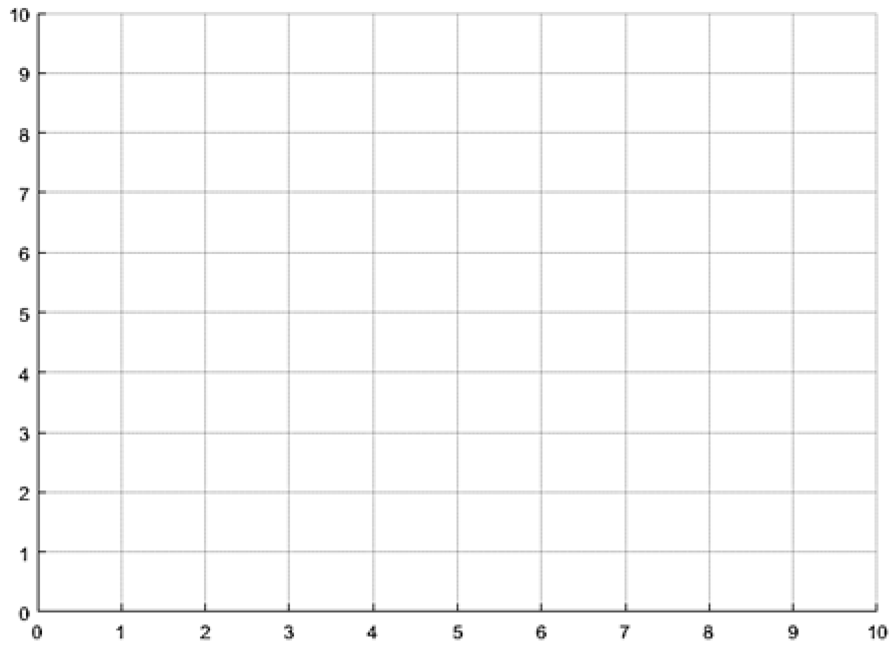
You may complete the following entirely in a Jupyter notebook. Ensure that the notebook has your name on it. Save the notebook as a PDF and submit the PDF through Canvas.

1) **Practice KNN by hand.**
Note that this problem needs to be done on paper, not in a notebook.

Given the following feature matrix, complete the next steps for the KNN algorithm.

Obseration	Feature 1	Feature 2	Feature 3	Class
1	4	7	4	Group 1
2	2	8	2	Group 2
3	2	6	2	Group 1
4	6	4	7	Group 1
5	4	6	8	Group 1
6	8	1	7	Group 2
7	8	5	10	Group 2

1. Make a scatter plot of the data using the first two features. Use ‘x’ for group 1 points, and ‘o’ for group 2 points.



2. Using KNN with $k = 3$, fill in the missing values in the table. Note: KNN can be used to predict the label of a feature vector or to fill in the missing entries of a column in the feature matrix. In this question, use regression KNN to predict the missing entries of feature 3 for the following 3 data samples. To predict the class of each data sample, rely only on the first two

features and use classification KNN (you would normally need to rely on the three features to predict the class label, but for brevity, you can only rely on the first two features).

(a)

Observation	Feature 1	Feature 2	Feature 3	Class
1	9	2		

3. Using kNN with a k=2, fill in the missing values in the table.

(a)

Observation	Feature 1	Feature 2	Feature 3	Class
1	3	6		

4. Using kNN with a k=7, fill in the missing values in the table.

(a)

Observation	Feature 1	Feature 2	Feature 3	Class
1	9	9		

2) **Confusion Matrices**

(a) Given the following vectors of true and predicted labels, construct a confusion matrix. Be sure to indicate which dimensions correspond to the true and predicted labels.

True	Predicted
Bee	Wasp
Bee	Bee
Bee	Bee
Wasp	Wasp
Wasp	Bee
Wasp	Bee
Wasp	Wasp
Wasp	Wasp

(b) Given the following confusion matrix, label the true positives, false positives, false negatives, and true negatives. Assume that "cancer" is the positive class.

	cancer	benign
cancer	35	14
benign	38	60

(c) Calculate accuracy, precision, recall and F1-score from the confusion matrix in (b).

3) **Train-test data splitting**
In lecture, we talked about the importance of splitting data into training and testing sets if we want to evaluate models in a realistic fashion. In this problem, you are going to implement your own function for randomly splitting data and look at the impact of using stratification.

(a) Load in the `haberman.csv` dataset as a feature matrix. The last column indicates the class (either 1 = the patient survived 5 years or longer or 2 = the patient died within 5 years) of each observation.

(b) Randomly assign each observation (row) in the feature matrix to the training or testing set. 70% of the samples should be assigned to the training set, while 30% should be assigned to the testing set. Hint: Consider the Numpy function `random.rand`. If the random number is ≤ 0.7, assign the sample to the training set. Otherwise, assign the sample to the test set.

(c) How many observations of each class do you expect there to be in the training and testing sets? Count the number of observations of each class in the entire data set and in both the training and testing sets. How do your counts for the training and tests compare with the counts for the entire data set? (Hint: Consider the Numpy function `unique` and its `return_counts` argument.)

(d) Randomly assign the observations again. This time, stratify (divide) the observations by class first. (Hint: Consider how you might use a mask.) Then, repeat the random assignment into training and testing sets for each class.

(e) Count the number of observations for each class in both the training and testing sets. How do your counts compare with your expectation?

(f) How could an uneven distribution of the observations impact the training and evaluation of a model? Explain why stratification is important.

4) **The bias-variance tradeoff**
A model's error can be decomposed into variance, bias and irreducible error:

- **Bias** refers to the error that is introduced due to wrong assumptions such as approximating a complicated pattern in data with a simple model. A **high-bias** model is a model that fails to capture the structure in data and results in **underfitting** the training data. High bias or underfitting is usually caused by a model that is too simple or when there is a few features.
- **Variance** refers to the amount by which the model would change if we used a different training data set. A **high-variance** model is a model that does not generalize well to predict new data but performs well on training data, which is also know as **overfitting**. High variance or overfitting is usually caused by a model that is too complex for the data.
- **Irreducible** error refers to the noise that exists in the data itself.

In general, there is a tradeoff between bias and variance. As the model's complexity increases (for example going from a simple linear model to a polynomial model with a higher degree), its bias will typically decrease but its variance will increase. As the model's complexity decreases, its variance will decrease but its bias will increase.

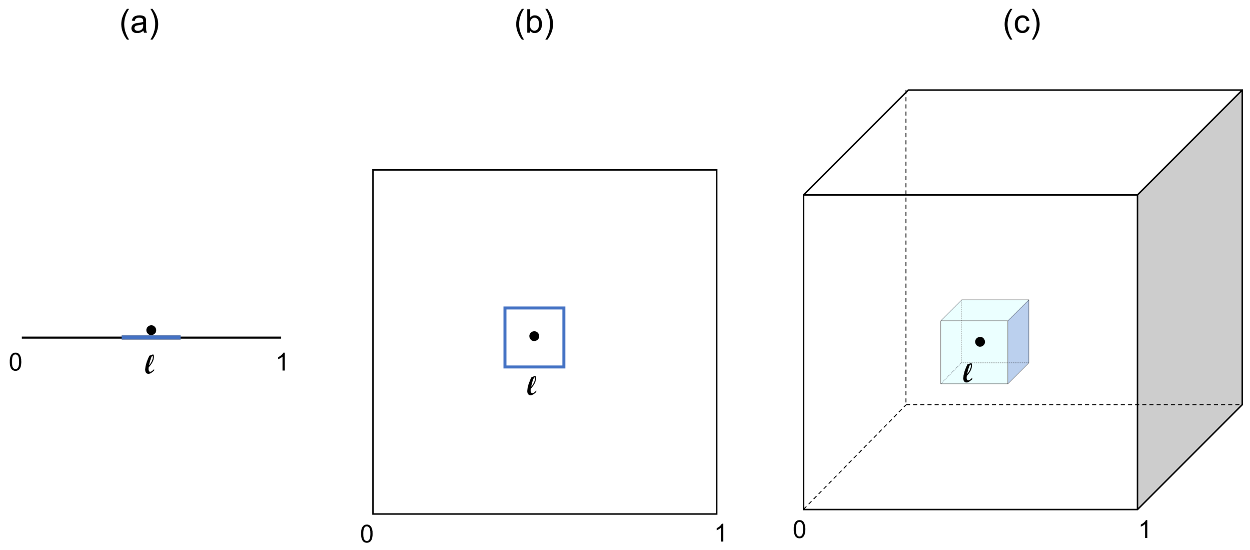
When a model does not perform well on the training set, we know that the model underfits the training set. When a model performs very well on the training set, but fails to show similar performance on the test set, we conclude that the model overfits the training set (hence the importance of splitting data into training and test sets).

In this exercise, you will visualize the bias-variance tradeoff of a KNN model. The complexity of a KNN model can be varied by changing k (the number of nearest neighbors).

1. How do you expect the complexity of a KNN classifier will change if k increases? In other words, as we increase k, will the decision boundary become simpler and smoother or will it have more angles and curves?
2. Load diabetes.csv and extract the feature matrix X and the label vector $\mathbf{y}$. The data consists of diagnostic measurements for some female patients and shows whether they have diabetes or not, as indicated by the last column labeled as outcome. (The data was extracted from here).
3. Use the scikit-learn implementation of KNN (`KNeighborsClassifier`) to instantiate a KNN classifier. For now, keep the default value for the number of neighbors.
4. You will now evaluate the model using cross validation. Use the method `cross_validate` provided by scikit-learn to evaluate KNN. This method splits the data into 5 folds and returns the training scores and testing scores for each fold. Make sure to set the argument `return_train_score` to `True`. Find the average of the 5 training scores and the average of the 5 testing scores.
5. `KNeighborsClassifier` uses k=5 (number of neighbors) by default. Repeat what you did in the previous part by varying k from 1 to 50. For each value of k, find the average accuracy score and the average testing score. On a same figure, plot two lines: one showing how the training accuracy changes with k and another line showing how the testing accuracy changes with k.
6. Can you spot where the model has high variance (overfitting) and where it has low variance?
7. KNN can be also used for regression. Repeat the previous steps using the scikit-learn implementation of KNN regressor (`KNeighborsRegressor`). For this part, load the boston dataset that is provided by scikit-learn (`load_boston`) and use the mean squared error (`mean_squared_error`) as evaluation metric: you will need to specify the `scoring` argument in `cross_validate` as `scoring = make_scorer(mean_squared_error)`. Vary k from 1 to 400.

5) **The curse of dimensionality**
This exercise was adapted from the book "Introduction to Statistical Learning" (exercise 4 in chapter 4).

KNN tends to perform poorly as the number of features increases. This is due to a phenomenon known as the curse of dimensionality, which we will explore in this exercise.



1. Suppose we have a set of observations that consist of one feature; those observations cover uniformly the interval from 0 to 1 (the black line shown in (a)). Suppose that we wish to predict the response for a new observation (the black dot shown in (a)); this new observation is centered in a sub-interval of length ℓ observations. To make the prediction for this new data point, we want to only use the observations that lie in this sub-interval (the blue line shown in (a)). On average, what fraction of the available observations we will use to make this prediction?

2. Suppose we have a set of observations that consist of two features; those obervations cover uniformly the unit square (the black square shown in (b)). Suppose that we wish to predict the response for a new observation (the black dot shown in (b)); this new observation is centered in a smaller square with ℓ as the side length. To make the prediction for this new data point, we want to only use the observations that lie in this small square (the blue square shown in (b)). On average, what fraction of the available observations we will use to make this prediction?

3. Repeat the same question for when the data has 3 features (as shown in (c)). Can you generalize your answer for any number of features p?

4. Using the general form from subpart 3 and assuming that $\ell = 0.1$ (representing the fraction of needed observations), how does the fraction of the available observations used in prediction change with p? You can show a plot or explain your answer. Can you argue that a disadvantage of KNN is that when p is large, there are few observations that are close to the new data

point? *Optional* - you can think about the fraction of observations as k/N, where k is the number of nearest neighbors and N is the number of observations. How many observations (N expressed in terms of p) do we need for $k = 10$ and $\ell = 0.1$?

5. Assume that you need inside the hypercube 10% of the available observations. (Note: a hypercube is a generalization of a cube; when p=1 it is a line, when p=2 it is a square, when $p = 3$ it is a cube). What is the length of each side of the hypercube? How does the length change as p increases? Again you can show a plot or explain your answer. Can you argue that when p is large, what KNN assumes as a near neighbor to a data point, might not be actually similar to this data point?

6) **ROC plot**

In this problem, we are going to walk you through creating a Receiver Operator Characteristic (ROC) curve from scratch. Most classifiers can output a continuous-valued "score," not just a categorical class label. The score represents the confidence of the classifier in its decision. Some classifiers such as Logistic Regression and Naïve Bayes are able to predict not just a score but the probability $P(y = 1|\mathbf{x})$ that a sample with feature vector $\mathbf{x}$ belongs to class 1. A ROC curve allows us to see how different thresholds for the score produce different trade offs between true and false positive rates. The true positive rate indicates what fraction of the positive predictions are correct, while the false positive rate indicates what fraction of the negative are mis-predicted as positives.

(a) Load the data in the provided file `ex1_results.csv`. The first column contains the estimated probabilities (generated from a machine learning model) that the observation belongs to class 1, while the second column contains the true labels (either 0 or 1).

(b) Use 3 thresholds (0.25, 0.5, and 0.75) for the scores to get binary predicted labels. If the score for an observation is less than the threshold, its predicted label should be 0. Otherwise, the predicted label should be 1.

(c) For each threshold, create a confusion matrix.

(d) For each confusion matrix, calculate the true and false positive rates. What do you notice about the two rates as the threshold is changed? Which thresholds give the highest and lowest true positive rates? Which thresholds give the highest and lowest false positive rates?

(e) Create a list of all unique scores. Sort the scores from highest to lowest. Use each score as a threshold and predict the true and false positive rates:

   (i) Use the threshold to predict the labels for each observation.

   (ii) Calculate the true and false positive rates.

(f) Plot the true (horizontal) and false (vertical) positive rates for each threshold.