CPSC 340 Assignment 2 (due 2019-01-25 at 11:55pm)

Yihao Zhang, #82626169, u6m1b
Hafsa Binte Zahid, #11668150, i6r0b

Instructions

Rubric: {mechanics:5}

IMPORTANT!!! Before proceeding, please carefully read the general homework instructions at `https://www.cs.ubc.ca/~fwood/CS340/homework/`. The above 5 points are for following the submission instructions. You can ignore the words "mechanics", "reasoning", etc.

We use blue to highlight the deliverables that you must answer/do/submit with the assignment.

1 Training and Testing

If you run `python main.py -q 1`, it will load the *citiesSmall.pkl* data set from Assignment 1. Note that this file contains not only training data, but also test data, `X_test` and `y_test`. After training a depth-2 decision tree with the information gain splitting rule, it will evaluate the performance of the classifier on the test data. With a depth-2 decision tree, the training and test error are fairly close, so the model hasn't overfit much.

1.1 Training and Testing Error Curves

Rubric: {reasoning:2}

Make a plot that contains the training error and testing error as you vary the depth from 1 through 15. How do each of these errors change with the decision tree depth?

Note: it's OK to reuse code from Assignment 1.
Classification Error Rate vs Depth of Tree is shown in following figure.

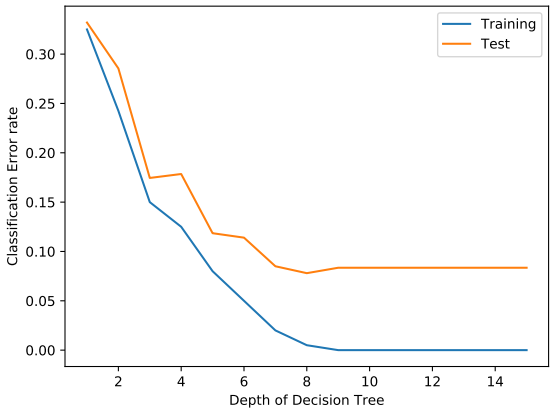


Figure 1: Classification Error Rate vs Depth of Tree

Code for 1.1:

```
elif question == "1.1":
    with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
        dataset = pickle.load(f)

    X, y = dataset["X"], dataset["y"]
    X_test, y_test = dataset["Xtest"], dataset["ytest"]

    depth = np.arange(1, 16)
    tr_error = np.zeros(depth.size)
    te_error = np.zeros(depth.size)

    for i, cur_depth in enumerate(depth):
        model = DecisionTreeClassifier(max_depth=cur_depth,
                                       criterion='entropy',
                                       random_state=1)
        model.fit(X, y)
        y_pred = model.predict(X)
        tr_error[i] = np.mean(y_pred != y)

        y_pred = model.predict(X_test)
        te_error[i] = np.mean(y_pred != y_test)

    plt.plot(depth, tr_error, label="Training")
    plt.plot(depth, te_error, label="Test")
    plt.xlabel("Depth of Decision Tree")
    plt.ylabel("Classification Error rate")
    plt.legend()
    fname = os.path.join("..", "figs", "q1_1.pdf")
    plt.savefig(fname)
```

1.2 Validation Set

Rubric: {reasoning:3}

Suppose that we didn't have an explicit test set available. In this case, we might instead use a *validation* set. Split the training set into two equal-sized parts: use the first $n/2$ examples as a training set and the second $n/2$ examples as a validation set (we're assuming that the examples are already in a random order). What depth of decision tree would we pick to minimize the validation set error? Does the answer change if you switch the training and validation set? How could use more of our data to estimate the depth more reliably?

The answer does change if training and validation set is switched. Because decision stump choice is dependent on the data used in actual training stage. Use the first $n/2$ examples as a training set and the second $n/2$ examples as a validation set gives minimum validation error of 0.1 at depth of 8.

Switching training and validation set gives minimum training error of 0.085 at depth of 6.

To estimate the depth more reliably, cross-validation could be used to separate dataset into multiple folds and obtain depth from average minimum error.

Code for Q1.2:

```
elif question == '1.2':
    with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
        dataset = pickle.load(f)

    X, y = dataset["X"], dataset["y"]
    n, d = X.shape

    depth = np.arange(1, 16)
    tr_error = np.ones(depth.size)
    te_error = np.ones(depth.size)

    in_order = True

    if in_order:
        X_train, X_val, y_train, y_val = \
            train_test_split(X,
                             y,
                             test_size=0.5,
                             shuffle=False)
    else:
        X_val, X_train, y_val, y_train = \
            train_test_split(X,
                             y,
                             test_size=0.5,
                             shuffle=False)

    for i, cur_depth in enumerate(depth):
        model = DecisionTreeClassifier(max_depth=cur_depth,
                                       criterion='entropy',
                                       random_state=1)
        model.fit(X_train, y_train)

        y_pred = model.predict(X_val)
        te_error[i] = np.mean(y_pred != y_val)

    print("minimum training error of {} at depth of {}"
          .format(np.min(te_error), np.argmin(te_error) + 1))
```

2 Naive Bayes

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

2.1 Naive Bayes by Hand

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

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

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

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

2.1.1 Prior probabilities

Rubric: {reasoning:1} Compute the estimates of the class prior probabilities (you don't need to show any work):

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

$p(spam) = \frac{3}{5}$
$p(notspam) = \frac{2}{5}$

2.1.2 Conditional probabilities

Rubric: {reasoning:1}

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

- $p(\text{<your name>} = 1 \mid \text{spam}) = \frac{1}{6}$.
- $p(\text{pharmaceutical} = 1 \mid \text{spam}) = \frac{5}{6}$.
- $p(\text{PayPal} = 0 \mid \text{spam}) = \frac{1}{3}$.
- $p(\text{<your name>} = 1 \mid \text{not spam}) = 1$.

- $p(\text{pharmaceutical} = 1 \mid \text{not spam}) = \frac{1}{4}$.
- $p(\text{PayPal} = 0 \mid \text{not spam}) = \frac{1}{4}$.

2.1.3 Prediction

Rubric: {reasoning:1}

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.)

$$\begin{aligned} p(\text{not spam} \mid x_1 = 1, x_2 = 1, x_3 = 0) &= p(x_1 = 1 \mid \text{not spam}) \times p(x_2 = 1 \mid \text{not spam}) \times p(x_3 = 0 \mid \text{not spam}) \times p(\text{not spam}) \\ &= 1 \times \frac{1}{4} \times \frac{1}{4} \times \frac{2}{5} \\ &= \frac{1}{40} \end{aligned}$$

$$\begin{aligned} p(\text{spam} \mid x_1 = 1, x_2 = 1, x_3 = 0) &= p(x_1 = 1 \mid \text{spam}) \times p(x_2 = 1 \mid \text{spam}) \times p(x_3 = 0 \mid \text{spam}) \times p(\text{spam}) \\ &= \frac{1}{6} \times \frac{5}{6} \times \frac{1}{3} \times \frac{3}{5} \\ &= \frac{5}{36} \end{aligned}$$

Since $\frac{5}{36}$ is larger than $\frac{1}{40}$, label will be spam.

2.1.4 Laplace smoothing

Rubric: {reasoning:2}

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 that we could add to the original training set that would make the basic estimates give us the estimates with Laplace smoothing (in other words give a set of extra training examples that, if they were included in the training set and we didn't use 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.

Laplace smoothing is equivalent to augmenting the training set with extra counts. $\beta = 1$ gives the same smoothing as adding one fake "spam" example that contains a feature for each feature, and one fake "not spam" example that contains a feature for each feature. So, we add $\beta * k$ extra spam and not spam examples
A set of extra training examples that, when added to the training set, produce the same effect:
x_1: [name=1, pharm=0, paypal=1]; y_1: spam
x_2: [name=0, pharm=1, paypal=1]; y_2: spam
x_3: [name=0, pharm=0, paypal=0]; y_3: spam
x_4: [name=1, pharm=0, paypal=1]; y_4: not spam
x_5: [name=0, pharm=1, paypal=1]; y_5: not spam
x_6: [name=0, pharm=0, paypal=0]; y_6: not spam

2.2 Bag of Words

Rubric: {reasoning:3}

If you run `python main.py -q 2.2`, 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 51 of X? (This is column 50 in Python.)
   Column 51 of X is the word 'lunar'

2. Which words are present in training example 501?
   Training example 501 has words: car, fact, gun, video

3. Which newsgroup name does training example 501 come from?
   Training example 501 comes from newsgroup talk.*

Code for Q2.2:

```
elif question == '2.2':
    dataset = load_dataset("newsgroups.pkl")

    X = dataset["X"]
    y = dataset["y"]
    X_valid = dataset["Xvalidate"]
    y_valid = dataset["yvalidate"]
    groupnames = dataset["groupnames"]
    wordlist = dataset["wordlist"]

    print("Column 51 of X is the word '{}'".format(wordlist[50]))
    words = ', '.join(wordlist[np.where(X[500, ] == 1)])
    print("Training example 501 has words: {}".format(words))

    print("Training example 501 comes from newsgroup {}"
          .format(groupnames[y[500]]))
```

2.3 Naive Bayes Implementation

Rubric: {code:5}

If you run `python main.py -q 2.3` 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 and the validation error that you obtain. Also, compare your validation error to what you obtain with scikit-learn's implementation, `BernoulliNB`.
Naive Bayes (ours) validation error: 0.188. BernoulliNB validation error: 0.187. Our Naive Bayes implementation is slightly worse than BernoulliNB.

Code for Q2.3:

```
    def fit(self, X, y):
        N, D = X.shape

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

        # Compute the probability of each class i.e p(y==c)
        counts = np.bincount(y)
        p_y = counts / N

        # Compute the conditional probabilities i.e.
        # p(x(i,j)=1 | y(i)==c) as p_xy
        # p(x(i,j)=0 | y(i)==c) as p_xy
        # p_xy = 0.5 * np.ones((D, C))
        # TODO: replace the above line with the proper code
        p_xy = np.zeros((D, 4))

        feature_id = 0
        for row in X.T:
            for example_id, example_val in enumerate(row):
                if example_val == 1:
                    p_xy[feature_id, y[example_id]] += 1
            feature_id += 1

        p_xy[:, 0] /= counts[0]
        p_xy[:, 1] /= counts[1]
        p_xy[:, 2] /= counts[2]
        p_xy[:, 3] /= counts[3]

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

Code for BernoulliNB:

```
        model = BernoulliNB()
        model.fit(X, y)
        y_pred = model.predict(X_valid)
        v_error = np.mean(y_pred != y_valid)
        print("BernoulliNB validation error: %.3f" % v_error)
```

2.4 Runtime of Naive Bayes for Discrete Data

Rubric: {reasoning:3}

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 $\log(ab) = \log(a) + \log(b)$,

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

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?
We have t test examples, so O(classifying t) will be O(t*classifying one)
To classify one point, we need to run $p(y_i \mid x_i)$ $k-1$ times because y_i can take on k values.
Every time we run $p(y_i \mid x_i)$, we call the sum for $p(x_{ij} \mid y_i)$ d times.
Since x_i is the given test example, it has a definite valie for its features and we don't need to iterate over all c values that these features can take up.
So in total, the complexity of running this is O(tdk)

3 K-Nearest Neighbours

Rubric: {code:3, reasoning:4}

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 Euclidean distance, and may 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.

```
def predict(self, Xtest):
    y_hat = []
    (t, d) = Xtest.shape  # t*d
    dist = utils.euclidean_dist_squared(self.X, Xtest)  # n*t
```

```
        # For each row in test
        for i in range(0, t):
            # each col in N*t corresponds one test vector
            row = sorted(dist[:, i])
            row_values = np.array(row[0:self.k])

            row_indices = []
            for j in range(0, self.k):
                row_indices.append(np.where(dist[:, i] == row_values[j]))

            y_kneighbors = []
            for m in row_indices:
                y_kneighbors.append(self.y[m])

            y_hat.append(utils.mode(y_kneighbors))

        return y_hat
```

2. Report the training and test error obtained on the *citiesSmall* dataset for $k = 1$, $k = 3$, and $k = 10$. How do these numbers compare to what you got with the decision tree?
   KNN with k=1 gets training error of 0.0
   KNN with k=1 gets validation error of 0.0645
   KNN with k=3 gets training error of 0.0275
   KNN with k=3 gets validation error of 0.066
   KNN with k=10 gets training error of 0.0725
   KNN with k=10 gets validation error of 0.097
   With decision tree, test error increases with the increase of k. With k = 1, we get the minimum validation error of 0.0645 and it is lower than the value we got from decision tree. Training error goes to 0 after some depth of the tree. However, for KNN, with k increasing, there is no guarantee for training error to be 0 or even decreasing.
   Code for Q3.2:

```
elif question == '3':
    with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
        dataset = pickle.load(f)

    X = dataset['X']
    y = dataset['y']
    Xtest = dataset['Xtest']
    ytest = dataset['ytest']

    k = [1, 3, 10]
    for k_test in k:
        model = KNN(k_test)
        model.fit(X, y)
        y_pred = model.predict(X)
        y_error = np.mean(y_pred != y)
        print("KNN with k={} gets training error of {}"
              .format(k_test, y_error))

        y_pred = model.predict(Xtest)
        y_error = np.mean(y_pred != ytest)
        print("KNN with k={} gets validation error of {}"
```

```
.format(k_test, y_error))
```

3. Hand in the plot generated by `utils.plotClassifier` on the *citiesSmall* dataset for $k = 1$, using both your implementation of KNN and the KNeighborsClassifier from scikit-learn.
   Training and test classification for our KNN:

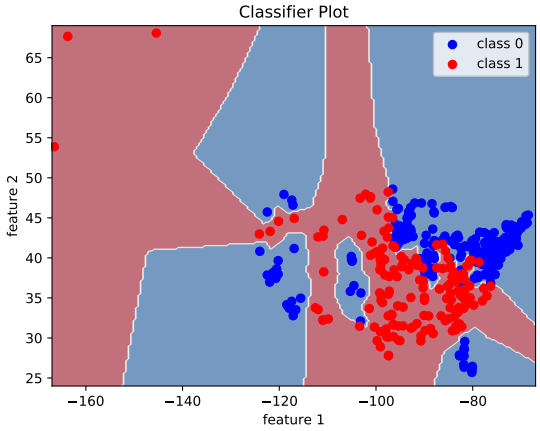


Figure 2: Training classification of our KNN model

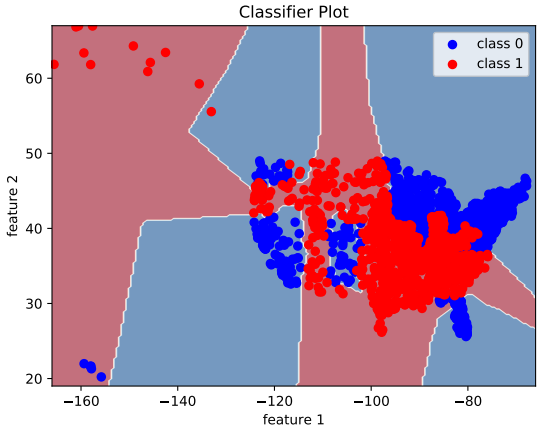


Figure 3: Testing classification of our KNN model

Training and test classification of KNeighborsClassifier model:

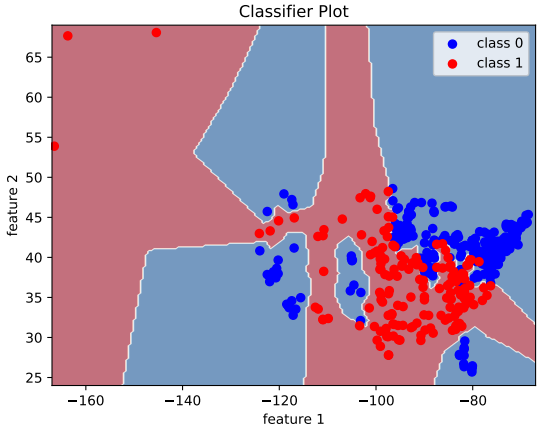


Figure 4: Training classification of KNeighborsClassifier model

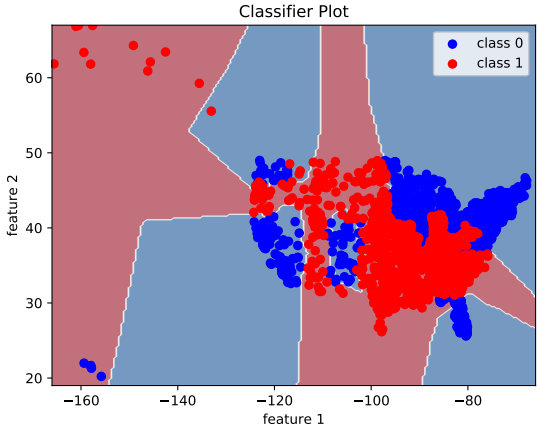


Figure 5: Test classification of KNeighborsClassifier model

code for Q3.3:

```
with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
    dataset = pickle.load(f)

X = dataset['X']
y = dataset['y']
Xtest = dataset['Xtest']
ytest = dataset['ytest']

k = 1
model = KNN(k)
model.fit(X, y)
utils.plotClassifier(model, X, y)
fname = os.path.join("..", "figs", "q3_3_knn_training.pdf")
plt.savefig(fname)
```

```
print("\nFigure saved as '%s'" % fname)

utils.plotClassifier(model, Xtest, ytest)
fname = os.path.join("..", "figs", "q3_3_knn_test.pdf")
plt.savefig(fname)
print("\nFigure saved as '%s'" % fname)

model = KNeighborsClassifier(n_neighbors=1, p=2)
model.fit(X, y)
utils.plotClassifier(model, X, y)
fname = os.path.join("..", "figs", "q3_3_sklearn_kneighbors_training.pdf")
plt.savefig(fname)
print("\nFigure saved as '%s'" % fname)

utils.plotClassifier(model, Xtest, ytest)
fname = os.path.join("..", "figs", "q3_3_sklearn_kneighbors_test.pdf")
plt.savefig(fname)
print("\nFigure saved as '%s'" % fname)
```

4. Why is the training error 0 for $k = 1$?
   Because with $k = 1$, every example's nearest neighbor is itself. The result of predicted y is just the value in the labels.

5. If you didn't have an explicit test set, how would you choose k?
   Separate the data into training set and validation set. Use validation set to predict the performance of the model and tune value of k.

4 Random Forests

4.1 Implementation

Rubric: {code:4,reasoning:3}

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.[1] 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. Why doesn't the random tree model have a training error of 0?
   Random tree info gain
   Training error: 0.186
   Testing error: 0.489
   Because when we do bootstrap, replace parameter is set to True. Thus, some data is missing and some data is duplicated in the training set. The training process doesn't see all the data and it results in training error not being 0. For the same reason, random choice of features inside random stump will

[1] 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)`.

also increase the training error.

2. Create a class `RandomForest` in a file called `random_forest.py` that 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.

```
class RandomForest():

    def __init__(self, max_depth, num_trees):
        self.max_depth = max_depth
        self.num_trees = num_trees
        self.trees = []

    def fit(self, X, y):
        N, D = X.shape
        trees = [None] * self.num_trees
        for i in range(self.num_trees):
            model = RandomTree(max_depth=self.max_depth)
            model.fit(X, y)
            trees[i] = model
        self.trees = trees

    def predict(self, Xtest):
        T, D = Xtest.shape
        predictions = np.zeros([T, self.num_trees])
        for j in range(self.num_trees):
            predictions[:, j] = self.trees[j].predict(Xtest)
        predictions_mode = np.zeros(T)
        for i in range(T):
            predictions_mode[i] = utils.mode(predictions[i,:])
        return predictions_mode
```

3. 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.
Decision tree info gain
Training error: 0.000
Testing error: 0.367
Random tree info gain
Training error: 0.186
Testing error: 0.489
Random forest info gain
Training error: 0.000
Testing error: 0.186
The results are what we expected. Random forest gives lower testing error than decision tree and random tree since random forest averages deep random to reduce overfitting. Bootstrap in random forest also reduce overfitting. Unlike single random tree, random forest with 50 random trees ensures that all data has been visited during training process which results in training error of 0.
Code for Q4.1.3:

```
print("Decision tree info gain")
evaluate_model(DecisionTree(max_depth=np.inf,
                            stump_class=DecisionStumpInfoGain))
```

```
print("Random tree info gain")
evaluate_model(RandomTree(max_depth=np.inf))
print("Random forest info gain")
evaluate_model(RandomForest(max_depth=np.inf, num_trees=50))
```

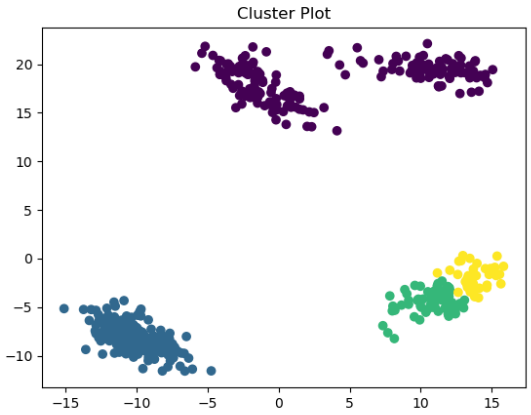
4. Compare your implementation with scikit-learn's `RandomForestClassifier` for both speed and accuracy, and briefly discuss. You can use all default hyperparameters if you wish, or you can try changing them.
   Random forest info gain
   Training error: 0.000
   Testing error: 0.163
   Our random forest took 7.693237 seconds
   RandomForestClassifier from sklearn info gain
   Training error: 0.000
   Testing error: 0.167
   Sklearn RandomForestClassifier took 0.094955 seconds

   The training error are both 0. The testing error are very similar between our random forest and sklearn's RandomForestClassifier. However, sklearn's RandomForestClassifier is much faster than our random forest. our decision stump implementation runs in $O(n^2d)$ where sklearn's implementation runs in $O(ndlog(n))$
   Code for Q4.1.4:

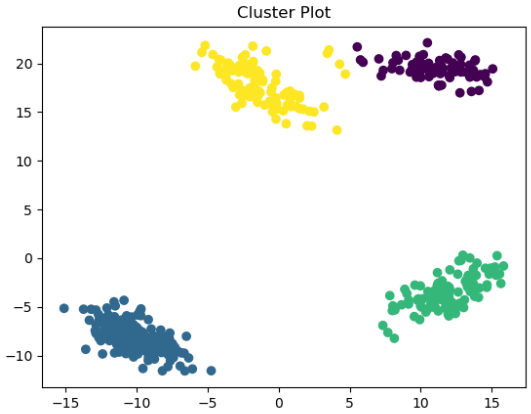
```
print("Random forest info gain")
t = time.time()
evaluate_model(RandomForest(max_depth=np.inf, num_trees=50))
print("Our random forest took %f seconds"
      % (time.time()-t))

print("RandomForestClassifier from sklearn info gain")
t = time.time()
evaluate_model(RandomForestClassifier(n_estimators=50,
                                      criterion='entropy',
                                      max_depth=None))
print("Sklearn RandomForestClassifier took %f seconds"
      % (time.time()-t))
```

5 Clustering

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

Rubric: {reasoning:5}

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, add a new function called `error` that takes the same input as the `predict` function but that returns the value of this above objective function.
   9370.516947021912
   Code for Q5.1.1:

```
def error(self, X):
    N, D = X.shape
    y_pred = self.predict(X)
    approx_val = np.zeros((N, D))

    for i in range(N):
        approx_val[i] = self.means[y_pred[i]]

    error = 0
    for i in range(N):
        for j in range(D):
            error += math.pow((approx_val[i, j] - X[i, j]), 2)

    return error
```

2. What trend do you observe if you print the value of this error after each iteration of the k-means algorithm?
   I can't figure out how to do this haha.
   Error decreases after each iteration. However, it depends on the initialization of clusters. Since it's randomly innitialized, if the error at the beginning is close enough to the optimal error, it decreases but not dramatically. If the initialization is far away from the end result, error decreases much more dramatically.

3. Using the code from question 5 in `main.py` (modify if needed), output the clustering obtained by running k-means 50 times (with $k = 4$) and taking the one with the lowest error. Submit your plot.
   The minimum error was 3071.4680526538564. The clustering plot is shown in following figure

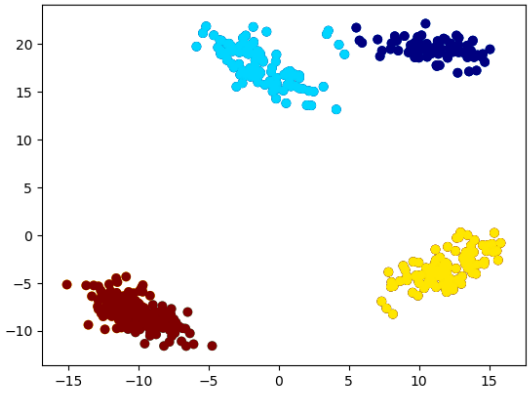


Figure 6: Clustering with lowest error

4. Looking at the hyperparameters of scikit-learn's `KMeans`, explain the first four (`n_clusters`, `init`, `n_init`, `max_iter`) very briefly.
   n_clusters is the number of clusters to form.
   init specifies the method for initialization. Default is k-means++. It also has "random" and ndarray.
   n_init is the number of time the k-means algorithm will be run with different centroid seeds.
   max_iter is maximum number of iterations of the k-means algorithm for a single run.

Code for Q5.1:

```
elif question == '5.1':
    X = load_dataset('clusterData.pkl')['X']
    min_error = np.Inf

    for i in range(50):
        model = Kmeans(k=4)
        model.fit(X)

        error = model.error(X)
        if (error < min_error):
            min_error = error
            plt.scatter(X[:, 0], X[:, 1], c=model.y, cmap="jet")

            fname = os.path.join("..", "figs", "q5_1.png")
            plt.savefig(fname)
        print("Figure saved as {} with minimum error of {}"
              .format(fname, min_error))
```

5.2 Selecting k in k-means

Rubric: {reasoning:5}

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` function.
   By choosing k = number of examples will give error of 0 and every training data point becomes its own mean and we don't get "regions" that new data can by mapped to easily. However, in fact we want to choose a smaller k for generalization.
2. Explain why even evaluating the `error` function on test data still wouldn't be a suitable approach to choosing k.
   The test error will still be small if a large k is chosen.
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.

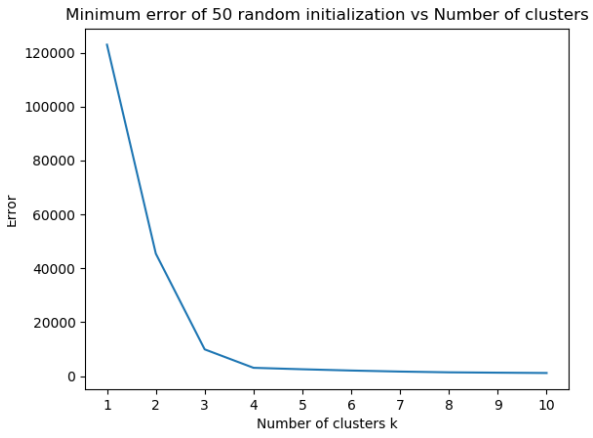


Figure 7: Clustering with lowest error with k from 1 to 10

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.
   With $k = 2$, the slope changes but not greatly. With $k = 3$, the biggest change in slope appear. With $k = 4$, although the change in slope is not as big as with $k = 3$, it gives lower error and no significant change after that. So $k = 3$ and $k = 4$ are reasonable to pick.

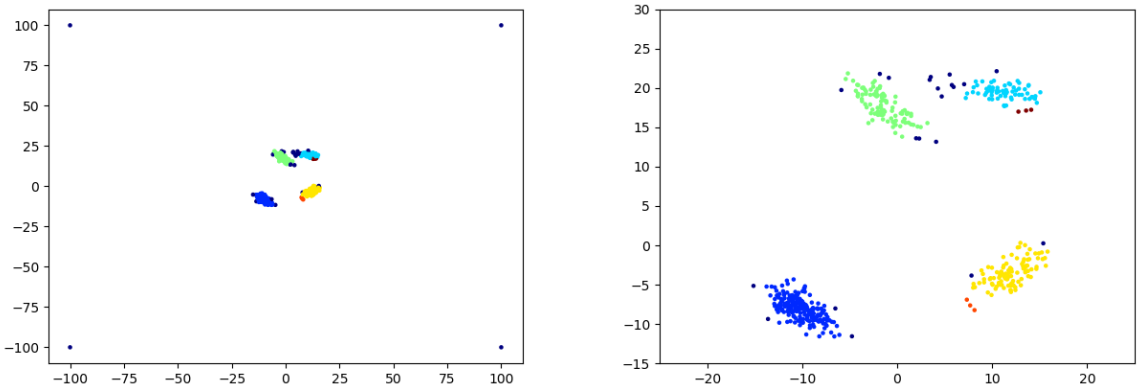
Code for Q5.2:

```
elif question == '5.2':
    X = load_dataset('clusterData.pkl')['X']
    min_error = np.full(10, np.inf)

    for kk in range(1,11):
        for i in range(50):
            model = Kmeans(k=kk)
            model.fit(X)
            error = model.error(X)
            if error < min_error[kk - 1]:
                min_error[kk - 1] = error

    plt.plot(min_error)
    plt.xticks(np.arange(0, 10), np.arange(1, 11))
    plt.xlabel("Number of clusters k")
    plt.ylabel("Error")
    plt.title("Minimum error of 50 random initialization vs Number of clusters")

    fname = os.path.join("..", "figs", "q5_2.png")
    plt.savefig(fname)
```

5.3 Density-Based Clustering

Rubric: {reasoning:2}

If you run `python main.py -q 5.3`, it will apply the basic density-based clustering algorithm to the dataset from the previous part, but with some outliers added. The final output should look somewhat like this:



(The right plot is zoomed in to show the non-outlier part of the data.) Even though we know that each object was generated from one of four clusters (and we have 4 outliers), the algorithm finds 6 clusters and

does not assign some of the original non-outlier objects to any cluster. However, the clusters will change if we change the parameters of the algorithm. Find and report values for the two parameters, `eps` (which we called the "radius" in class) and `minPts`, such that the density-based clustering method finds:

1. The 4 "true" cluster
   With eps=3 and minPts=3, it gives 4 clusters

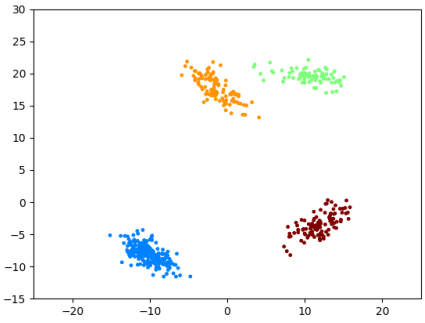


Figure 8: 4 clusters

2. 3 clusters (merging the top two, which also seems like a reasonable interpretaition)
   With eps from 4 to 12 and minPts=3, it gives 3 clusters

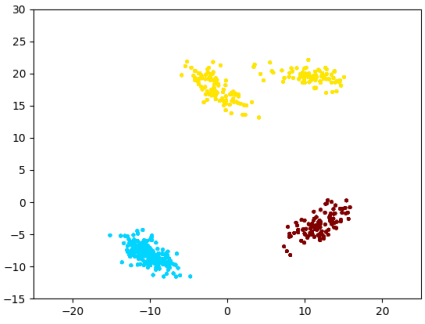


Figure 9: 3 clusters

3. 2 clusters
   With eps from 13 to 15 and minPts=3, it gives 2 clusters

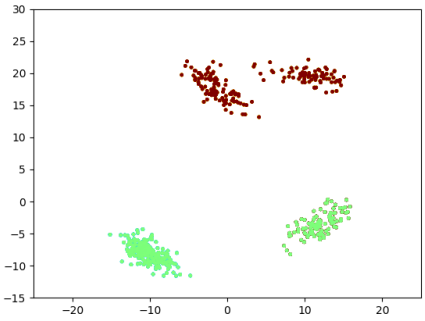


Figure 10: 2 clusters

4. 1 cluster
   With eps from 16 and minPts=3, it gives 1 cluster

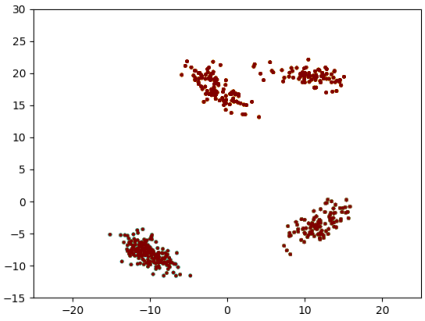


Figure 11: 1 cluster

Code for Q5.3:

```
    elif question == '5.3':
        X = load_dataset('clusterData2.pkl')['X']

        for epsilon in range(1, 17):
            model = DBSCAN(eps=epsilon, min_samples=3)
            y=model.fit_predict(X)
            if np.unique(model.labels_).size <= 5:
                print("witl ep of {}, it gives clustering of {}"
                      .format(epsilon, np.unique(model.labels_)))
                plt.scatter(X[:, 0], X[:, 1], c=y, cmap="jet", s=5)
                fname = os.path.join("..", "figs", "q5_3_" +
                                     str(np.unique(model.labels_).size - 1) + ".png")
                plt.xlim(-25, 25)
                plt.ylim(-15, 30)
                plt.savefig(fname)
                print("\nFigure saved as '%s'" % fname)
```

6 Very-Short Answer Questions

Rubric: {reasoning:13}

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 an advantage of using a boxplot to visualize data rather than just computing its mean and variance?
   With boxplot, the data distribution can be visualized. With mean and variance, two sets of can have same mean and variance while they can distribute very differently. The boxplot also clearly isolate any outliers that might affect the mean and variance.

2. What is a reason that the the data may not be IID in the email spam filtering example from lecture?
   In email spam filtering example, each feature is the appearance of certain word. However, words may have dependencies on each other so data is not independent. Also some words appear way more frequently than other. So it is not identically distributed either.

3. What is the difference between a validation set and a test set?
   A validation set is part of training set and it is used to estimate the test error without looking into the test set. A test set is the data set which should not be visible before the training finish.

4. Why can't we (typically) use the training error to select a hyper-parameter?
   To avoid over-fitting. If the training error is used to select a hyper-parameter, the model will be biased to the training data.

5. What is the effect of n on the optimization bias (assuming we use a parametric model).
   Optimization bias shrinks quickly with increasing n. But it's still non-zero and will grow if validation set is over-used.

6. What is an advantage and a disadvantage of using a large k value in k-fold cross-validation.
   Advantage: Less dependency on the specified section of the data set and Validation error will be more accurate. Disadvantage: More computationally expensive.

7. Why can we ignore $p(x_i)$ when we use naive Bayes?
   While applying Bayes rule to $p(y = c_i) \mid x + ij$, $p(x_i)$ becomes the common denominator on both sides. Ignoring it won't affect the result of final probability comparison.

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

   (a) Our estimate of $p(y_i)$ for some y_i.
   Parameter; we can extract the value by looking at y

   (b) Our estimate of $p(x_{ij} \mid y_i)$ for some x_{ij} and y_i.
   Parameter; we can extract this probability by looking at all x_j's where y_i is true

   (c) The value β in Laplace smoothing.
   Hyper-parameter; we must provide β

9. What is the effect of k in KNN on the two parts (training error and approximation error) of the fundamental trade-off. Hint: think about the extreme values.
   Eapprox = Etest - Etraining = 0. When k=1, training error is 0 because every point's nearest neighbour is itself. As k gets bigger, training error gets bigger as well. Eapprox starts off high and gets smaller with increasing k, until we hit the optimal k value, where Eapprox is very close to 0. Then, Etest rises at approximately the same rate as Etraining, which leaves Eapprox near 0.

10. Suppose we want to classify whether segments of raw audio represent words or not. What is an easy way to make our classifier invariant to small translations of the raw audio?
    Adding slightly transformed data during training such as changing volume on existing training data can capture these differences in the model.

11. Both supervised learning and clustering models take in an input x_i and produce a label y_i. What is the key difference?
    The true labels are provided for supervised learning, but no labels are provided for clustering during training. Supervised learning will apply a pre-trained model to x_i to classify it, whereas clustering models will compare x_i to the training dataset to decide what label to give y_i.

12. Suppose you chose k in k-means clustering (using the squared distances to examples) from a validation set instead of a training set. Would this work better than using the training set (which just chooses the largest value of k)?
    If we use the training set and it chooses the largest value of k, the edge case would be k=n. Since the validation set uses m training points, where m ¡ n, the maximum k will be k=m. Training and testing error gets worse as k gets closer to n, so the fit over the validation set will perform better

13. In k-means clustering the clusters are guaranteed to be convex regions. Are the areas that are given the same label by KNN also convex?
    No. Areas by KNN are not guaranteed to be convex.

Appendices

A main.py

```
# basics
import os
import pickle
import argparse
import time
import math
import matplotlib.pyplot as plt
import numpy as np

# sklearn imports
from sklearn.naive_bayes import BernoulliNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# our code
import utils

from knn import KNN

from naive_bayes import NaiveBayes

from decision_stump import DecisionStumpErrorRate, DecisionStumpEquality, DecisionStumpInfoGain
from decision_tree import DecisionTree
from random_tree import RandomTree
from random_forest import RandomForest

from kmeans import Kmeans
from sklearn.cluster import DBSCAN


def load_dataset(filename):
    with open(os.path.join('..','data',filename), 'rb') as f:
        return pickle.load(f)


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-q','--question', required=True)

    io_args = parser.parse_args()
    question = io_args.question

    if question == "1":
        with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
            dataset = pickle.load(f)

        X, y = dataset["X"], dataset["y"]
        X_test, y_test = dataset["Xtest"], dataset["ytest"]
```

```
    model = DecisionTreeClassifier(max_depth=2, criterion='entropy', random_state=1)
    model.fit(X, y)

    y_pred = model.predict(X)
    tr_error = np.mean(y_pred != y)

    y_pred = model.predict(X_test)
    te_error = np.mean(y_pred != y_test)
    print("Training error: %.3f" % tr_error)
    print("Testing error: %.3f" % te_error)

elif question == "1.1":
    with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
        dataset = pickle.load(f)

    X, y = dataset["X"], dataset["y"]
    X_test, y_test = dataset["Xtest"], dataset["ytest"]

    depth = np.arange(1, 16)
    tr_error = np.zeros(depth.size)
    te_error = np.zeros(depth.size)

    for i, cur_depth in enumerate(depth):
        model = DecisionTreeClassifier(max_depth=cur_depth,
                                       criterion='entropy',
                                       random_state=1)
        model.fit(X, y)
        y_pred = model.predict(X)
        tr_error[i] = np.mean(y_pred != y)

        y_pred = model.predict(X_test)
        te_error[i] = np.mean(y_pred != y_test)

    plt.plot(depth, tr_error, label="Training")
    plt.plot(depth, te_error, label="Test")
    plt.xlabel("Depth of Decision Tree")
    plt.ylabel("Classification Error rate")
    plt.legend()
    fname = os.path.join("..", "figs",
                         "q1_1.pdf")
    plt.savefig(fname)

elif question == '1.2':
    with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
        dataset = pickle.load(f)

    X, y = dataset["X"], dataset["y"]
    n, d = X.shape

    depth = np.arange(1, 16)
    tr_error = np.ones(depth.size)
    te_error = np.ones(depth.size)

    in_order = True

    if in_order:
```

```
            X_train, X_val, y_train, y_val = \
                train_test_split(X,
                                 y,
                                 test_size=0.5,
                                 shuffle=False)
        else:
            X_val, X_train, y_val, y_train = \
                train_test_split(X,
                                 y,
                                 test_size=0.5,
                                 shuffle=False)

        for i, cur_depth in enumerate(depth):
            model = DecisionTreeClassifier(max_depth=cur_depth,
                                           criterion='entropy',
                                           random_state=1)
            model.fit(X_train, y_train)

            y_pred = model.predict(X_val)
            te_error[i] = np.mean(y_pred != y_val)

        print("minimum training error of {} at depth of {}"
              .format(np.min(te_error), np.argmin(te_error) + 1))
        print(te_error)

    elif question == '2.2':
        dataset = load_dataset("newsgroups.pkl")

        X = dataset["X"]
        y = dataset["y"]
        X_valid = dataset["Xvalidate"]
        y_valid = dataset["yvalidate"]
        groupnames = dataset["groupnames"]
        wordlist = dataset["wordlist"]

        print("Column 51 of X is the word '{}'".format(wordlist[50]))
        words = ', '.join(wordlist[np.where(X[500, ] == 1)])
        print("Training example 501 has words: {}".format(words))

        print("Training example 501 comes from newsgroup {}"
              .format(groupnames[y[500]]))

    elif question == '2.3':
        dataset = load_dataset("newsgroups.pkl")

        X = dataset["X"]
        y = dataset["y"]
        X_valid = dataset["Xvalidate"]
        y_valid = dataset["yvalidate"]

        print("d = %d" % X.shape[1])
        print("n = %d" % X.shape[0])
        print("t = %d" % X_valid.shape[0])
        print("Num classes = %d" % len(np.unique(y)))

        model = NaiveBayes(num_classes=4)
```

```
    model.fit(X, y)
    y_pred = model.predict(X_valid)
    v_error = np.mean(y_pred != y_valid)
    print("Naive Bayes (ours) validation error: %.3f" % v_error)

    model = BernoulliNB()
    model.fit(X, y)
    y_pred = model.predict(X_valid)
    v_error = np.mean(y_pred != y_valid)
    print("BernoulliNB validation error: %.3f" % v_error)

elif question == '3':
    with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
        dataset = pickle.load(f)

    X = dataset['X']
    y = dataset['y']
    Xtest = dataset['Xtest']
    ytest = dataset['ytest']

    k = [1, 3, 10]
    for k_test in k:
        model = KNN(k_test)
        model.fit(X, y)
        y_pred = model.predict(X)
        y_error = np.mean(y_pred != y)
        print("KNN with k={} gets training error of {}"
              .format(k_test, y_error))

        y_pred = model.predict(Xtest)
        y_error = np.mean(y_pred != ytest)
        print("KNN with k={} gets validation error of {}"
              .format(k_test, y_error))

elif question == '3.3':
    with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
        dataset = pickle.load(f)

    X = dataset['X']
    y = dataset['y']
    Xtest = dataset['Xtest']
    ytest = dataset['ytest']

    k = 1
    model = KNN(k)
    model.fit(X, y)
    utils.plotClassifier(model, X, y)
    fname = os.path.join("..", "figs", "q3_3_knn_training.pdf")
    plt.savefig(fname)
    print("\nFigure saved as '%s'" % fname)

    utils.plotClassifier(model, Xtest, ytest)
    fname = os.path.join("..", "figs", "q3_3_knn_test.pdf")
    plt.savefig(fname)
    print("\nFigure saved as '%s'" % fname)
```

```
    model = KNeighborsClassifier(n_neighbors=1, p=2)
    model.fit(X, y)
    utils.plotClassifier(model, X, y)
    fname = os.path.join("..", "figs", "q3_3_sklearn_kneighbors_training.pdf")
    plt.savefig(fname)
    print("\nFigure saved as '%s'" % fname)

    utils.plotClassifier(model, Xtest, ytest)
    fname = os.path.join("..", "figs", "q3_3_sklearn_kneighbors_test.pdf")
    plt.savefig(fname)
    print("\nFigure saved as '%s'" % fname)

elif question == '4':
    dataset = load_dataset('vowel.pkl')
    X = dataset['X']
    y = dataset['y']
    X_test = dataset['Xtest']
    y_test = dataset['ytest']
    print("\nn = %d, d = %d\n" % X.shape)

    def evaluate_model(model):
        model.fit(X,y)

        y_pred = model.predict(X)
        tr_error = np.mean(y_pred != y)

        y_pred = model.predict(X_test)
        te_error = np.mean(y_pred != y_test)
        print("    Training error: %.3f" % tr_error)
        print("    Testing error: %.3f" % te_error)

    print("Decision tree info gain")
    evaluate_model(DecisionTree(max_depth=np.inf,
                                stump_class=DecisionStumpInfoGain))

    print("Random tree info gain")
    evaluate_model(RandomTree(max_depth=np.inf))

    print("Random forest info gain")
    t = time.time()
    evaluate_model(RandomForest(max_depth=np.inf, num_trees=50))
    print("Our random forest took %f seconds"
          % (time.time()-t))

    print("RandomForestClassifier from sklearn info gain")
    t = time.time()
    evaluate_model(RandomForestClassifier(n_estimators=50,
                                          criterion='entropy',
                                          max_depth=None))
    print("Sklearn RandomForestClassifier took %f seconds"
          % (time.time()-t))

elif question == '5':
    X = load_dataset('clusterData.pkl')['X']

    model = Kmeans(k=4)
```

```
    model.fit(X)
    y = model.predict(X)
    plt.scatter(X[:,0], X[:,1], c=y, cmap="jet")

    fname = os.path.join("..", "figs", "kmeans_basic.png")
    plt.savefig(fname)
    print("\nFigure saved as '%s'" % fname)

elif question == '5.1':
    X = load_dataset('clusterData.pkl')['X']
    min_error = np.Inf

    for i in range(50):
        model = Kmeans(k=4)
        model.fit(X)

        error = model.error(X)
        if (error < min_error):
            min_error = error
            plt.scatter(X[:, 0], X[:, 1], c=model.y, cmap="jet")

            fname = os.path.join("..", "figs", "q5_1.png")
            plt.savefig(fname)
        print("Figure saved as {} with minimum error of {}"
                .format(fname, min_error))

elif question == '5.2':
    X = load_dataset('clusterData.pkl')['X']
    min_error = np.full(10, np.inf)

    for kk in range(1,11):
        for i in range(50):
            model = Kmeans(k=kk)
            model.fit(X)
            error = model.error(X)
            if error < min_error[kk - 1]:
                min_error[kk - 1] = error

    plt.plot(min_error)
    plt.xticks(np.arange(0, 10), np.arange(1, 11))
    plt.xlabel("Number of clusters k")
    plt.ylabel("Error")
    plt.title("Minimum error of 50 random initialization vs Number of clusters")

    fname = os.path.join("..", "figs", "q5_2.png")
    plt.savefig(fname)

elif question == '5.3':
    X = load_dataset('clusterData2.pkl')['X']

    for epsilon in range(1, 17):
        model = DBSCAN(eps=epsilon, min_samples=3)
        y=model.fit_predict(X)
        if np.unique(model.labels_).size <= 5:
            print("witl ep of {}, it gives clustering of {}"
                  .format(epsilon, np.unique(model.labels_)))
```

```
            plt.scatter(X[:, 0], X[:, 1], c=y, cmap="jet", s=5)
            fname = os.path.join("..", "figs", "q5_3_" +
                                 str(np.unique(model.labels_).size - 1) + ".png")
            plt.xlim(-25, 25)
            plt.ylim(-15, 30)
            plt.savefig(fname)
            print("\nFigure saved as '%s'" % fname)
    else:
        print("Unknown question: %s" % question)
```

B kmeans.py

```
import numpy as np
from utils import euclidean_dist_squared
import math

class Kmeans:

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

    def fit(self, X):
        N, D = X.shape
        y = np.ones(N)

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

        self.means = means

        while True:
            y_old = y

            # Compute euclidean distance to each mean
            # return dist2 of N*k
            dist2 = euclidean_dist_squared(X, means)
            dist2[np.isnan(dist2)] = np.inf
            y = np.argmin(dist2, axis=1)

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

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

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

```
            error = self.error(X)
            print(error)

        self.means = means

    def predict(self, X):
        means = self.means
        dist2 = euclidean_dist_squared(X, means)
        dist2[np.isnan(dist2)] = np.inf
        self.y = np.argmin(dist2, axis=1)
        return self.y

    def error(self, X):
        N, D = X.shape
        y_pred = self.predict(X)
        approx_val = np.zeros((N, D))

        for i in range(N):
            approx_val[i] = self.means[y_pred[i]]

        error = 0
        for i in range(N):
            for j in range(D):
                error += math.pow((approx_val[i, j] - X[i, j]), 2)

        return error
```

C knn.py

```
"""
Implementation of k-nearest neighbours classifier
"""

import numpy as np
from scipy import stats
import utils

class KNN:

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

    def fit(self, X, y):
        self.X = X # just memorize the training data
        self.y = y

    def predict(self, Xtest):
        y_hat = []
        (t, d) = Xtest.shape  # t*d
        dist = utils.euclidean_dist_squared(self.X, Xtest)  # n*t

        # For each row in test
        for i in range(0, t):
            # each col in N*t corresponds one test vector
            row = sorted(dist[:, i])
```

```
            row_values = np.array(row[0:self.k])

            row_indices = []
            for j in range(0, self.k):
                row_indices.append(np.where(dist[:, i] == row_values[j]))

            y_kneighbors = []
            for m in row_indices:
                y_kneighbors.append(self.y[m])

            y_hat.append(utils.mode(y_kneighbors))

        return y_hat
```

D naive_bayes.py

```
import numpy as np

class NaiveBayes:
    # Naive Bayes implementation.
    # Assumes the feature are binary.
    # Also assumes the labels go from 0,1,...C-1

    def __init__(self, num_classes, beta=0):
        self.num_classes = num_classes
        self.beta = beta

    def fit(self, X, y):
        N, D = X.shape

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

        # Compute the probability of each class i.e p(y==c)
        counts = np.bincount(y)
        p_y = counts / N

        # Compute the conditional probabilities i.e.
        # p(x(i,j)=1 | y(i)==c) as p_xy
        # p(x(i,j)=0 | y(i)==c) as p_xy
        # p_xy = 0.5 * np.ones((D, C))
        # TODO: replace the above line with the proper code
        p_xy = np.zeros((D, 4))

        feature_id = 0
        for row in X.T:
            for example_id, example_val in enumerate(row):
                if example_val == 1:
                    p_xy[feature_id, y[example_id]] += 1
            feature_id += 1

        p_xy[:, 0] /= counts[0]
        p_xy[:, 1] /= counts[1]
        p_xy[:, 2] /= counts[2]
        p_xy[:, 3] /= counts[3]
```

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

    def predict(self, X):

        N, D = X.shape
        C = self.num_classes
        p_xy = self.p_xy
        p_y = self.p_y

        y_pred = np.zeros(N)
        for n in range(N):

            probs = p_y.copy() # initialize with the p(y) terms
            for d in range(D):
                if X[n, d] != 0:
                    probs *= p_xy[d, :]
                else:
                    probs *= (1-p_xy[d, :])

            y_pred[n] = np.argmax(probs)

        return y_pred
```