CPSC 340 Assignment 2 (see course page for due date)

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 = X_hat.shape[0]
    preds = np.zeros(n)

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

    for i in range(n):
        ## Find k-closest point for each x_i
        k_closest_points = np.argsort(distances[i])[:self.k]
        most_common_label = utils.mode(self.y[k_closest_points])
        preds[i] = most_common_label

    return preds
```

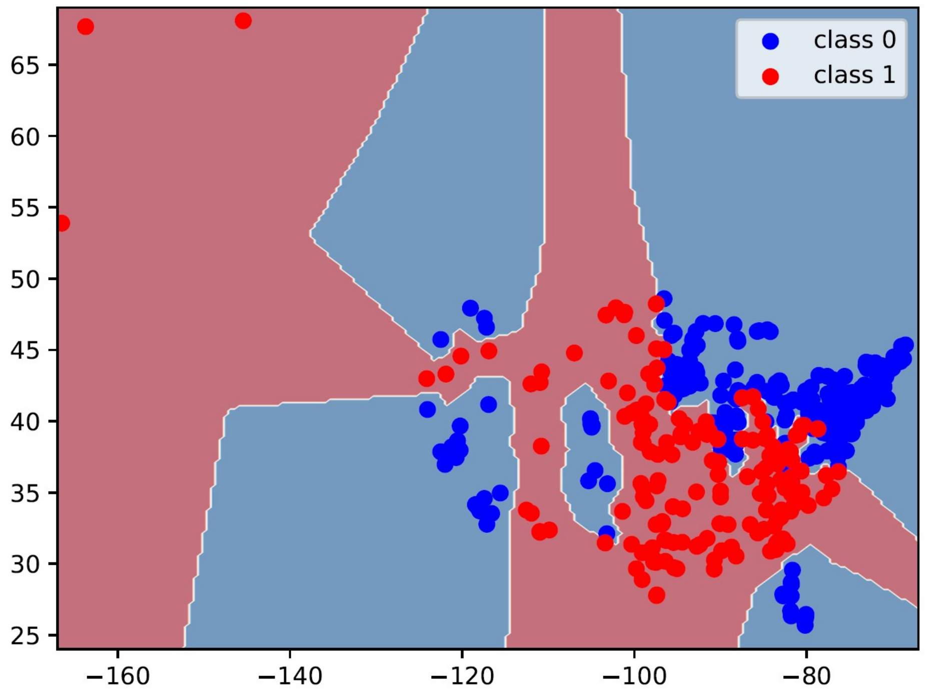
2. Report the training and test error obtained on the *citiesSmall* dataset for $k = 1$, $k = 3$, and $k = 10$. *Optionally*, try running a decision tree on this same train/test split; which gets better test accuracy? [4 points]

   Answer:

k	Training Error	Testing Error
1	0	0.065
3	0.028	0.066
10	0.072	0.097

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: This is because during the fitting phase, we store all the values in the dataset X to use for comparison later. Then, as we calculcate training error, we are running the knn algorithm **on the training data again**. When running knn with $k = 1$, the algorithm will find the single closest point from the data used to fit, but this happens to be the exact same point that we are current trying to classify. This causes 100% accuracy (and thus 0 error) because we are using every points' own label to classify itself.

5. Recall that we want to choose hyper-parameters so that the test error is (hopefully) minimized. How would you choose k? [2 points]

   Answer: The best way would be a method whereby I was able to simulate training and testing cycles, multiple times, without using the "real" testing data. We can do so through validation sets. This way, we are not using testing data, but we are simulating the train-test cycle by "saving" some training data for validation purposes. We can also use cross-validation here as a good technique. The validation error will be plotted compared to k, and the k with an appropriate error will be selected.

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:

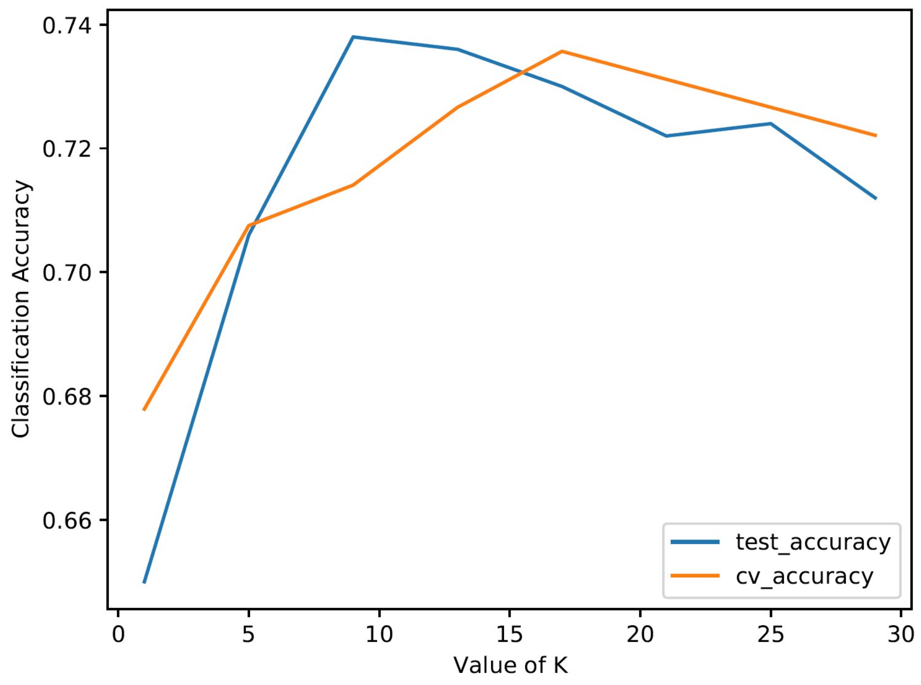
```
n,d = X.shape
ks = list(range(1, 30, 4))

## Variables needed for 10-fold CV
cv_accs = np.zeros(len(ks))
tenth_of_data = int(n/10)

for k in ks:
    model = KNN(k)

    mask = np.ones(n, dtype = bool)

    ## Will increment cv_quantile as we go
    cv_quantile = 0

    each_fold_acc = np.zeros(10)

    while (cv_quantile < 10):
        ## "Cover up" one quartile of the data
        mask[cv_quantile * tenth_of_data : (cv_quantile+1) * tenth_of_data - 1] = 0
        x_train = X[mask,]
        y_train = y[mask]
        x_validation = X[~mask,]
        y_validation = y[~mask]
```

[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)`.

```
            model.fit(x_train, y_train)
            y_validation_pred = model.predict(x_validation)
            accuracy = np.mean(y_validation_pred == y_validation)

            each_fold_acc[cv_quantile] = accuracy

            # Restore the mask before reusing again
            mask[cv_quantile * tenth_of_data : (cv_quantile+1) * tenth_of_data - 1] = 1

            cv_quantile += 1

        avg_accuracy = each_fold_acc.mean()
        cv_accs[ks.index(k)] = avg_accuracy
```

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:

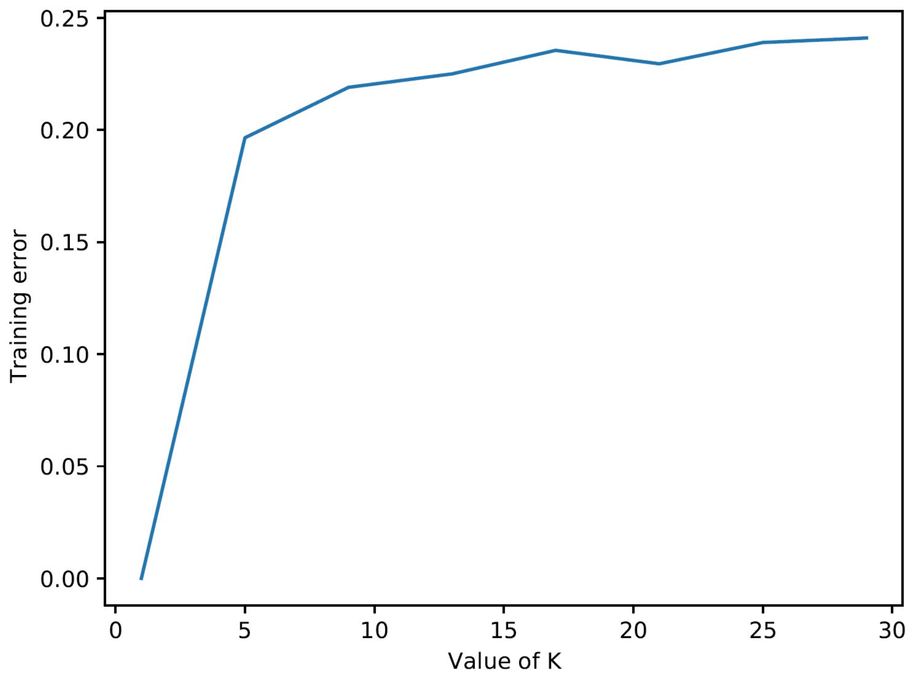


3. Which k would cross-validation choose in this case? Which k has the best test accuracy? Would the cross-validation k do okay (qualitatively) in terms of test accuracy? [2 points]

   Answer: The k that cross validation would choose would be $k = 17$. However, from test accuracy, we can see that $k = 9$ has the best test accuracy. Choosing $k = 17$ from the CV trials would still result in a decently high accuracy, about 0.73, and out of the list of 8 ks to choose from, $k = 17$ would still be top 3.

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:



By the training error, we should pick $k = 1$, since it has a training error of 0. But, this $k = 1$ would do very poorly on the testing error, as we are not ”learning”, rather overfitting and being very affected by the point immediately closest to each point of the dataset.

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:

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

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: 0.6

- Pr(not spam).

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

- Pr(lottery = 1 | spam).

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

- Pr(Venmo = 0 | spam).

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

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

  Answer: $\frac{1}{2}$

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

  Answer: $\frac{1}{2}$

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

  Answer: 1

3.1.3 Prediction [2 points]

Under the naive Bayes model and your estimates of the above probabilities, what is the most likely label for the test example? **(Show your work.)**

Answer: We need to evaluate the probability that $\hat{x}$ is spam and that it is not spam. The conditional probability that $\hat{x}$ is spam is: $Pr(\text{y} = \text{"spam"} \mid \hat{x})$, and the conditional probability that it is *not* spam is $Pr(\text{y} = \text{"not spam"} \mid \hat{x})$

$$Pr(\text{y} = \text{"spam"} \mid \hat{x}) = \frac{Pr(\hat{x} \mid \text{y} = \text{"spam"}) * Pr(\text{y} = \text{"spam"})}{Pr(\hat{x})}$$

$$Pr(\text{y} = \text{"not spam"} \mid \hat{x}) = \frac{Pr(\hat{x} \mid \text{y} = \text{"not spam"}) * Pr(\text{y} = \text{"not spam"})}{Pr(\hat{x})}$$

Since we are directly comparing these two, we do not need the denominator. By the IID assumption for different words,

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

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

By plugging in all the values from question 3.1, we get:

$$Pr(\hat{x} \mid \text{"spam"}) = \frac{1}{6} * \frac{5}{6} * \frac{1}{3} * 0.6 = \frac{1}{36}$$

$$Pr(\hat{x} \mid \text{"not spam"}) = \frac{1}{2} * \frac{1}{2} * 1 * 0.4 = \frac{1}{10}$$

Therefore, by the Naïve Bayes model, the most likely label for the test example 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: To simulate laplace smoothing, you need to add data points that **do not affect the relative probabilities** and yet increases the total count of the data. To do so, we require **two exact opposite datapoints** to be added. One such example is below:

$$X_{add} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad y_{add} = \begin{bmatrix} \text{spam} \\ \text{spam} \\ \text{not spam} \\ \text{not spam} \end{bmatrix}.$$

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: ["cancer", "card", "drive", "help", "number"]

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

   Answer: "talk"

3.3 Naïve Bayes Implementation [4 points]

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

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

Answer:

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

    # Compute the number of class labels
    k = self.num_classes

    # Compute the probability of each class i.e p(y==c), aka "baseline -ness"
    counts = np.bincount(y)
    p_y = counts / n

    # Compute the conditional probabilities i.e.
    # p(x_ij=1 | y_i==c) as p_xy[j, c]
    # p(x_ij=0 | y_i==c) as 1 - p_xy[j, c]

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

    for cls in range(k):
        ## Filter so we operate on every class k separately
```

```
            rows_of_class_k = y == cls
            class_k_domain = X[rows_of_class_k]
            num_examples_in_class_k = class_k_domain.shape[0]

            for ft in range(d):
                ## count how many occurances of 1 in a particular feature, ft
                feature_values_count = np.bincount(class_k_domain[:,ft], minlength=2)
                p_xy[ft, cls] = feature_values_count[1]/num_examples_in_class_k

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

Training error: 0.200 Validation error: 0.188

3.4 Laplace Smoothing Implementation [4 points]

Laplace smoothing is one way to prevent failure cases of Naïve Bayes based on counting. Recall what you know from lecture to implement Laplace smoothing to your Naïve Bayes model.

- Modify the `NaiveBayesLaplace` class provided in `naive_bayes.py` and write its `fit()` method to implement Laplace smoothing. Submit this code.

  Answer: The code here is mostly the same as for question 3.3, just except when counting the number of occurrences of each feature, we add β to the numerator and $\beta * k$ to the denominator.

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

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.ones((d, k))

for cls in range(k):
    rows_of_class_k = y == cls
    class_k_domain = X[rows_of_class_k]
    num_examples_in_class_k = class_k_domain.shape[0]

    for ft in range(d):
        feature_values_count = np.bincount(class_k_domain[:,ft], minlength=2)

        p_xy[ft, cls] = (feature_values_count[1] + self.beta) / (
↪                       num_examples_in_class_k + k*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: Yes, there are some significant differences. Though all the values' relative sizes have not changed with Laplace smoothing (largest probability without Laplace smoothing is still the largest, just of a different value), **their absolute sizes** have changed. Most notably, all the 0 probabilities are now replaced by a very small decimal value. This was the point of Laplace smoothing - to prevent any 0s that naturally occur in the dataset that may lead to the model calculating the probability = 0.

- 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: The probabilities are all around the same, around the range of 0.23 to 0.25. This is because by using $\beta = 1000$, we are adding 1000 to the numerator and $\beta \times k = 4000$ to both numerator and denominator. This very significant value being added causes the probabilities to start to converge to 0.25, which is what we see in the laplace smoothing model.

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 $O(kcd + nd)$, since you only need to do a constant amount of work for each x_{ij} value; usually $kc \ll n$ and so this is $O(nd)$. (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? It's preferable to leave your answer in terms of k and c if relevant.

Answer: For every example we are classifying, we need to calculate $\log p(y_i \mid x_i) = \log p(y_i) + \sum_{j=1}^{d} \log p(x_{ij} \mid y_i)$. Every probability in this equation is already calculated and stored in p_y and $p_x y$. Thus, it will take constant time to retrieve each $p(y_i)$ and $O(d)$ time to retrieve all of the $p(x_{ij} \mid y_i)$s. Then, it takes $O(d)$ time to add them all together.

Then, this above calculation must be done for each class within the dataset, to compare $\log p(y_i \mid x_i)$ for all $i \leq k$, so th $O(d)$ calculation will be done k times.

Once all k calculations are done, we can classify the particular example to one of the classes. Then, we will repeat this for all t test examples. Hence, the total running time is $O(dkt)$.

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: Even if the RandomTree is of infinite depth, it still only has $\sqrt{d}$ number of features. These selection of features may not be enough to fully discern between the different classses, resulting in multiple examples with different classes falling into the same leaf node.

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: This is because we eventually hit our "base". In the DecisionTree code, there is a part that checks if the RandomStump we made at that layer contributes to the classification. This is our "base case" check, as even though there are theoretically infinite splits you could make, eventually not all of them will be useful or be able to classify more than the decision stump before.

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:

    def __init__(self, num_trees, max_depth):
        self.num_trees = num_trees
        self.max_depth = max_depth
        ## To store all the RandomTrees
        self.lofTrees = []

    def fit(self, X, y):

        for i in range(self.num_trees):
            rt = RandomTree(self.max_depth)
            rt.fit(X,y)
            self.lofTrees.append(rt)

    def predict(self, X_hat):

        n = X_hat.shape[0]

        predictions = np.zeros((self.num_trees, n))
```

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

```
        for i in range(self.num_trees):
            predictions[i] = self.lofTrees[i].predict(X_hat)

        all_preds = np.transpose(predictions)

        final_preds = np.zeros(n)
        for i in range(n):
            final_preds[i] = utils.mode(all_preds[i])

        return final_preds
```

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:

	RandomForest	RandomTree	DecisionTree
Training Error:	0	0.174	0
Testing Error:	0.201	0.367	0.458

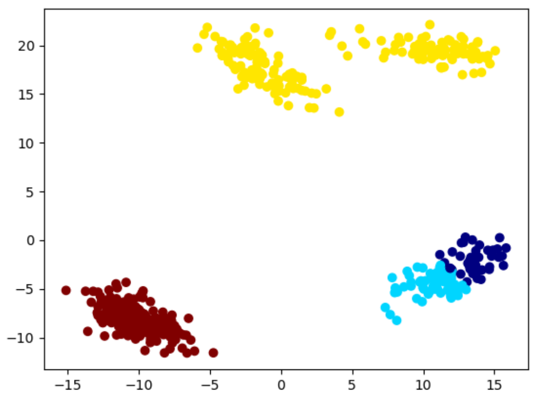
   Though I knew that in class we talked about how RandomForests are the best "out-of-the-box" classifiers, I was still surprised with how well they performed in this experiment. It has a testing error quite significantly less than both a single RandomTree and single DecisionTree. This may be due to the fact that randomly creating trees encourages random and independent error, causing the errors to "cancel each other out" in total. The significant difference between training and testing error for the RandomForest is likely due to the inf depth, causing some over-fitting. However, very impressive regardless.

5. Why does a random forest typically have a training error of 0, even though random trees typically have a training error greater than 0? [3 points]

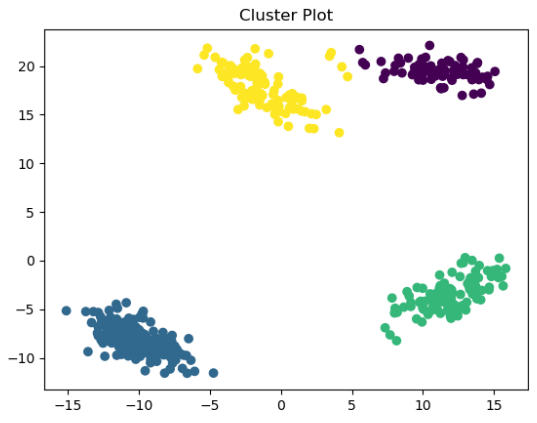
   Answer: With more trees, each with a random assortment of bootstrapped samples and $\sqrt{d}$ features to classify upon, this reduces the chance that any particular initialization will lead the classification astray. Particularly, the limitation of features also allow the model to be less affected by any co-dependencies between the d features that might cause a bias to the model. This also encourage independent errors between each RandomTree, which helps with reducing error when taking the mode of all trees.

5 Clustering [15 points]

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



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



5.1 Selecting among k-means Initializations [7 points]

If you run the demo several times, it will find different clusterings. To select among clusterings for a *fixed* value of k, one strategy is to minimize the sum of squared distances between examples x_i and their means w_{y_i},

$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_2^2 = \sum_{i=1}^{n}\sum_{j=1}^{d}(x_{ij} - w_{y_i j})^2.$$

where y_i is the index of the closest mean to x_i. This is a natural criterion because the steps of k-means alternately optimize this objective function in terms of the w_c and the y_i values.

1. In the `kmeans.py` file, complete the `error()` method. `error()` takes as input the data used in fit (`X`), the indices of each examples' nearest mean (`y`), and the current value of means (`means`). It returns the

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

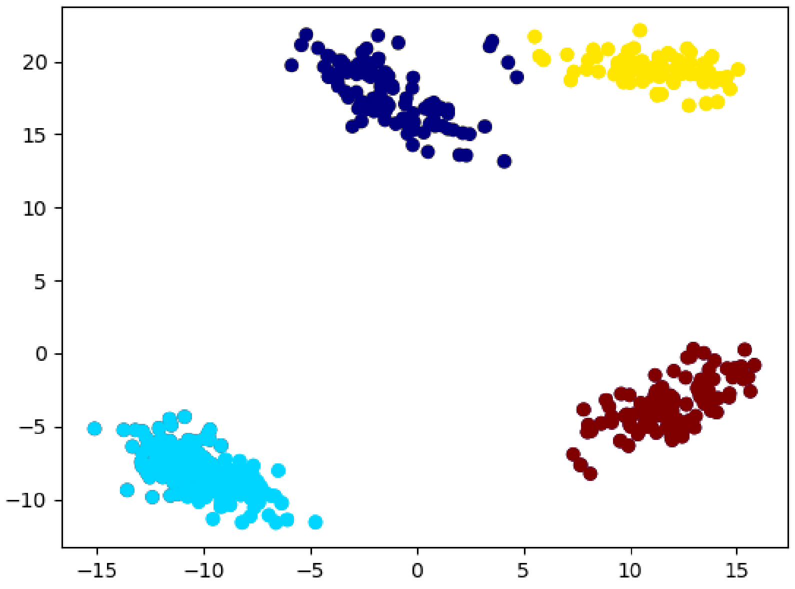
Answer:

```
def error(self, X, y, means):

    distance_matrix = euclidean_dist_squared(X, means)
    n = distance_matrix.shape[0]
    all_errors = np.array([distance_matrix[i][y[i]] for i in range(n)])
    return np.sum(all_errors)
```

As more iterations go by, the error is **strictly non-increasing**.

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 is 3071.46805 (5 d.p.), with the classification plot of:



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 cannot choose k by the minimum error, because if we did, then $k = n$, where $n =$ number of total data points would have zero error. However, we are trying to find generalizations of the code's clusters and thus this is not helpful.

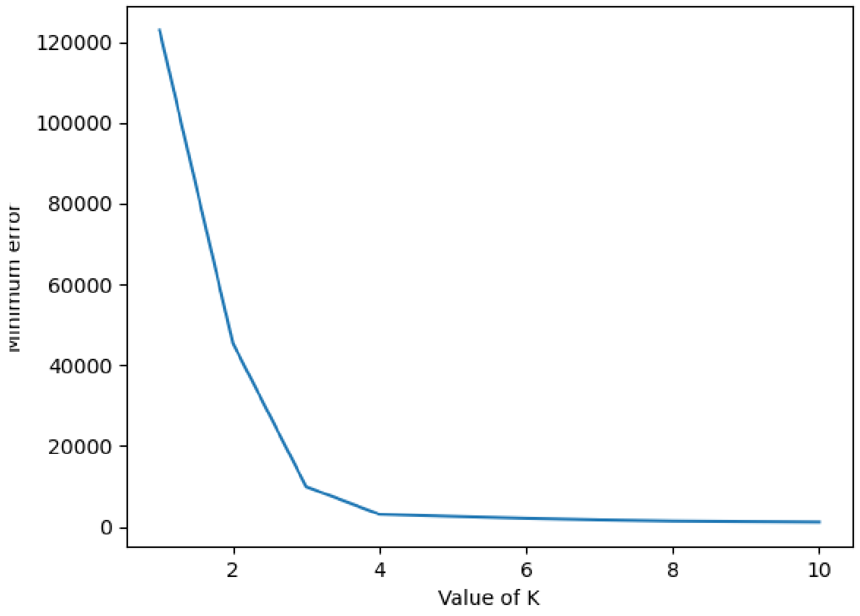
2. Is evaluating the `error` function on validation (or test) data a suitable approach to choosing k? [2 points]

Answer: No. Validation/Test data requires us to have an "answer" to compare our own classification with, however, no labels are given for these datasets, and the **intention** is so that the model can discover good clusters on its own, *without* external labelling.

Let us assume that we somehow **did** find labelled data, claiming that certain points $x_1, x_2, ..., x_n$ should be in clusters $y_1, y_2, ..., y_n$. When we re-run the kmeans algorithm, the specific cluster labels "1", "2" and etc **are arbitrary**. Hence, the labelled cluster for x_1 in the validation/test set may have "3" while the kmeans algorithm outputs "1", but there is no exact mapping between the two labels - another person would have to manually valiate each of them and relabel the examples.

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: I would say that the elbow is around $k = 3$ or potentially 4 as well.

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: Certain words could be inter-dependent, like perhaps the P(”computer”) is higher when the email contains many words like ”math” or ”java” or ”code”, but they are assumed to be independent.

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

   Answer: If we did, we would end up picking a k that has overfit to the training data and hence not very generalizable to non-training data.

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

   Answer: As the training/validation set size n increases, the optimization bias also increases.

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

   Answer: Advantage: Higher k means that there will be more repeated CV trials (k trials to be exact), and that will allow us to probably better approximate testing error.

   Disadvantage: With higher k, the validation set becomes smaller (size $= \frac{n}{k}$), and our tests will become less accurate and could lead to weird results because of an abnormal, small subset of data that was selected for the validation set.

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: One simple example is COVID-19 testing - false positives are certainly safer than false negatives.

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

   Answer: Because $p(x_i)$ is in the denominator of all conditional probabilities that we need to calculate for Naïve Bayes, and so can be ignored.

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

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

   Answer: Parameter: This is a value we use in our classifications and is updated throughout the fitting of the data.

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

   Answer: Parameter: Same reason as above - it is being updated and will depend on the data we are analyzing

   (c) The value β in Laplace smoothing.

   Answer: A hyper-parameter: This value is pre-determined by a human and not (usually) determined by the data. This value also does not change throughout the fitting 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: Supervised learning has labels already provided for inputs x_i, and we are using the "answers" to train our model. However, clustering models (and other unsupervised learning methods) **do not** have labels, and learn through the data itself.

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, definitely not. If we have a few points labeled "0" surrounded by a ring of points labeled "1", then knn (with the right k) can produce regions such that there is a small circular region labeled "0" in the middle, and a ring-shaped region labeled "1" around it. That ring-shaped region is **not convex**.