**Decision Trees**[**¶**](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/2_Decision_Trees_%26_ML_Fundamentals/decision-trees.ipynb#Final-comments,-summary,-and-reflection)

What did we learn today?

* There is a lot of terminology and jargon used in ML. Some of the basic terminology includes:
  + Features, target, examples, training
  + Supervised vs. Unsupervised machine learning
  + Classification and regression
  + Accuracy and error
  + Parameters and hyperparameters
  + Decision boundary
* Baselines and steps to train a supervised machine learning model
  + Baselines serve as reference points in ML workflow.
* Decision trees
  + are models that make predictions by sequentially looking at features and checking whether they are above/below a threshold
  + learn a hierarchy of if/else questions, similar to questions you might ask in a 20-questions game.
  + learn axis-aligned decision boundaries (vertical and horizontal lines with 2 features)
  + One way to control the complexity of decision tree models is by using the depth hyperparameter (max\_depth in sklearn).

### **Summary of train, validation, test, and deployment data**

|  | **fit** | **score** | **predict** |
| --- | --- | --- | --- |
| Train | ✔️ | ✔️ | ✔️ |
| Validation |  | ✔️ | ✔️ |
| Test |  | once | once |
| Deployment |  |  | ✔️ |

You can typically expect 𝐸𝑡𝑟𝑎𝑖𝑛<𝐸𝑣𝑎𝑙𝑖𝑑𝑎𝑡𝑖𝑜𝑛<𝐸𝑡𝑒𝑠𝑡<𝐸𝑑𝑒𝑝𝑙𝑜𝑦𝑚𝑒𝑛𝑡

### **Questions on generalization and data splitting**[**¶**](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/2_Decision_Trees_%26_ML_Fundamentals/ml-fundamentals.ipynb#%E2%9D%93%E2%9D%93-Questions-on-generalization-and-data-splitting)

1. A decision tree model with no depth is likely to perform very well on the deployment data.

2. Data splitting helps us generalize our model better.

3. Deployment data is used at the very end and only scored once.

4. Validation data could be used for hyperparameter optimization.

1. False
2. False. Data splitting helps us **assess** how well our model would generalize.
3. False. You cannot score on the deployment data as there are no labels available.
4. True

### **The "fundamental tradeoff" of supervised learning:**

**As you increase model complexity,**𝐸train **tends to go down but**𝐸valid−𝐸train**tends to go up.**

The fundamental trade-off is also called the bias/variance tradeoff in supervised machine learning.

**Bias** : the tendency to consistently learn the same wrong thing (high bias corresponds to underfitting)

**Variance** : the tendency to learn random things irrespective of the real signal (high variance corresponds to overfitting)

**State whether True/False.**

1. In supervised learning, the training error is always lower than the validation error.
2. The fundamental tradeoff of ML states that as training error goes down, validation error goes up.
3. More "complicated" models are more likely to overfit than "simple" ones.
4. If we had an infinite amount of training data, overfitting would not be a problem.
5. If our training error is extremely low, we are likely to be overfitting.
6. False
7. False
8. True
9. True
10. True

### **Pros of k-NNs for supervised learning**

* Easy to understand, interpret.
* Simple hyperparameter 𝑘k (n\_neighbors) controlling the fundamental tradeoff.
* Can learn very complex functions given enough data.
* Lazy learning: Takes no time to fit

### **Cons of k-NNs for supervised learning**

* Can be potentially be VERY slow during prediction time, especially when the training set is very large.
* Often not that great test accuracy compared to the modern approaches.
* It does not work well on datasets with many features or where most feature values are 0 most of the time (sparse datasets).

### **Parametric vs non parametric**

* You might see a lot of definitions of these terms.
* A simple way to think about this is:
  + do you need to store at least 𝑂(𝑛)O(n) worth of stuff to make predictions? If so, it's non-parametric.
* Non-parametric example: 𝑘k-NN is a classic example of non-parametric models.
* Parametric example: decision stump
* If you want to know more about this, find some reading material [here](https://www.cs.ubc.ca/~schmidtm/Courses/340-F16/L6.pdf), [here](http://mlss.tuebingen.mpg.de/2015/slides/ghahramani/gp-neural-nets15.pdf), and [here](https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/).
* By the way, the terms "parametric" and "non-paramteric" are often used differently by statisticians, see [here](https://help.xlstat.com/s/article/what-is-the-difference-between-a-parametric-and-a-nonparametric-test?language=en_US) for more...

### **Curse of dimensionality**

* Affects all learners but especially bad for nearest-neighbour.
* 𝑘k-NN usually works well when the number of dimensions 𝑑d is small but things fall apart quickly as 𝑑d goes up.
* If there are many irrelevant attributes, 𝑘k-NN is hopelessly confused because all of them contribute to finding similarity between examples.
* With enough irrelevant attributes the accidental similarity swamps out meaningful similarity and 𝑘k-NN is no better than random guessing.

**Summary KNN**

* We have KNNs as a new supervised learning technique in our toolbox.
* These are analogy-based learners and the idea is to assign nearby points the same label.
* Unlike decision trees, all features are equally important.
* Both can be used for classification or regression (much like the other methods we've seen).

**What did we learn today (preprocessing)?**

* Motivation for preprocessing
* Common preprocessing steps
  + Imputation
  + Scaling
  + One-hot encoding
* Golden rule in the context of preprocessing
* Building simple supervised machine learning pipelines using sklearn.pipeline.make\_pipeline

**Summary of linear models**

* Linear regression is a linear model for regression whereas logistic regression is a linear model for classification.
* Both these models learn one coefficient per feature, plus an intercept.
* The main hyperparameter is the "regularization" hyperparameter controlling the fundamental tradeoff.
  + Logistic Regression: C
  + Linear SVM: C
  + Ridge: alpha

### Strengths of linear models[¶](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/6_Linear_Models/linear-models.ipynb#Strengths-of-linear-models)

* Fast to train and predict
* Scale to large datasets and work well with sparse data
* Relatively easy to understand and interpret the predictions
* Perform well when there is a large number of features

### **Advantages of RandomizedSearchCV[¶](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/7_hyper-parameters/hyperparameter-optimization.ipynb" \l "Advantages-of-RandomizedSearchCV)**

* Faster compared to GridSearchCV.
* Adding parameters that do not influence the performance does not affect efficiency.
* Works better when some parameters are more important than others.
* In general, I recommend using RandomizedSearchCV rather than GridSearchCV.

#### **Automated hyperparameter optimization**

* Advantages
  + reduce human effort
  + less prone to error and improve reproducibility
  + data-driven approaches may be effective
* Disadvantages
  + may be hard to incorporate intuition
  + be careful about overfitting on the validation set

## Confusion matrix[¶](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/8_classification_metrics/classification-metrics.ipynb#Confusion-matrix)

One way to get a better understanding of the errors is by looking at

* false positives (type I errors), where the model incorrectly spots examples as fraud
* false negatives (type II errors), where it's missing to spot fraud examples

### What is "positive" and "negative"?[¶](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/8_classification_metrics/classification-metrics.ipynb#What-is-%22positive%22-and-%22negative%22?)

* Two kinds of binary classification problems
  + Distinguishing between two classes
  + Spotting a class (spot fraud transaction, spot spam, spot disease)
* In case of spotting problems, the thing that we are interested in spotting is considered "positive".
* Above we wanted to spot fraudulent transactions and so they are "positive".

## Precision, recall, f1 score

* We have been using .score to assess our models, which returns accuracy by default.
* Accuracy is **misleading** when we have class imbalance.
* We need other metrics to assess our models.
* We'll discuss three commonly used metrics which are based on confusion matrix:
  + recall
  + precision
  + f1 score
* Note that these metrics will only help us assessing our model.
* Later we'll talk about a few ways to address class imbalance problems.

### Recall

Among all positive examples, how many did you identify?

𝑟𝑒𝑐𝑎𝑙𝑙=𝑇𝑃/(𝑇𝑃+𝐹𝑁)=𝑇𝑃/#𝑜𝑟𝑖𝑔𝑖𝑛𝑎𝑙 𝑝𝑜𝑠𝑖𝑡𝑖𝑣𝑒𝑠

### Precision

Among the positive examples you identified, how many were actually positive?

𝑝𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛=𝑇𝑃/(𝑇𝑃+𝐹𝑃)=𝑇𝑃/#𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑 𝑝𝑜𝑠𝑖𝑡𝑖𝑣𝑒𝑠

### F1-score

* F1-score combines precision and recall to give one score, which could be used in hyperparameter optimization, for instance.
* F1-score is a harmonic mean of precision and recall.

𝑓1=2×[(𝑝𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛×𝑟𝑒𝑐𝑎𝑙𝑙) / (𝑝𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛+𝑟𝑒𝑐𝑎𝑙𝑙) ]

### **Macro average**

* You give equal importance to all classes and average over all classes.
* For instance, in the example above, recall for non-fraud is 1.0 and fraud is 0.63, and so macro average is 0.81.
* More relevant in case of multi-class problems.

### **Weighted average**

* Weighted by the number of samples in each class.
* Divide by the total number of samples.

### **Interim summary**[¶](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/8_classification_metrics/classification-metrics.ipynb#Interim-summary)

* Accuracy is misleading when you have class imbalance.
* A confusion matrix provides a way to break down errors made by our model.
* We looked at three metrics based on confusion matrix:
  + precision, recall, f1-score.
* Two kinds of binary classification problems
  + Distinguishing between two classes (e.g., dogs vs. cats)
  + Spotting a class (e.g., spot fraud transaction, spot spam)

### **Operating point**[**¶**](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/8_classification_metrics/classification-metrics.ipynb#Operating-point)

* Now our recall for "fraud" class is >= 0.75.
* Setting a requirement on a classifier (e.g., recall of >= 0.75) is called setting the **operating point**.
* It's usually driven by business goals and is useful to make performance guarantees to customers.

### **Precision/Recall tradeoff**

* But there is a trade-off between precision and recall.
* If you identify more things as "fraud", recall is going to increase but there are likely to be more false positives.

### **AP score**

* Often it's useful to have one number summarizing the PR plot (e.g., in hyperparameter optimization)
* One way to do this is by computing the area under the PR curve.
* This is called **average precision** (AP score)
* AP score has a value between 0 (worst) and 1 (best).

### **AP vs. F1-score**

It is very important to note this distinction:

* F1 score is for a given threshold and measures the quality of predict.
* AP score is a summary across thresholds and measures the quality of predict\_proba.

### **Receiver Operating Characteristic (ROC) curve**

* Another commonly used tool to analyze the behavior of classifiers at different thresholds.
* Similar to PR curve, it considers all possible thresholds for a given classifier given by predict\_proba but instead of precision and recall it plots false positive rate (FPR) and true positive rate (TPR or recall).

### **Handling imbalance**

Can we change the model itself rather than changing the threshold so that it takes into account the errors that are important to us?

There are two common approaches for this:

* **Changing the data (optional)** (not covered in this course)
  + Undersampling
  + Oversampling
    - Random oversampling
    - SMOTE
* **Changing the training procedure**
  + class\_weight

**Scoring metrics Regressor**

* 𝑅2 is the default .score(), it is unitless, 0 is bad, 1 is best
* MSE (mean squared error) is in units of target squared, hard to interpret; 0 is best
* RMSE (root mean squared error) is in the same units as the target; 0 is best
* MAPE (mean average percent error) is unitless; 0 is best, 100 is bad

**Ensembles**

* **Ensembles** are models that combine multiple machine learning models to create more powerful models.

There are various ways of Ensemble learning but two of them are widely used:

* Bagging
* Boosting

### Bagging (Bootstrap Aggregating)

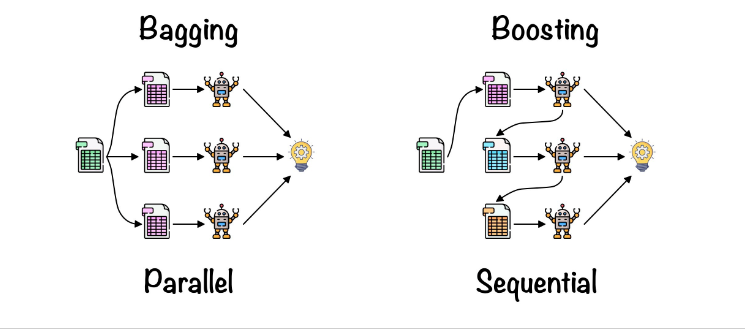
In a nutshell, BAGGING comes from two words “Bootstrap” & “Aggregation”.

* Bootstrap refers to subsetting the data
* Aggregation refer to aggregating the results that we will be getting from different models

The Random Forest model uses Bagging, where decision tree models with higher variance are present. It makes random feature selection to grow trees. Several random trees make a Random Forest

### Boosting

Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model. The succeeding models are dependent on the previous model and hence work sequentially. It fits a sequence of weak learners − models that are only slightly better than random guessings, such as small decision trees − to weighted versions of the data. More weight is given to examples that were misclassified by earlier rounds/iterations.



### **Tree-based ensemble models**

* A number of ensemble models in ML literature.
* Most successful ones on a variety of datasets are tree-based models.
* We'll briefly talk about two such models:
  + Random forests
  + Gradient boosted trees
* We'll also talk about averaging and stacking.

### **Tree-based models**

* Decision trees models are
  + Interpretable
  + They can capture non-linear relationships
  + They don't require scaling of the data and theoretically can work with categorical features.
* But with single decision trees are likely to overfit.
* Key idea: Combine multiple trees to build stronger models.
* These kinds of models are extremely popular in industry and machine learning competitions

## Random forests

### General idea

* A single decision tree is likely to overfit
* Use a collection of diverse decision trees
* Each tree overfits on some part of the data but we can reduce overfitting by averaging the results
  + can be shown mathematically

### How do they work?

* Decide how many decision trees we want to build
  + can control with n\_estimators hyperparameter
* fit a diverse set of that many decision trees by **injecting randomness** in the classifier construction
* predict by voting (classification) or averaging (regression) of predictions given by individual models

### The random forests classifier[¶](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/10_ensemble_learning/ensembles.ipynb#The-random-forests-classifier)

* Create a collection (ensemble) of trees. Grow each tree on an independent bootstrap sample from the data.
* At each node:
  + Randomly select a subset of features out of all features (independently for each node).
  + Find the best split on the selected features.
  + Grow the trees to maximum depth.
* Prediction time
  + Vote the trees to get predictions for new example.

### Some important hyperparameters:[¶](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/10_ensemble_learning/ensembles.ipynb#Some-important-hyperparameters:)

* n\_estimators: number of decision trees (higher = more complexity)
* max\_depth: max depth of each decision tree (higher = more complexity)
* max\_features: the number of features you get to look at each split (higher = more complexity)

#### Number of trees and fundamental trade-off

* Above: seems like we're beating the fundamental "tradeoff" by increasing training score and not decreasing validation score much.
* This is the promise of ensembles, though it's not guaranteed to work so nicely.

More trees are always better! We pick less trees for speed.

### Strengths and weaknesses

* Usually one of the best performing off-the-shelf classifiers without heavy tuning of hyperparameters
* Don't require scaling of data
* Less likely to overfit
* Slower than decision trees because we are fitting multiple trees but can easily parallelize training because all trees are independent of each other
* In general, able to capture a much broader picture of the data compared to a single decision tree.

### Weaknesses

* Require more memory
* Hard to interpret
* Tend not to perform well on high dimensional sparse data such as text data

## Gradient boosted trees

Another popular and effective class of tree-based models is gradient boosted trees.

* No randomization.
* The key idea is combining many simple models called weak learners to create a strong learner.
* They combine multiple shallow (depth 1 to 5) decision trees
* They build trees in a serial manner, where each tree tries to correct the mistakes of the previous one.

### Important hyperparameters

* n\_estimators
  + control the number of trees to build
* learning\_rate
  + controls how strongly each tree tries to correct the mistakes of the previous trees
  + higher learning rate means each tree can make stronger corrections, which means more complex model

### **Stacking**[**¶**](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/10_ensemble_learning/ensembles.ipynb#Stacking)

* Another type of ensemble is stacking.
* Instead of averaging the outputs of each estimator, instead use their outputs as inputs to another model.
* By default for classification, it uses logistic regression.
  + We don't need a complex model here necessarily, more of a weighted average.
  + The features going into the logistic regression are the classifier outputs, not the original features!
  + So the number of coefficients = the number of base estimators!

**Summary**[**¶**](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/10_ensemble_learning/ensembles.ipynb#Summary)

* You have a number of models in your toolbox now.
* Ensembles are usually pretty effective.
  + Tree-based classifiers are particularly popular and effective on a wide range of problems.
  + But they trade off code complexity and speed for prediction accuracy.
  + Don't forget that hyperparameter optimization multiplies the slowness of the code!
* Stacking is a bit slower than voting, but generally higher accuracy.
  + As a bonus, you get to see the coefficients for each base classifier.
* All the above models have equivalent regression models.

**Types of machine learning**

Recall the typical learning problems we discussed at the beginning of the course.

* Supervised learning ([Gmail spam filtering](https://support.google.com/a/answer/2368132?hl=en))
  + Training a model from input data and its corresponding targets to predict targets for new examples. (571, 572, 573)
* **Unsupervised learning** (this lecture) ([Google News](https://news.google.com/))
  + Training a model to find patterns in a dataset, typically an unlabeled dataset.
* Reinforcement learning ([AlphaGo](https://deepmind.com/research/case-studies/alphago-the-story-so-far))
  + A family of algorithms for finding suitable actions to take in a given situation in order to maximize a reward.
* Recommendation systems ([Amazon item recommendation system](https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf))
  + Predict the "rating" or "preference" a user would give to an item.

### Unsupervised learning

* Training data consists of observations (𝑋X) without any corresponding targets.
* Unsupervised learning could be used to group similar things together in 𝑋X.

# **Principal Component Analysis**

In this section, we explore what is perhaps one of the most broadly used of unsupervised algorithms, principal component analysis (PCA). PCA is fundamentally a dimensionality reduction algorithm, but it can also be useful as a tool for visualization, for noise filtering, for feature extraction and engineering, and much more

### PCA intuition

Principal component analysis is a fast and flexible unsupervised method for dimensionality reduction in data.

### PCA as dimensionality reduction[¶](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/11_unsupervised_learning_%26_PCA/unsupervised-learning-pca.ipynb#PCA-as-dimensionality-reduction)

Using PCA for dimensionality reduction involves zeroing out one or more of the smallest principal components, resulting in a lower-dimensional projection of the data that preserves the maximal data variance.

Here is an example of using PCA as a dimensionality reduction transform:

## Summary

In this section we have discussed the use of principal component analysis for dimensionality reduction, for visualization of high-dimensional data, for noise filtering, and for feature selection within high-dimensional data. Because of the versatility and interpretability of PCA, it has been shown to be effective in a wide variety of contexts and disciplines. Given any high-dimensional dataset, I tend to start with PCA in order to visualize the relationship between points (as we did with the digits), to understand the main variance in the data (as we did with the eigenfaces), and to understand the intrinsic dimensionality (by plotting the explained variance ratio). Certainly PCA is not useful for every high-dimensional dataset, but it offers a straightforward and efficient path to gaining insight into high-dimensional data.

PCA's main weakness is that it tends to be highly affected by outliers in the data. For this reason, many robust variants of PCA have been developed, many of which act to iteratively discard data points that are poorly described by the initial components. Scikit-Learn contains a couple interesting variants on PCA, including RandomizedPCA and SparsePCA, both also in the sklearn.decomposition submodule. RandomizedPCA, which we saw earlier, uses a non-deterministic method to quickly approximate the first few principal components in very high-dimensional data, while SparsePCA introduces a regularization term that serves to enforce sparsity of the components.

## Clustering motivation

### Why clustering?

* Most of the data out there is unlabeled.
* Getting labeled training data is often difficult, expensive, or simply impossible in some cases.
* Can we extract some useful information from unlabeled data?
* The most intuitive way is to group similar examples together to get some insight into the data even though we do not have the targets.

**Clustering** is the task of partitioning the dataset into groups called clusters.

The goal of clustering is to discover underlying groups in a given dataset such that:

* examples in the same group are as similar as possible;
* examples in different groups are as different as possible.

### Meaningful groups in clustering

* In clustering, meaningful groups are dependent on the **application**.
* It usually helps if we have some prior knowledge about the data and the problem.
* This makes it hard for us to objectively measure the quality of a clustering algorithm (or think about "true" clusters).

### Common applications: Data exploration

Although there is no notion of the "right" answer, we might still get something useful out of clustering. There are a number of common applications for clustering.

* Summarize or compress data.
* Partition the data into groups before further processing.
* For instance, you could use it in supervised learning setting as follows. Carry out clustering and examine performance of your model on individual clusters. If the performance is lower on a particular cluster, you could either try building a separate model for that cluster and improve the overall performance of your supervised model.

### Common applications: Customer segmentation

* Understand landscape of the market in businesses and craft targeted business or marketing strategies tailored for each group.

### Similarity and distances[¶](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/12_clustering_%26_kmeans/clustering-kmeans.ipynb#Similarity-and-distances)

* Clustering is based on the notion of similarity or distances between points.
* How do we determine similarity between points in a multi-dimensional space?
* Can we use something like 𝑘k-neighbours for similarity?
  + Yes! That's a good start!
  + With 𝑘k-neighbours we used Euclidean distances to find nearby points.
  + We can use the same idea for clustering!

## K-Means clustering

### K-Means clustering

One of the most commonly used clustering algorithm.

**Input**

* X →→ a set of data points
* K (or 𝑘k or n\_clusters) →→ number of clusters

**Output**

* K clusters (groups) of the data points

### K-Means clustering algorithm

**Input**: Data points X and the number of clusters K

**Initialization**: K initial centers for the clusters

**Iterative process**:

repeat

* Assign each example to the closest center.
* Estimate new centers as average of observations in a cluster.

until **centers stop changing** or **maximum iterations have reached**.

### Hyperparameter tuning for K

* K-Means takes K (n\_clusters in sklearn) as a hyperparameter. How do we pick K?
* In supervised setting we carried out hyperparameter optimization based on cross-validation scores.
* Since in unsupervised learning we do not have the target values, it becomes difficult to objectively measure the effectiveness of the algorithms.
* There is no definitive approach.
* However, some strategies might be useful to help you determine K.

### **Important points to remember**[**¶**](http://localhost:8888/notebooks/PythonCode/ECE6420/Notes/12_clustering_%26_kmeans/clustering-kmeans.ipynb#Important-points-to-remember)

**K-Means**

* It requires us to specify the number of clusters in advance.
* Each example is assigned to one (and only one) cluster.
* The labels provided by the algorithm have no actual meaning.
* The centroids live in the same space as of the dataset but they are **not** actual data points, but instead are average points.
* It always converges. Convergence is dependent upon the initial centers and it may converge to a sub-optimal solution.
* Two ways to provide insight into how many clusters are reasonable for the give problem are: **the Elbow method** and **the Silhouette method**.
* Some applications of K-Means clustering include data exploration, feature engineering, customer segmentation, and document clustering.
* It takes fair amount of manual effort and domain knowledge to interpret clusters returned by K-Means.