CPSC 340 Assignment 2

Sadaf Sadeghian (39841333) — Ehsan Soltan Aghai(40218729)

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, d = X_hat.shape

       s_dist_mat = euclidean_dist_squared(X_hat, self.X)

       result = np.zeros(n)
       for test_index in range(n):
           distances = s_dist_mat[test_index, :]
           sorted_args = np.argsort(distances, axis=0)
           neighbours_classes = self.y[sorted_args[:self.k]]
           result[test_index] = utils.mode(neighbours_classes)
       return result
   ```

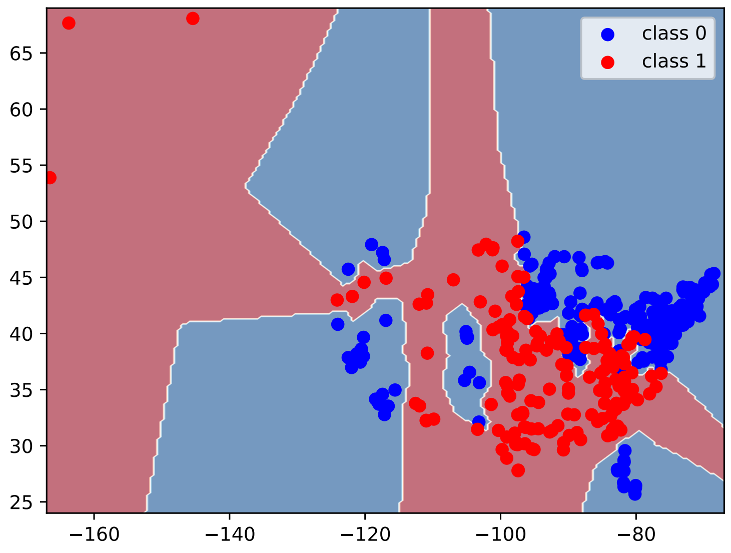
2. Report the training and test error obtained on the *citiesSmall* dataset for $k = 1$, $k = 3$, and $k = 10$. *Optionally*, try running a decision tree on this same train/test split; which gets better test accuracy? [4 points]

   Answer:
   KNN Classifier k = 1 : train error 0.0 , test error 0.0645
   KNN Classifier k = 3 : train error 0.0275, test error 0.066

KNN Classifier k = 10 : train error 0.0725, test error 0.097
Decision Tree Classifier: train error 0.1325, test error 0.175
KNN Classifier K = 1 gets the better test accuracy than decision tree and KNN Classifier K = 3 and 10.

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: When we have a KNN model with k = 1 and we want to classify a new sample, the model finds the closest neighbor to the sample and outputs the label of that closest neighbor as the prediction. So, when we use the train set as the test set, the model selects itself as the closest neighbor and outputs its own label as the prediction. Therefore, it doesn't have any wrong predictions and the error equals zero.

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

   Answer: We can calculate the model error on validation data for different values of k in a specific range and plot the relationship between error and k. Then we can choose k based on the plot, where the validation error is minimized. For finding the validation error we can use K-fold cross-validation so that we don't violate the golden rule.

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:

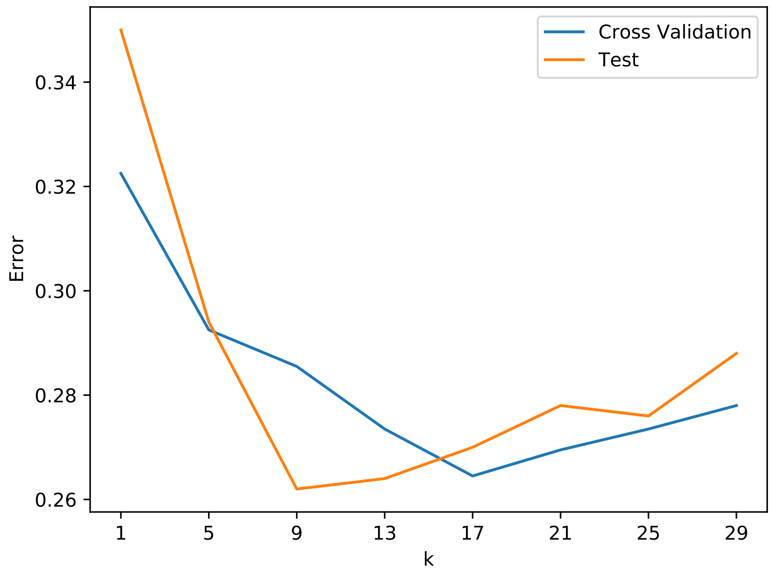
```
dataset = load_dataset("ccdebt.pkl")
X = dataset["X"]
y = dataset["y"]
X_test = dataset["Xtest"]
y_test = dataset["ytest"]

print()

ks = list(range(1, 30, 4))

cv_accs = []
n, d = X.shape
n_fold = 10
fold_size = math.ceil(n/n_fold)

for k in ks:
    KNN_model = KNN(k)

    val_errors = []
    for i in range(n_fold):
        mask = np.ones(n, dtype=bool)
        mask [i * fold_size : min(n, (i+1) * fold_size)] = False
        X_train, y_train = X[mask], y[mask]
        X_val, y_val = X[~mask], y[~mask]
```

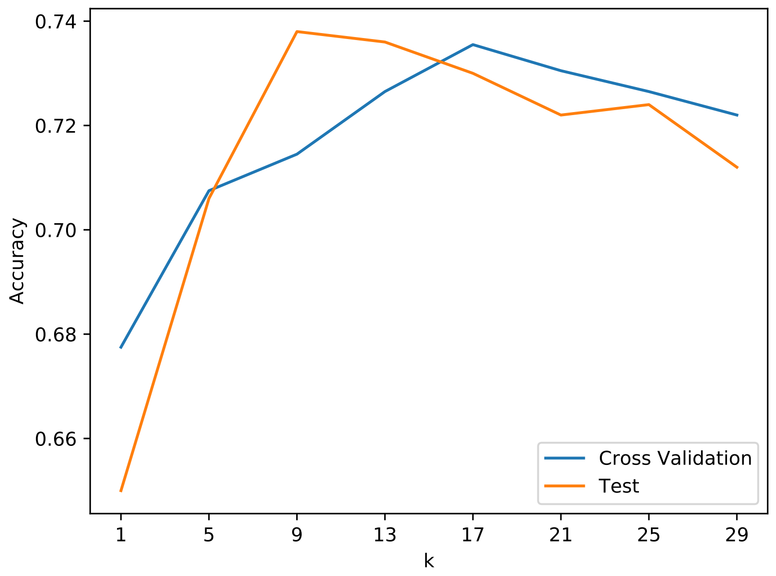
[1] If you haven't finish the code for question 1, or if you'd just prefer a slightly faster implementation, you can use scikit-learn's `KNeighborsClassifier` instead. The `fit` and `predict` methods are the same; the only difference for our purposes is that `KNN(k=3)` becomes `KNeighborsClassifier(n_neighbors=3)`.

```
                KNN_model.fit(X_train, y_train)
                y_predicted = KNN_model.predict(X_val)
                val_error = np.mean(y_predicted != y_val)
                val_errors += [val_error]

            cv_accs += [np.mean(val_errors)]

        print('ks: ', ks)
        print('cv_accs: ', 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:



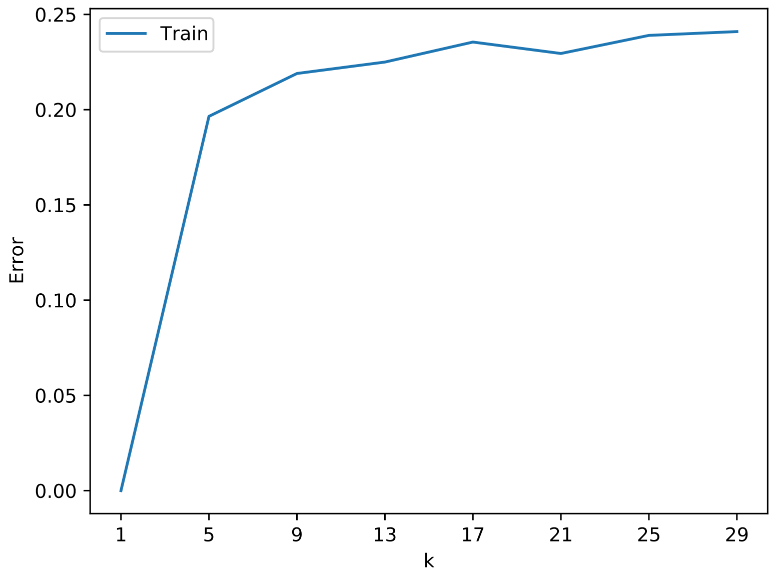


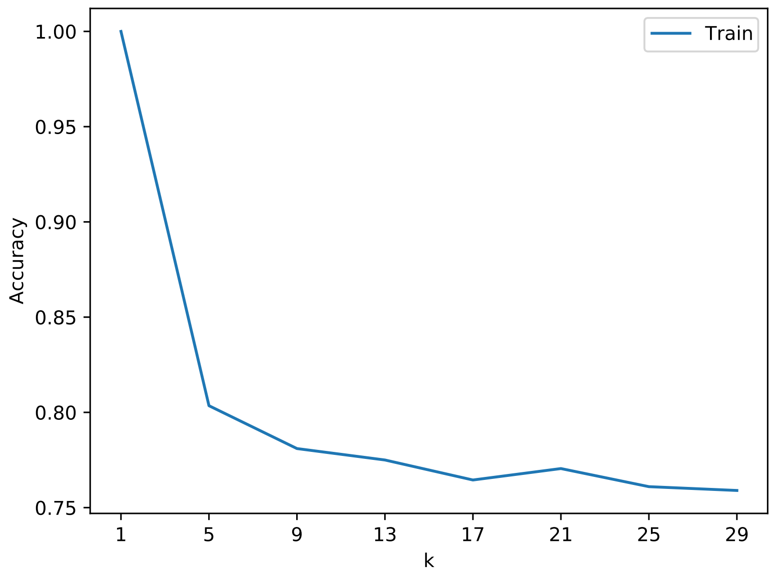
3. Which k would cross-validation choose in this case? Which k has the best test accuracy? Would the cross-validation k do okay (qualitatively) in terms of test accuracy? [2 points]

   Answer:
   The cross-validation chooses k=17.
   k=9 has the best test Accuracy.
   Yes, the test accuracy is roughly the same in k=9 and k=17.(0.738 and 0.73 respectively) The model works relatively good in both cases.

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 train data has the best accuracy in k=1, since the train data and test data are the same and the model selects the same sample as the closest neighbor and always outputs the correct label. We can say that in that case the classifier is over fitted to the data. However, the model with k=1 doesn't have a good accuracy on the test data. (test accuracy is 0.65 in k=1 while its best accuracy is 0.738) This is because the over fitted model cannot generalize well.


Train
Error
k
0.00
0.05
0.10
0.15
0.20
0.25
1
5
9
13
17
21
25
29


Train
Accuracy
k
0.75
0.80
0.85
0.90
0.95
1.00
1
5
9
13
17
21
25
29

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: 0.7

- Pr(not spam).

  Answer: 0.3

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: $\frac{2}{7}$

- Pr(lottery = 1 | spam).

  Answer: $\frac{5}{7}$

- Pr(Venmo = 0 | spam).

  Answer: $\frac{3}{7}$

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

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

- Pr(lottery = 1 | not spam).

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

- Pr(Venmo = 0 | 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: $P(\hat{y}|\hat{x}) = \frac{P(\hat{x}|\hat{y})P(\hat{y})}{P(\hat{x})}$. Since $P(\hat{x})$ is equal for all and given IID assumption, we have:
$P(\hat{y}|\hat{x}) \propto P(\hat{x}|\hat{y})P(\hat{y}) \approx P(\hat{y})\prod_{j=1}^{3} P(\hat{x_j}|\hat{y})$

$P(\hat{y} = spam|\hat{x}) \propto P(\hat{y} = spam)P(name = 1|spam)P(lottery = 1|spam)P(venmo = 0|spam) = \frac{7}{10}\frac{2}{7}\frac{5}{7}\frac{3}{7} = \frac{3}{49} = 0.0612$

$P(\hat{y} = not\ spam|\hat{x}) \propto P(\hat{y} = not\ spam)P(name = 1|not\ spam)P(lottery = 1|not\ spam)P(venmo = 0|not\ spam) = \frac{3}{10}\frac{2}{3}\frac{1}{3}1 = \frac{1}{15} = 0.066$

So, $P(\hat{y} = not\ spam|\hat{x}) > P(\hat{y} = spam|\hat{x})$, not spam is more likely for the 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:

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

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

Here is the python code:

```
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"""
    p_xy = np.zeros((d, k))
    count_xy = np.zeros((d, k))
    for j in range(d):
        for i in range(n):
            if X[i, j]:
                count_xy[j, y[i]] += 1
        for l in range(k):
            p_xy[j, l] = count_xy[j, l]/counts[l]
    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:

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

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

        # 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))
        count_xy = np.zeros((d, k))
        not_p_xy = np.zeros((d, k))
        not_count_xy = np.zeros((d, k))

        for j in range(d):
            for i in range(n):
                if X[i, j]:
                    count_xy[j, y[i]] += 1
                else:
                    not_count_xy[j, y[i]] += 1
            for l in range(k):
                p_xy[j, l] = (count_xy[j, l] + self.beta) / (counts[l] + self.beta * k)
                not_p_xy[j, l] = (not_count_xy[j, l] + self.beta) / (counts[l] + self.beta * k)

        self.p_y = p_y
        self.p_xy = p_xy
        self.not_p_xy = not_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: Naive Bayes model without Laplace smoothing have some zero probability in p_xy which result the total production of probabilities to zero. However, in Naive Bayes model with Laplace smoothing, those zero probabilities are exchanged with very low probabilities (generally very low) based on Laplace smoothing.

- 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: With large β values, probability values which are zero in Naive Bayes without Laplace smoothing, approach $\frac{1}{k}$ in Laplace smoothing, which is close to uniform value across number of features. If we have enough training data, we expect very low probability since there is no sample available in training data for that case.

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: O(tdk)
We should compute tk conditional probabilites $\log p(y_i \mid x_i)$ and for each one we have to go over the probabilities $\log p(x_{ij} \mid y_i)$ which are the d features.

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: In decision tree of unlimited depth, the training error would be zero since we can split enough on every features and makes a new path for different cases of our training data. However, in random tree of unlimited depth we just have $\lfloor\sqrt{d}\rfloor$ randomly-chosen features at each step and it's not possible to build every cases since we can't condition on some of the features.

2. For `RandomTree`, if you set the `max_depth` value to `np.inf`, why do the training functions terminate instead of making an infinite number of splitting rules? [2 points]

   Answer: Because the number of different split modes regarding selected features and available samples is finite. At some point all of the samples available at a random stump could have the same label and it terminates in this case. Also, at some cases the error of mode of y is less than all possible splits.

3. Complete the `RandomForest` class in `random_tree.py`. This class takes in hyperparameters `num_trees` and `max_depth` and fits `num_trees` random trees each with maximum depth `max_depth`. For prediction, have all trees predict and then take the mode. Submit this code. [5 points]

   Answer:

```
class RandomForest:
    num_trees = None
    max_depth = None
    random_trees = None

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

    def fit(self, X, y):
        random_trees = np.empty(self.num_trees, dtype=object)
        for i in range(self.num_trees):
            random_tree = RandomTree(self.max_depth)
            random_tree.fit(X, y)
```

[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)`.

```
            random_trees[i] = random_tree

        self.random_trees = random_trees

    def predict(self, X):
        n, d = X.shape
        y_trees = np.zeros((self.num_trees, n))
        y = np.zeros(n)
        for i in range(self.num_trees):
            y_trees[i, :] = self.random_trees[i].predict(X)

        for i in range(n):
            y[i] = stats.mode(y_trees[:, i].flatten())[0][0]

        return y
```

4. Using 50 trees, and a max depth of ∞, report the training and testing error. Compare this to what we got with a single `DecisionTree` and with a single `RandomTree`. Are the results what you expected? Discuss. [3 points]

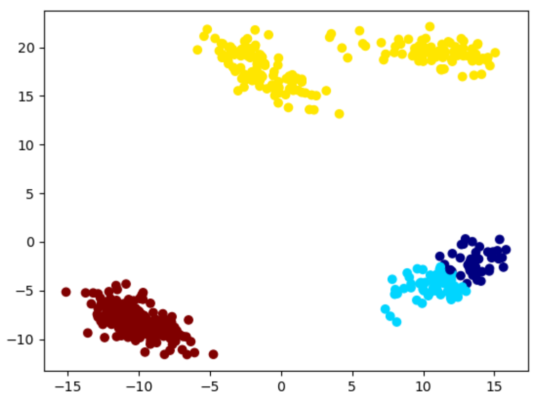
   Answer:
   Decision tree info gain:
    Training error: 0.000
    Testing error: 0.367
   Random tree:
    Training error: 0.178
    Testing error: 0.530
   Random forest:
    Training error: 0.000
    Testing error: 0.170
   Yep, as you see Random Forest has zero training error and low test error comparing to the other two methods. This result happened thanks to taking mode of different random trees with bootstrap sample of the data which reduces the risk of overfitting. Also, random tree has worse error compared to decision tree because it has just a subset of features at each stump.

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]

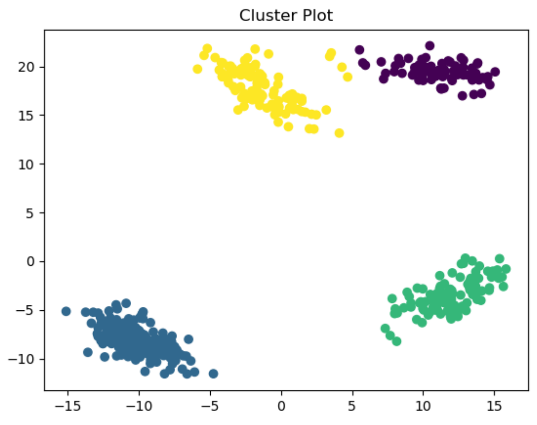
   Answer: Random tree has a low training error but not zero. We fit different random trees with bootstrap sample of the data (all with low training error) and then take mode of different random trees. It result to zero training error since more than half of random trees have true result on a sample, and mode of all random trees is always a right prediction on the training data.

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:



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},

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

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

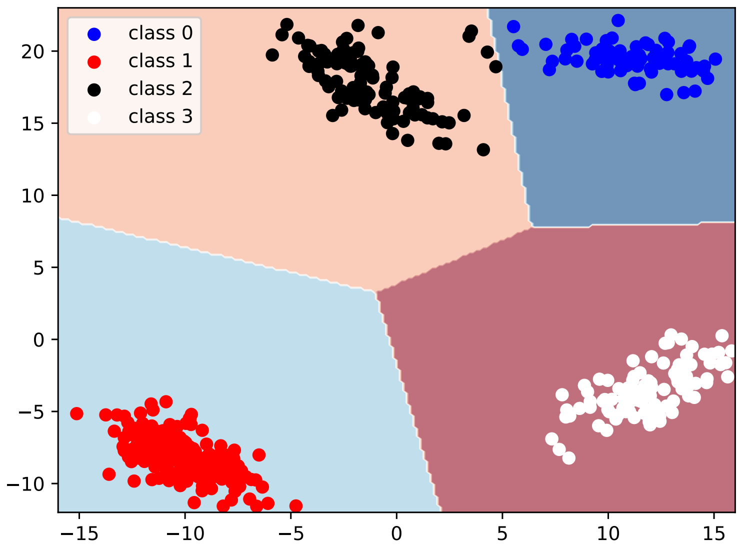
value of this above objective function. Submit this code. What trend do you observe if you print the value of this error after each iteration of the k-means algorithm? [4 points]

Answer: If we print the error value after each iteration, we can observe that it is decreasing.

```
def error(self, X, y, means):
    n, d = X.shape
    selected_means = np.array([means[yy] for yy in y])
    error = np.sum((X - selected_means)**2)
    return error
```

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 is 3071.468052653855.



5.2 Selecting k in k-means [8 points]

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

1. Explain why we should not choose k by taking the value that minimizes the `error` value. [2 points]

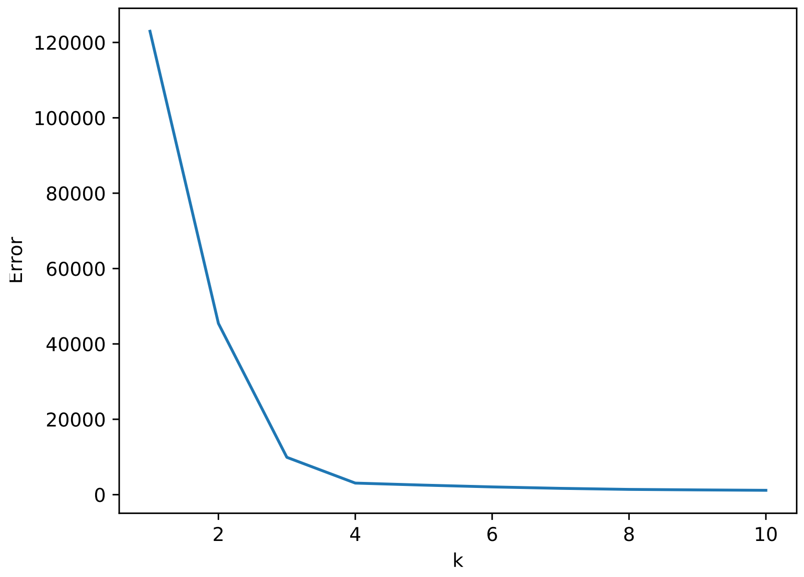
   Answer: Becaue in that case the model will be over-fitted on our train data and doesn't generalize well. So the model's accuracy would be very low on test data. The error decreases with increasing k and it will be zero in k=n, since each point has its own cluster.

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

   Answer: No, since based on the golden rule we should see the test data only once to evaluate our model, and we shouldn't involve the test data in the training or tuning hyper parameters phase, and the test data should be kept unseen.

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: Based on the elbow method and the plot shown above, k = 3 seems to be a reasonable value since in that point we have the biggest slope change

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: Emails are not necessarily independent from each other, and they can be affected by each other like in a conversation or whatever. Also, interests, jobs, engagements (e.g. subscriptions) can change, so emails are not identical and independent necessarily.

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

   Answer: We have used training data to fit parameters, so fitting hyper-parameter on training data is biased and overfits training data badly.

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

   Answer: In parametric models, as the size of training or validation increase, chances of overfitting decrease, so optmiziation bias is less.

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

   Answer: k-fold cross validation with larger k is more accurate, but learning process is more expensive because we compute cross validation score for each depth and we have to train and validate k times for each.

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: A test identifies a food is corrupted/spoiled or not. False Positive is to identify a healthy food as corrupted which has less risk/cost compared to identifying a corrupted food as healthy. So, we would rather False Positive than False Negative.

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

   Answer: We want the label with maximum conditional probability, and marginal probability $p(x_i)$ is the same in all conditional probabilities, so if we ignore it the comparison between conditional probabilities is the same as before.

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

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

   Answer: parameter

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

   Answer: parameter

   (c) The value β in Laplace smoothing.

   Answer: hyper-parameter,

8. Both supervised learning and clustering models take in an input x_i and produce a label y_i. What is the key difference between these types of models?

   Answer: In supervised learning, labels are given and the goal is to predict labels on new data examples. However, in clustering, labels are not given ant the goal is just to cluster data to different groups. There is no test error and label prediction in clustering.

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, it can be either convex or not. The 1.3 plot is an example of non convex areas.