# Polynomial regression assignment

Previously, you have learned linear regression (least squares method) to train linearly separable data sets. Also, you have also learned the cross-validation method to choose the best hyperparameters and kernel trick to featurize data matrix in an efficient way.

In this coding assignment, you will learn how to learn data with nonlinear boundaries by using polynomial features. In the first half of the assignment, you are asked to learn several toy data sets. You will see that by lifting the data matrix with polynomial features you can learn non-linear decision boundaries that you weren't able to learn with linear regression. The degree of the polynomial features is an important parameter in the polynomial regression. It controls the complexity of the decision boundary and whether or not we overfit our data. You will apply the validation method to tune this hyperparameter. In addition, you will train the same data set using both polynomial regression and poly kernel regression. By doing this, you will be able to see the pros and cons of these two learning algorithms.

In the second half of the assignment, you will deal with a real-life data set. It is important to realize that real-life data is often non-ideal and it requires extra preprocessing to get good training results. After some preprocessing of the data, you will apply the polynomial regression and poly kernel regression algorithms you developed above to learn the data set. The pros and cons of these two methods should be more obvious in this setting. In reality, we often do machine learning without writing our own algorithms. Instead, we can leverage a lot of powerful and convenient machine learning packages to accelerate our machine learning projects. To get you familiar with machine learning package Sklearn, you will be asked to learn the same real-life data set using the Sklearn functions. You will see how your own implementation of polynomial regression and poly kernel regression compare to the Sklearn version.

## Section 0: Helper functions

**Objective**: We want to check that you understand the mathematical machinery behind polynomial regression so even if you use ML libraries in the future, you know what those functions are doing behind the scenes.

Implement the following helper functions:

(0) Poly_regression(): Inputs are the training data, training labels, and also the test data. Trains a linear regression model using the training data and labels. Returns the predicted values using the test data (these will not be classified into labels yet) and the weights used to make the prediction.

Please note that we need to use ridge regularization later in this problem which hasn't been covered yet in the course (you will learn about it later this week). For now don't worry about the details but do be sure to add the regularization term in the closed form solution. (Hint: use the closed form solution to ridge regression i.e. w = (X.T @ X + lambda*identity)^{-1} @ X.T @ y instead of the classic OLS solution)

(1) Classify(): Inputs are the predicted y values. Returns the classified labels. In this case, classify into two classes -1 and +1 based on the boundary 0.

(2) Accuracy_percentage(): Inputs are the true labels and the predicted labels. Returns the accuracy percentage i.e. the number of labels that matched between true and predicted / total number of labels.

(3) Assemble_feature(x, D): Inputs are the x raw data and the D degree of polynomial. Returns the "lifted" D degree polynomial features created from the raw data. Do not use any sklearn functions for this. We want you to explicitly create the monomials for polynomial regression.

## Section 1: Circle data - a simple example

Here we have some randomly generated data centered around 0 with two classes, +1 and -1. We have already split the data into training, validation, and test sets for you. In this section, we will see in what situations polynomial regression will out perform linear regression.

### Visualize the data and train linear/polynomial regression:

**Objective**: Through a simple example, we want you to understand why polynomial regression performs better than linear regression in certain classification problems i.e. why polynomial is necessary.

(4) (i) Do you expect linear regression or polynomial regression to perform better? Why?

(ii) Use the helper functions to train a linear regression model using the train data and predict on the test data. Print the test accuracy and visualize the predicted boundary. Comment on what you see. Is this consistent with what you expected?

(5) Use the helper functions to train a 2-degree polynomial regression model using the train data and predict on the test data. Be sure to lift your features first. Print

the test accuracy and visualize the predicted boundary. Comment on what you see. Is this consistent with what you expected?

### Hyperparameter tuning:

**Objective**: We want you to practice hyperparameter tuning with picking the degree of polynomial. Picking hyperparameters is a skill that is needed from many different machine learning algorithms so it will be good to practice on this simple model.

(6) How do we know that a 2-degree polynomial is the best degree polynomial? We need to do hyperparameter tuning. Use the interactive plot to observe how the test accuracy changes as we change the degree of the polynomial.

(7) (i) Now we will do hyperparameter tuning analytically by plotting the training accuracy vs. the degree. Complete the code to calculate the prediction accuracy as a function of the polynomial degree using the validation process. Note we have already split the data into train, validation, and test for you. Which degree is the best based on what you observe?

(ii) Notice how the run time changes as we have larger degree polynomials. Why is this happening?

(8) Finally use your results from hyperparameter tuning to fully train your model and report the test accuracy.

### Kernel and run time:

**Objective**: We want you to see how to practically implement polynomial regression especially when your model becomes more complex. This will give you a better understanding of why the kernel trick is so useful with respect to computation time.

(9) Let’s try a polynomial kernel. Fill in the formula for a polynomial kernel i.e. create the gram matrix. Run the code and comment on what you see in the interactive plot.

(10) Why and when would we want to use polynomial kernel over polynomial regression? You don’t need to write any code for this. Run the code and comment on what you see in the plots. What is happening with run time? Why is this happening?

## Section 2: Star data - more complex boundaries

**Objective**: We want you to practice developing some intuition on the complexity of polynomial regression boundaries by giving you another small dataset to play with.

Here we have some new toy data which has a more complicated boundary than the circle data. We will observe how polynomial regression performs in this scenario.

### Visualize the data:

(11) Run the code and visualize the data. Intuitively, what is your guess for the best degree polynomial in this case?

### Hyperparameter tuning:

(12) Use the interactive plot to see if your guess was correct. What is the best degree in this case? Why is it different from the best degree for the circle data set?

## Section 3: Multiclass classification

**Objective**: Again, we want you to build some intuition about the complexity of polynomial regression boundaries. In this case, instead of having a more complex boundary, we instead have a scenario with more than two classes.

So far, we have only looked at data with two classes, {-1, +1}. What if we had more than two classes? Does polynomial regression still work? Here we have some data with four classes and labels {-3, -1, +1, +3}.

### Visualize the data:

(13) Run the code and visualize the data. What do you think is the best degree polynomial that will fit this data?

(14) Complete the new classify_four_classes() function. You will need to come up with new boundaries for the classes. Afterwards, run the code and use the interactive plot to find the best degree which fits the data. Comment on what you find.

## Section 4: Real world application using wine quality data

**Objective**: We now want you to get some experience with real data! This will include the processing part and some common issues you might run into. Then you need to apply your skills and intuition of polynomial regression to train a classifier. Finally, we want you to get some familiarity with the sk-learn library which lets you do polynomial regression but via lifting and kernel without writing all the helper functions.

The wine quality dataset archives a numerical description to 1600 different kinds of wine. There are 12 numerical values to describe the property of each wine in the dataset. The first 11 values can be seen as the input features. While the twelfth value is an integer output label varies from from 1 to 10, which evaluates the quality of each wine. Our goal is to use the 11-D input feature vector and the 1-D output label to train a machine learning model, using polynomial regression techniques. In this question, you will start with pre-processing a real-world dataset to have a basic understanding of the dataset and the task. Then we will walk you through some tricky problems in machine learning with the problem setting and the starter code, to gain you some intuition on how to deal with the defects of a dataset. At the end we want you to compare the accuracy and the efficiency of different machine learning implementations by observing the behaviors of them. Working on the question will help you roughly see how real-world machine learning would be like. This dataset is available from the UCI machine learning repository (see https://archive.ics.uci.edu/ml/datasets/wine+quality)

## Rescale to prevent overflow:

**Objective:** From this section, you should see the importance of normalizing data so it can be better handled by limited computing power.

First let's load the data. Before applying any learning algorithm to the wine dataset, we will need to preprocess the dataset. In the real-world, pre-processing the data is a very important step since real-life data can be quite imperfect.

(15) Since the raw data matrix is not normalized and contains large values that can cause overflow. We will first rescale each column of the data matrix to the same scale and then perform regression on the new data matrix. Complete the function rescale_matrix() to scale each feature so that it now takes on a value between minValue and maxValue.

## Will regression work?

**Objective**: We want you to see what might go wrong when we don't have a good sample of each class when we are handling a classification problem.

(16) Run the following code to load the data. Play with interactive plot to visualize data with two selected features. Based on your observations on the interactive plot, do you think this data set is linearly separable?

(17) Now let's try to learn the data using regression with polynomial features. The output of regression is a continuous variable. Complete the function classify_wine() below to convert the continuous variable into the wine quality

score (ie. the class label for each data point). Please pay attention to the number of classes in the data.

(18) Complete the missing codes in the function visualize_wine_regression_results() to do regression with polynomials on this data set.

(19) Run the code and make your observations. The interactive plot shows the prediction labels vs the ground-truth label for selected degree of polynomial. Play with this plot, and report the best degree.Comment on effectiveness on training this data set.

## Importance of balanced classes:

**Objective:** Here we will show you the phenomena of unbalanced classes and how that can create bias in our classifier. We also will show you how to get around it by simplifying the problem (although it may or may not be applicable to a real-world setting).

(20) Plot histogram of the data labels. Write down your observations. Based on your observations, what could be one of the reasons that causes the low prediction accuracy in the previous cell?

(21) Now let's complete the missing code below to rebin the data into two classes (with class labels -1 and 1) so that two classes will have a similar number of data points. The histogram above might give you a hint on how to split the data. Note that in real life, if you are asked to predict wine quality score, this approach is not going to help. You will likely have to use a more complex modeling algorithm but for this exercise, we will just simplify the problem to be applicable to polynomial regression.

## Polynomial regression:

**Objective:** Here is more practice for you to implement polynomial regression. In particular, here we ask you to implement the polynomial kernel version.

(22) Now let’s try polynomial regression again on this re-binned data. Implement polynomial regression (you can use the helper functions from before). Comment on how the performance has changed from before.

(23) We saw earlier that using the kernel trick is beneficial for data with a lot of features and when our model gets complex. Implement the polynomial kernel and compare the performance. (You can find inspiration on how to do this implementation in the starter code from earlier parts).

## Using SK-learn:

**Objective:**

Now that we’ve done all the grunt work and understand polynomial regression inside and out, let’s see what tools are available to us when we want to do this in the real world. Here we will get some familiarity with the tools available in sk-learn.

(24) Implement polynomial regression using lifted features with sklearn functions. You will find Ridge() and PolynomialFeatures() helpful. Compare the results you get with your manual implementation and write down your observations.

(25) Implement polynomial regression using a polynomial kernel with sklearn functions. You will find KernelRidge() helpful. Compare the results you get with your manual implementation and write down your observations.