CMU ONLINE
Carnegie Mellon University

Syllabus Overview Sample: Machine Learning & AI for Engineers

Program: AI Engineering for Mechanical Engineering

Faculty: Dr. Levent Burak Kara, George Tallman Ladd and Florence Barrett Ladd Professor, Mechanical Engineering & Associate Head for Academic Planning, College of Engineering

Course Number: 24887 **Units:** 12

Zoom Class: Weekday Evening, ET

Course Description

This course provides an introduction to the fundamental methods and algorithms at the core of modern machine learning. It also covers theoretical foundations as well as essential algorithms and practical techniques for supervised and unsupervised learning.

Course Learning Objectives

By the end of this course, students will be able to:

- Describe the fundamental problems addressed by machine learning and artificial intelligence.
- Design and implement various machine learning algorithms.
- Apply mathematical concepts (including linear algebra, probability, statistics, and optimization) in the context of machine learning algorithms.
- Formulate data-driven approaches to problems and communicate these solutions through implemented algorithms and write-ups.

Faculty Introduction



Dr. Levant Burak Kara is a Professor in the Department of Mechanical Engineering, with a courtesy appointment in the Robotics Institute.

His research focuses on developing new computational analysis, design, and manufacturing technologies with wide-ranging applications in the space of mechanical CAD, topology optimization, additive manufacturing, electronics design, and bio-engineering. By leveraging his experiences in

this realm he aims to bridge the gap between learning and application for students.

Kara is the recipient of National Science Foundation Career award and American Society of Mechanical Engineers Design Automation Society Young Investigator Award. At CMU, he teaches courses in AI and Machine learning, Engineering Design, and Linear Algebra and Vector Calculus. He earned his B.S. in Mechanical Engineering from the Middle East Technical University (1998), and his Ph.D. in Mechanical Engineering from Carnegie Mellon University (2005).

Course Topics

Topic	Learning Objectives
Introduction	• Recognize the difference between classification and regression using an NN sandbox • Identify a problem as a regression or a classification • Pose a regression problem as a classification problem • Pose a classification problem as a regression problem • Outline the three requirements for a set to be a metric space • Describe basic p-norms and distance and similarity measures and how they are used • Write Python functions from scratch to compute them • Write a KNN classifier and regressor from scratch in Python • Identify the inbuilt Python functions for KNN • Identify the issues of axis scaling
Linear Least Squares Regression	• Setup a regression problem using polynomial functions with linear weights • Derive the closed-form solution to the optimal weights • Apply non-polynomial transformations to least squares regression • Use a combination of sine and cosine to represent oscillation in data, with a given period • Implement the concept of standard gradient descent • Identify when iterative solvers are more useful for LSR than matrix factorization/pseudo-inverse • Differentiate between GD and SGD • Identify the rationale behind using regularization • Distinguish between the definitions of lasso (L1) and ridge (L2) regression, considering the 2D geometric interpretation of each • Derive a closed-form solution for ridge regression
Logistic Regression	• Explain the concept of logistic regression for classification problem • Interpret how the sigmoid function is used for converting a linear regression output into a probabilistic class label

	● Implement in Python gradient descent for solving the parameters of a logistic regression model with a linear boundary ● Modify the feature representation to make it applicable to non-linear decision boundaries ● Recognize situations in which modifying the loss function, gradient, and update rule is beneficial ● Adapt the logistic regression formulation to handle classification tasks with more than two labels with one versus one, one versus all, and multinomial logistic regression methods
Support Vector Machines	● Identify how support vector machines attempt to find linear boundaries for classification ● Write code to optimize a large margin classifier for linearly separable data ● Write code to optimize a soft margin classifier for inseparable data ● Adapt home-grown SVM classifier code to apply the kernel trick for data that is not linearly separable ● Identify several popular kernels and their uses ● Identify how using the kernel trick changes the optimization algorithm ● Identify how SVMs can be used for soft-margin regression ● Compare how the optimization problem and loss function are different from the classification model
Decision Trees	● Calculate decision tree Gini impurity vs. entropy ● Identify how reduction of entropy/impurity informs which attribute splits should occur on ● Identify how a decision tree is used for binomial or multinomial classification ● Identify how a regression decision can also be made by a decision tree ● Identify the problems with non-smoothness and overfitting that can occur using a decision tree for regression ● Identify how multiple decision trees can be combined in a random forest ● Explain how voting for classification and averaging for regression combine the results of all tree predictors ● Indicate the hyperparameters of a random forest
Feature Engineering	● Differentiate between normalization and standardization ● Identify the purpose and definition of log transformation ● Implement normalization, standardization, and log transformation in Python for a set of features in a given dataset ● Demonstrate the effectiveness of normalization, standardization, and log transformation, noting the subtle differences between their performances ● Identify why feature collinearity may necessitate feature pre-processing ● Calculate a correlation matrix ● Implement the variance inflation factor, and know how to use its results

	● Identify concepts of dimensionality reduction for feature engineering ● Implement in Python a regression or classification model that performs feature expansion ● Compare a regression or classification model that performs feature expansion to a model that does not use an expanded feature set
Neural Networks I	● Express a general neural network using its vector/matrix formulation ● Identify how a neural network can perform either regression or classification, and the differences between the implementations of each ● Identify how softmax creates a partition of unity on the output layer for classification problems ● Compare different activation functions and their purposes in different types of networks (Sigmoid, Tanh, ReLU, etc.)
Neural Networks II	● Derive the weight and bias updates for backpropagation through a simple network by hand ● Write Python code to perform backpropagation for networks of arbitrary size ● Interpret a learning curve ● Identify examples of overfitting and underfitting, and target these issues ● Write code for a forward pass of a PyTorch regression neural network ● Use built-in PyTorch functions to train with backpropagation ● Make plots showing loss curve and network predictions with ground truth
Error Metrics	● Implement the use cases for MAE, MSE, RMSE, MAPE, L1 loss, L2 loss, and R-squared ● Apply error metrics in the context of regression ● Calculate cross-entropy, precision, recall, F1 score, and a confusion matrix for a classification problem ● Implement error metrics to evaluate and compare different models taught so far in the class
Model Training	● Define bias ● Identify low- vs. high-bias models ● Define variance ● Identify low- vs. high-variance models ● Represent error as a sum of variance, squared bias, and irreducible error ● Relate sum of variance, squared bias, and irreducible error to underfitting and overfitting models ● Divide data into training, validation, and test sets ● Perform k-fold cross validation to select between different ML models ● Perform hyperparameter optimization

	● Use SciKit-Learn in Python to train multiple types of regression and classification models, such as least-squares regression or a neural network ● Use train-test splits to compare different models on unseen test data
Unsupervised Learning	● Identify the algorithm behind k-means clustering ● Implement in Python the algorithm behind k-means clustering ● Identify how initial centroid locations can affect the final clusters ● Identify how the hierarchical clustering algorithm works ● Differentiate between the hierarchical clustering algorithm and other clustering methods ● Identify how distance metric, linkage criteria, and number of clusters can affect results using the hierarchical clustering algorithm ● Implement hierarchical clustering using built-in sklearn Python functions ● Select an optimal cluster count using the elbow method ● Write code in Python to generate an elbow plot ● Introduce intuition behind the silhouette method ● Introduce the formulation of spectral clustering from graph representation through spectral decomposition ● Implement spectral clustering using built-in sklearn Python functions" ● Explain how Gaussian mixture models attempt to fit a probability distribution to each cluster ● Compare the advantages of GMMs with those of K-Means ● Develop the three core point types in DBSCAN
Dimensionality Reduction	● Implement PCA from scratch in Python ● Determine the ideal number of principal components to compress a dataset ● Use sklearn functions for PCA ● Implement PCA from scratch in Python in the k > n case ● Use the sklearn PCA model to handle the k > n case ● Design an AE neural network with an encoder-decoder structure for dimensionality reduction ● Use sklearn functions for non-linear embedding methods ● Use t-SNE to visualize data in a 2D or 3D manifold
Ensemble Methods	● Explain the concept of an ensemble method ● Differentiate between bagging, boosting, and stacking methods ● Use off-the-shelf implementations of bagging and boosting methods ● Introduce the concept of gradient boosting, with examples ● Develop extreme gradient boosting formulation ● Implement XGBoost with built-in Python functions
Introduction to Deep Learning	● Deep neural networks, convolutional neural networks (CNN), recurrent neural networks/LSTM

	● Autoencoders, variational autoencoders ● Generative adversarial networks ● Graph neural networks/graph convolutions ● Attention models, diffusion models, transformers/large language models

---

NOTE: This is a sample designed to provide an overview of topics covered in the course based on previous course syllabi. All students will be provided an updated and final syllabus at the start of the course. Faculty, content, and schedule are all subject to change.