using this method.

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: The data may not be IID because the probability of certain words being in an email is dependant on the emailing being spam or not-spam. For example, if an email was spam, it has a higher chance of having the words "hot singles in your area" as opposed to an email not being spam.

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

   Answer: Low training error does not always mean that it is a good model. It is possible that it is overfitted.

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

   Answer: Optimization bias becomes large as the training or validation set size n increases.

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

   Answer:
   Advantage: Large k value means that we're validating the model more times, so it we will be able to adjust it better
   Disadvantage: It will take longer

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: COVID test results

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

   Answer: We can ignore $p(x_i)$ because it does not change in any class

7. For each of the three values below in a naive Bayes model, say whether it's better considered as a parameter or a hyper-parameter:

   (a) Our estimate of $p(y_i)$ for some y_i.

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

   (c) The value β in Laplace smoothing.

   Answer:
   (a)parameter
   (b)parameter
   (c)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: Clustering interprets with only input data whereas y_i values are given for the training for supervised learning.

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