Homework #1

Homework is **due at the start of class** on **Tuesday, September 11, 2018**, without exceptions unless permission was obtained from the instructor **in advance**.
You may collaborate with other students, discuss the problems and work through solutions together. However, you **must write up your solutions on your own**, without copying another student's work or letting another student copy your work. In your solution for each problem, you must write down the names of any person with whom you discussed it.
This will **not** affect your grade.

1. (**Nonlinear Regression**, 50 points) This problem extends ordinary least squares regression, which uses a *linear hypothesis class*, to non-linear hypothesis classes. This problem also serves as a refresher for Python, which we will use for the programming assignments.

   a. (**Setup**) Install Python 3.7 (https://www.python.org/download/releases/3.0/), and **optionally** a Python IDE such as PyCharm (https://www.jetbrains.com/pycharm/) In addition, we will also be using the following Python packages extensively: numpy, scipy, matplotlib, and scikit-learn, and these must be imported as needed:

   ```
   import numpy as np
   import matplotlib.pyplot as plt
   ```

   b. (**Generate Synthetic Data**, 10 points) The function we want to fit is $y_{\text{true}} = f_{\text{true}}(x) = \sin(x) + \sin(2x)$. This is a **univariate function** as it has only one input variable. Generate synthetic input (data) x_i by sampling $n = 200$ points from a uniform distribution on the interval $[0, 6]$.

   ```
   x = np.random.uniform(0.0, 6.0, n)
   ```

   Generate the corresponding noisy outputs (labels) as $y_i = f_{\text{true}}(x_i) + \epsilon_i$, where the noise in each point is assumed to be from a normal (Gaussian) distribution of the form $\epsilon_i \sim N(0, 0.25)$.

   ```
   x = np.random.uniform(0, 6.0, n)
   y = np.sin(x) + np.sin(2*x) + np.random.normal(0.0, 0.25, n)
   ```

   Plot this data set $\mathcal{D} = (x_i, y_i)_{i=1}^n$ and the true function $f_{\text{true}}(x)$.

   ```
   x_true = np.arange(0.0, 6.0, 0.05)              # Uniform input data mesh
   y_true = np.sin(x_true) + np.sin(2 * x_true)
   plt.plot(x_true, y_true, linestyle='-', linewidth=3.0, color='k')

   plt.plot(x, y, marker='o', markerfacecolor='r', markeredgecolor='k',
      linestyle='none')
   ```

   c. (**Create Training and Test Sets**, 10 points) Recall that we want to build a model to **generalize well on future data**. In many applications, we do not have future data, so a common practice to evaluate our models is to **split** the overall data set $\mathcal{D}$ into a **training set** $\mathcal{D}_{\text{trn}}$ and a **hold-out test set** $\mathcal{D}_{\text{tst}}$. The idea is to train several models on the training set (without ever using the test set) and evaluate them on the test set (which is why we hold it out, and treat it like "future" data). Randomly split $\mathcal{D}$ into D_{trn}, containing 75% of the data and D_{tst}, containing 25% of the data.

d. (**Monomial Basis Functions**, 10 points) In order to learn nonlinear models using linear regression, we have to explicitly **transform the data** into a higher-dimensional space. The nonlinear hypothesis class we will consider is the set of d-degree polynomials of the form:

$$f(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_d x^d = [w_0, w_1, w_2 \ldots, w_d]^T \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^d \end{bmatrix}.$$

The monomials $\{1, x, x^2, \ldots, x^d\}$ are called **basis functions**, and each basis function x^q has a corresponding weight w_q associated with it, for all $q = 1, \ldots, d$. Write a Python function `transform_data(x, d)` that transforms the univariate input data `x` into (high-dimensional) multivariate data `phi` under the transformation $\phi(x) \rightarrow [1, x, x^2, \ldots, x^d]$. When this transformation is applied to each data point, it produces the **Vandermonde matrix**:

$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^d \\ 1 & x_2 & x_2^2 & \ldots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^d \end{bmatrix}$$

e. (**Training and Testing**, 10 points) Write a Python function `train_model(phi, y)` that takes a Vandermonde matrix as input and learns weights ordinary least squares. Hint: $\mathbf{w} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{y}$, and remember that in Python, `@` performs matrix multiplication, not $*$ (which performs element-wise multiplication). Write a function `test_model(phi, w)` that takes a Vandermonde matrix as input and evaluates the model using **mean squared error**. That is, $\epsilon_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\Phi_i)^2$.

f. (**Model Selection**, 10 points) Put everything together to learn models for $d = 5, \ldots, 15$ using the training data and evaluate the models using the test data. Create a table showing how the test error ϵ_{MSE} changes with the degree of the polynomial d. Which model is the best, according to the test data? What can you say about the bias-variance tradeoff in this nonlinear regression example?

```
for d in range(5, 16):                          # Iterate over poly. order
    phi_trn = transform_data(x_trn, d)          # Transform training data
    w = train_model(phi_trn, y_trn)             # Learn model on training data
    phi_tst = transform_data(x_tst, d)          # Transform test data
    e_mse = test_model(w, phi_tst, y_tst)       # Evaluate model on test data
```

g. (**Optional Bonus: Visualization**, 10 points) Plot the learned nonlinear regression models for $d = 5, \ldots, 16$.

2. (**Kernel Perceptron**, 25 points) In the previous question, we considered a nonlinear transformation using the Vandermonde basis. In this question, we explore other nonlinear transformations that lead to kernel functions.

a. (**2d to 3d**, 5 points) Let $\mathbf{x}$ and $\mathbf{z}$ be two points in $\mathbb{R}^2$. Consider the transformation from $\mathbb{R}^2$ to $\mathbb{R}^3$: $\phi([x_1, x_2]) \rightarrow [x_1^2, \sqrt{2}x_1x_2, x_2^2]$. Show that $\phi(\mathbf{x})^T\phi(\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^2$.

b. (**2d to 4d**, 5 points) Let $\mathbf{x}$ and $\mathbf{z}$ be two points in $\mathbb{R}^2$. Consider the transformation from $\mathbb{R}^2$ to $\mathbb{R}^4$: $\varphi([x_1, x_2]) \rightarrow [x_1^2, x_1x_2, x_2^2, x_2x_1]$. Show that $\varphi(\mathbf{x})^T\varphi(\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^2$.

The function $\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z})^2$ is an example of a **kernel function**, which can be used to **kernelize** linear hypotheses into nonlinear hypotheses. If a hypothesis can be written as a linear combination of the input data, and the predictions depend only on inner (dot) product computations, then the kernel function yields a nonlinear hypothesis **without explicit feature transformation**.

c. (**Linear Combination**, 5 points) Recall the stochastic gradient descent update for the perceptron at step i, where a data point $(\mathbf{x}_i, y_i)$ is used to compute $\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha_i y_i \mathbf{x}_i$ (where α_i is some update constant that depends on the gradient of the loss function and the learning rate). Unroll this expression recursively with $\mathbf{w}_0 = \mathbf{0}$ to show that

$$\mathbf{w}_{i+1} = \sum_{j=1}^{i} \alpha_j y_j \mathbf{x}_j.$$

This shows that the perceptron classifier is a **linear combination of data points**. What can you say about the training data points for whom $\alpha_j = 0$?

d. (**Kernel Perceptron**, 10 points) Now assume we use one of the feature transformations from parts in (a) or (b). For instance, if we use the transformation $\phi(\mathbf{x})$, we will have

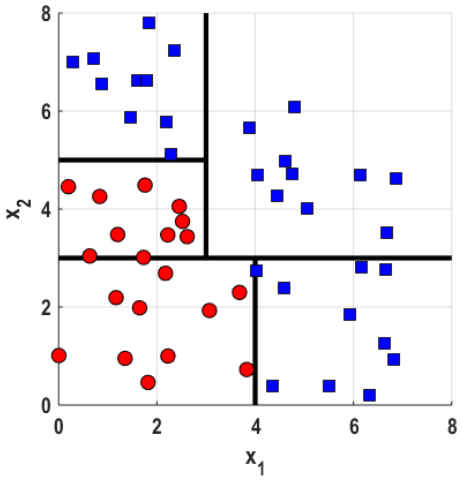
$$\mathbf{w}_{i+1} = \sum_{j=1}^{i} \alpha_j y_j \phi(\mathbf{x}_j).$$

Suppose we want to classify a new data point $\mathbf{x}_{\text{test}}$ using $\mathbf{w}_{i+1}$. Show that the expressions for $\mathbf{w}_{i+1}^T \phi(\mathbf{x}_{\text{test}})$ and $\mathbf{w}_{i+1}^T \varphi(\mathbf{x}_{\text{test}})$ are equal.

3. (**Decision Trees**, 25 points)

a. (**Interpreting a decision tree**, 5 points) Consider the decision boundary in Fig. 1 (left) and draw the equivalent decision tree. Red circles are Class $+1$ and blue squares, Class -1.

b. (**Visualizing a decision tree**, 5 points) Consider the decision in Fig. 1 (right) tree and draw the equivalent decision boundary. Make sure to label each decision region with the corresponding leaf node from the decision tree.



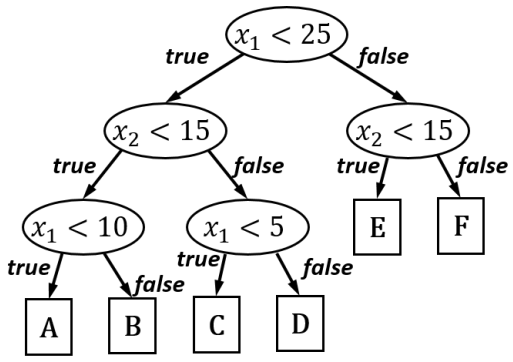


Figure 1: (**left**) a: Interpreting a decision tree; (**right**) b: Visualizing a decision tree.

c. (**Learning a decision tree**, 10 points) Consider the synthetic data set shown in the table below, where the task is to predict if a car is going to be bought based on three features: its color, type and origin. Draw the decision tree that will be learned if you use **accuracy** as the splitting criterion.

Color	Type	Origin	Buy?
Red	Sports	Domestic	Yes
Red	Sports	Domestic	No
Red	Sports	Domestic	Yes
Blue	Sports	Domestic	No
Blue	Sports	Imported	Yes
Blue	SUV	Imported	No
Blue	SUV	Imported	Yes
Blue	SUV	Domestic	No
Red	SUV	Imported	No
Red	Sports	Imported	Yes

d. (**Analyzing a decision tree**, 5 points) Analyze the overfitting nature of the decision tree.