Yonsei Univeristy: Intro to Machine Learning

Professor Woong Lim
Notes by Theodore Nguyen

Summer 2022

Contents

Week 1

1.1 Orientation

Goals

- Introduction to Machine Learning (ML).
- Learn the fundamentals of ML and apply select ML algorithms with big datasets.
- Review various techniques to build real-world AI applications.

Grading

- Daily and/or weekly mini-tasks: 70%
- Project (1-2 big tasks) and Participation 30%

Resources

- Intro to ML w/ Python by Muller & Guido [Muller Text]
- The 100-page ML book by Burkov [Burkov Text]
- ML for absolute beginners [Beginner Text]

1.2 Overview and Getting Started

Elements of Understanding and Doing ML

- The Problem: tasks, context, data, and potential solutions
- Python basics
- ML (or deep learning) algorithms
- Executing ML and evaluating efficacy

What is Machine Learning?

1. Building ML
    - Algorithm
    - Data (train, test)
2. ML Model
    - Input (parameters)
3. Output
    - Decision: Prediction/Solution

Muller Textbook pg. 1-4

- Machine learning is about extracting knowledge from data. Also known as predictive analytics or statistical learning.
- The most successful kinds of machine learning algorithms are those that automate decision-making processes by generalization from known examples.
  - In this setting, which is known as *supervised learning*, the user provides the algorithm with pairs of inputs and desired outputs, and the algorithm finds a way to produce the desired output given an input. In particular, the algorithm is able to create an output for an input it has never seen before without any help from a human.
- Machine learning algorithms that learn from input/output pairs are called supervised learning algorithms because a "teacher" provides supervision to the algorithms in the form of the desired outputs for each example that they learn from.
- Examples of supervised machine learning tasks:
  - Identifying the zip code from handwritten digits on an envelope
    - Input: scan of the handwriting. Output: actual digits in zip code.
  - Determining whether a tumor is benign based on a medical image
    - Input: image. Output: whether the tumor is benign.
  - Detecting fraudulent activity in credit card transactions
    - Input: a record of the credit card transaction. Output: whether it is likely to be fraudulent or not.
- In *unsupervised learning*, only the input data is known, and no known output data is given to the algorithm.
- Examples of unsupervised learning:
  - Identifying topics in a set of blog posts (might not know beforehand what these topics are, or how many topics there might be; hence there are no known outputs)
  - Segmenting customers into groups with similar preferences (don't know in advance what these groups might be; hence no known outputs)
  - Detecting abnormal access patterns to a website (only observe traffic to the website, so don't know what constitutes normal and abnormal behavior)
- For both supervised and supervised learning tasks, it is important to have a representation of input data that a computer can understand
- Data Tables:
  - Each data point that you want to reason about (each email, customer, transaction, etc.) is a **row**
    - Each entity/row is known as a <u>sample</u> (or data point)
  - Each property that describes the data point (age of a customer, amount of a transaction, etc.) is a **column**
    - Each column/property is a <u>feature</u>

Learning: Hexagon

1. AI: can find hexagons
2. ML: Write an algorithm and use train/test data to develop a model. There is "labeling" involved to help the algorithm to tell hexagons from other figures.
3. Deep Learning: Generally, there is no labeling involved. Feedback (right or wrong) helps improve performance.

Python-based ML environments

1. Package: Anaconda (about 3GB)
   (a) Jupyter Notebook (or Colab)
   (b) Spyder (debugging)
   (c) Otherlibraries
2. Data/code cloud
   (a) Github

Python Libraries

1. NumPy (numeric python): Vector, matrix
2. Pandas: spreadsheet data
3. Matplotlib, Seaborn: data visualization
4. Scikit-learn: ML package ("playbook")

1.3 Key Terms

Key Concepts

- Supervised, Unsupervised
- Classification, Regression
- Overfitting, Underfitting
- Confusion matrix
- Accuracy, precision, recall
- ML algorithms

Classification, Regression, Clustering, Underfitting, Overfitting

- **Classification**: Classify an item into a definitive category (supervised technique)
- **Regression**: Predicting a numerical value (mostly supervised)
- **Clustering**: Putting similar items together (unsupervised technique)
- **Underfitting**: few features
- **Overfitting**: too many features

Confusion Matrix: Evaluating ML Models

	A	**B**	**C**	**D**
A	14	2	2	2
B	2	18	0	0
C	4	2	12	2
D	1	1	2	16

- In 20 trials, an ML model predicted:
  - A correctly 14 times;

- B correctly 18 times;
- C correctly 12 times; and
- D correctly 16 times

$$\text{Accuracy} = \frac{14 + 18 + 12 + 16}{20 \times 4} = 75\%$$

Accuracy, Precision, Recall

ML Model	Disease	No Disease
Test Positive	True Positive (TP)	False Positive (FP)
Test Negative	False Negative (FN)	True Negative (TN)

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

f_1-score (or f-measure):

$$f_1\text{-score} = 2 \cdot \frac{\text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$$

(the harmonic mean of recall and position)

1.4 K-nearest neighbor (KNN) Basics I

k-Nearest Neighbors (Muller Text pg. 35)

- To make a prediction for a new data point, the algorithm finds the closet data points in the training dataset–its "nearest neighbors."
- **k-Neighbors classification**
  - In its simplest version, the k-NN algorithm only considers exactly one nearest neighbor, which is the cloest training data point to the point we want to make a prediction for. The prediction is then simply the known output for this training point.
  - Instead of considering only the closest neighbor, we can also consider an arbitrary number, k, of neighbors. We use *voting* to assign a label. For each test point, we count how many neighbors belong to class 0 and how many belong to class 1. We then assign the class that is more freuqent: in other words, the majority class among the k-nearest neighbors.

```python
# Split data into a training and a test set so we can evaluate generalization
    performance
from sklearn.model_selection import train_test_split
X, y = mglearn.datasets.make_forge()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

```python
# Fit the classifier using the training set
clf.fit(X_train, y_train)
```

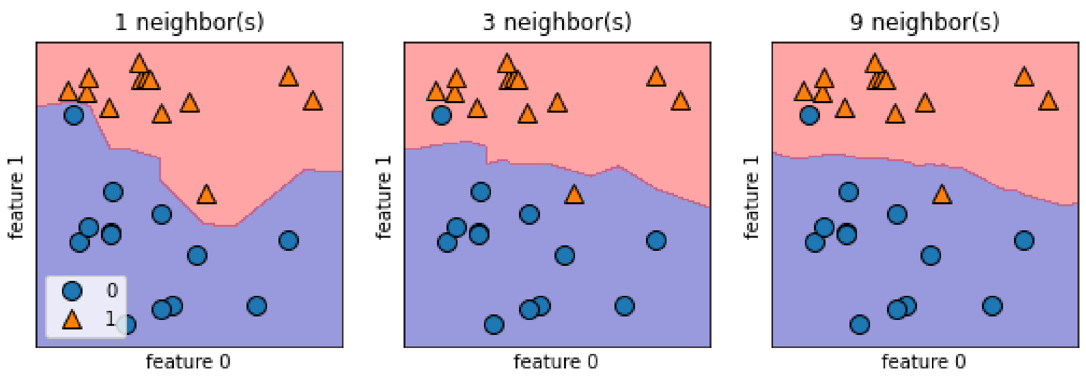
```python
# To make predictions on the test data, we call the predict method. For each data
    point in the test set, this computes its nearest neighbors in the training set
     and finds the most common class among these
print("Test set predictions: {}".format(clf.predict(X_test)))
```

```
# To evaluate how well the model generalizes, call the score method with the test
    data together with the test labels
print("Test set accuracy: {:.2f}".format(clf.score(X_test,y_test)))
```

- **Analyzing KNeighborsClassifier**
  - For two-dimensional datasets, we can illustrate the prediction for all possible test points in the xy-plane. We color the plane according to the class that would be assigned to a point in this region. This lets us view the *decision boundary*, which is the divide between where the algorithm assignes class 0 versus where it assigns class 1

```
# the following code produces visualizations of the decision boundaries for one,
    three, and nine neighbors
fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
    # the fit method returns the object self, so we can instantiate
    # and fit in one line
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{} neighbor(s)".format(n_neighbors))
    ax.set_xlabel("feature 0")
    ax.set_ylabel("feature 1")
axes[0].legend(loc=3)
```



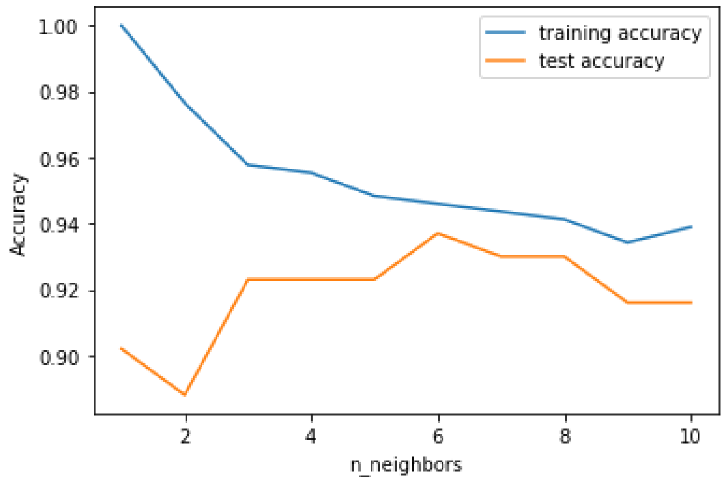
  - A smoother boundary corresponds to a simpler model.
  - Using few neighbors corresponds to high model complexity
  - Using many neighbors corresponds to low model complexity

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=66)

training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)
for n_neighbors in neighbors_settings:
    # build the model
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(clf.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(clf.score(X_test, y_test))
```

```python
plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```



- **Strengths, weaknesses, and parameters**
  - Two important parameters to `KNeighbors` classifier: the number of neighbors and how you measure distance between data points
  - When using the k-NN algorithm, it's important to preprocess your data

Concept of KNN

- Where am I?
- Where do I belong?
- Asking around - feature checking in the space (i.e., real world, vector space)

Cross Validation Scores

- `KNN_example_2.ipynb`
  - The `train.shape[0]` size is 80. We are taking half of the number of data, so `max_k_range` is 40. Since our range is from 3 to `max_k_range` (40) and we are skipping every 1, we have 37 k's.
  - Since the `cv` parameter is set to be 10, we get 10 cv scores for every k. Since we have 37 k's, we have $37 * 10 = 370$ cv scores. However, `cross_validation_scores` is comprised of the average cv scores, so we have 37 values in `cross_validation_scores`.
  - *When talking about cross validation score, we are referring to the mean cross validation score.*

Week 2

2.2 KNN 2

K-Fold CV

- $k = 5$ (**cv = 5**)

- Divide the training data into 5 big folds
- Divide each fold into 5 small folds
- Use 1 small fold for testing
- Repeat the procedure 5 times progressively

2.3 SVM 1

The basic steps of SVM in 2D

1. Select **two** hyperplanes with no points between them
2. Maximize their distance (the margin)
3. Add the average line (half way between the two hyperplanes). This will be the decision boundary.

Hard margins vs. Soft margins

- Soft margin: Penalty for non-separable cases

Nonlinear SVMs

- Linear Kernel
- Polynomial Kernel $d = 2$
- Polynomial Kernel $d = 5$

SVMs in 3D

- A hyperplane in $\mathbb{R}^2$ is a line
- A hyperplane in $\mathbb{R}^3$ is a plane

Kernel Tricks

- Non-linearity
- Higher dimensions
- Mapping to different (higher) dimensions
- **Choosing the right kernel**:
  - *Polynomial Kernel; Gaussian Radial Basis Function (RBF); Laplace RBF Kernel; Sigmoid Kernel*

Important Parameters

- **kernel: rbf** (the most common), n-degree **poly** or **sigmoid**
- **C: regularization parameter**
- **gamma**

Regularization Parameter ("C" in python)

- the SVM optimization to avoid non-separable points
- Lower $c \to$ wider margin
- Higher $c \to$ narrower margin

Gamma

- The gamma parameter indicates how far the model considers points to make decision boundaries
  - high gamma
  - low gamma

	Large Gamma	**Small Gamma**	**Large C**	**Small C**
Variance	Low	High (overfitting)	High (overfitting)	Low
Bias	High (underfitting)	Low	Low	High (underfitting)

Week 3

3.1 SVM Math 1

Derivation

1. "Naive" decision boundary: $x + y = 1x + 1y = 0$
   - Weight vector: $W = \begin{bmatrix}1 & 1\end{bmatrix}$
2. $$W^T \cdot X = 0$$ $$W = \begin{bmatrix}1 & 1\end{bmatrix}$$ $$X = \begin{bmatrix}x & y\end{bmatrix}$$ $$\begin{bmatrix}1 \\ 1\end{bmatrix} \begin{bmatrix}x & y\end{bmatrix} \equiv 0$$
3. Translating the line: $W^T \cdot X = b$
4. Finding best hyperplanes:
   (a) $W^T \cdot X = b + 1$
   (b) $W^T \cdot X = b - 1$
5. Rewriting the form:
   - Decision boundary: $W^T \cdot X - b = 0$
   - Hyperplane class $+1$: $W^T \cdot X - b = +1$
   - Hyperplnae class -1: $W^T \cdot X - b = -1$
6. Finding best margin (the width)
   - Decision boundary: $a_1x + a_2y - b = 0$
   - Hyperplane class $+1$: $a_1x + a_2y - b = +1$
   - Hyperplane class -1: $a_1x + a_2y - b = -1$
   - $W = \begin{bmatrix}a_1 & a_2\end{bmatrix}$
     - Recall: Distance from a point to line

$$d = \frac{|ax_1 + by_1 + c|}{\sqrt{(a^2 + b^2)}}$$

where the line is given by $ax + by + c = 0$, and the point of interest is (x_1, y_1).

- So, we have
$$d = \frac{|a_1x_1 + a_2y_2 - b|}{\sqrt{a_1^2 + a_2^2}}.$$
- We know that
$$a_1x_1 + a_2y_2 - b = -1,$$
so
$$\begin{aligned} d &= \frac{|-1|}{\sqrt{(a_1^2 + a_2^2)}} \\ &= \frac{1}{||W||.} \end{aligned}$$
This is the distance from one parallel hyperplane to the decision boundary. Thus, the distance from one parallel hyperplane to the other is $\frac{2}{||W||}$.

- Trivial and key insights
  - W = weight
  - b = bias
  - A "parallel" hyperplane *must* contain at least one data point (i.e. support vector)
  - In the algorithm of SVM, the two parallel hyperplanes should be equidistant from the decision boundary
    - The margin is set up to handle (i.e. penalize) the error points mathematically.
    - Optimally "separating" Hyperplane: the hyperplanes for which the margin of separation "2d" is maximized.

3.1 SVM Math 2

More General Derivation

1. Finding best hyperplanes
   - Decision boundary line: $ax + by = c$
   - Parallel hyperplane above: $ax + by = c + f$
   - Parallel hyperplane below: $ax + by = c - f$
2. Suppose our support vectors are $(2, 3)$ and $(5, 5)$.
3. The distance from $(2, 3)$ to the decision boundary line $ax + by = c$ is
$$D = \frac{|2a + 3b - c|}{\sqrt{a^2 + b^2}}.$$
   - We know:
     - The line $ax + by = c - f$ contains $(2, 3)$. Then $2a + 3b = c - f$.
     - The line $ax + by = c + f$ contains $(5, 5)$. Then $5a + 5b = c + f$.
4. Add our two equations:
$$7a + 8b = 2c \iff c = 3.5a + 4b.$$
5. Plug into D:
$$\begin{aligned} D &= \frac{|2a + 3b - (3.5a + 4b)|}{\sqrt{a^2 + b^2}} \\ &= \frac{|-1.5a - b|}{\sqrt{a^2 + b^2}} \end{aligned}$$

6. Maximize D using Wolfram Alpha Maximum Calculator. Let $D = f(a,b) = \frac{|-1.5a-b|}{\sqrt{a^2+b^2}}$. From the calculator, we get $y = \frac{2x}{3} \implies b = \frac{2a}{3} \iff a = \frac{3b}{2}$.

7. Put it all together. We have

$$a = \frac{3b}{2}$$
$$c = 3.5a + 4b.$$

Thus, plugging these two substitutions into our decision boundary line $ax + by = c$, we get

$$\left(\frac{3}{2}b\right) x + by = 3.5\left(\frac{3}{2}b\right) + 4b$$
$$y = -1.5x + 9.25 \iff 1.5x + y = 9.25$$

where $a = 1.5$, $b = 1$, and $c = 9.25$.

- To find the equations of the parallel hyperplanes, namely $ax + by = c \pm f$, plug in a, b, c, and the support vector coordinates (x, y), and solve for f.
- Additional notes:
  - In the above example, we used the distance from $(2,3)$ to the decision boundary. By symmetry of the other support vector $(5,5)$ (*since parallel hyperplanes should be equidistant from the decision boundary*), we could have also maximized the distance from $(5,5)$ to the decision boundary.
  - The distance from one parallel hyperplane to the other is equal to the distance from one support vector to the other.
  - The decision boundary line is perpendicular to the line connecting the two support vectors. That is, the slope of the decision boundary line is the opposite reciprocal of the slope of the line connecting the two support vectors.
  - The physical space between the two parallel hyperplanes is called the *marginal space*

3.2 Data Scaling

- Why Standardization?
  - Resets all features to a common scale keeping fundamental properties of data such as roder, comparative magnitude, etc.
- Scikit-learn Data Scalers
  - **StandardScaler**
  - **MinMaxScaler**
  - MaxAbsScaler
  - **RobustScaler**
  - Normalizer
- *For training data: do fitting first and do transforming later*
- *For testing data:* `transform()`

`fit()` vs. `transform()`

- `fit()`: Calculating the mean and variance of each feature data
- `transform()`: Using the mean and variance from "fitting" and conduct scaling

StandardScaler

- Standardize the data
- = Reformat (scaling) data (feature) to have the **Standard Normal Distribution** with the mean = 0 and standard deviation = 1

MinMaxScaler

- Reformat (scaling) data (feature) to have the minimum value = 0 and maximum = 1
- Spread out the data over $[0, 1]$

RobustScaler

- Reformat (scaling) data (feature) to have the median = 0 and Interquartile range (IQR) = 1
- Formula: $\frac{\text{“feature”} - \text{median value}}{\text{IQR}}$ where IQR $= Q_3 - Q_1$
- Not sensitive to outliers