Notes for Coursera course: Applied Machine Learning in Python

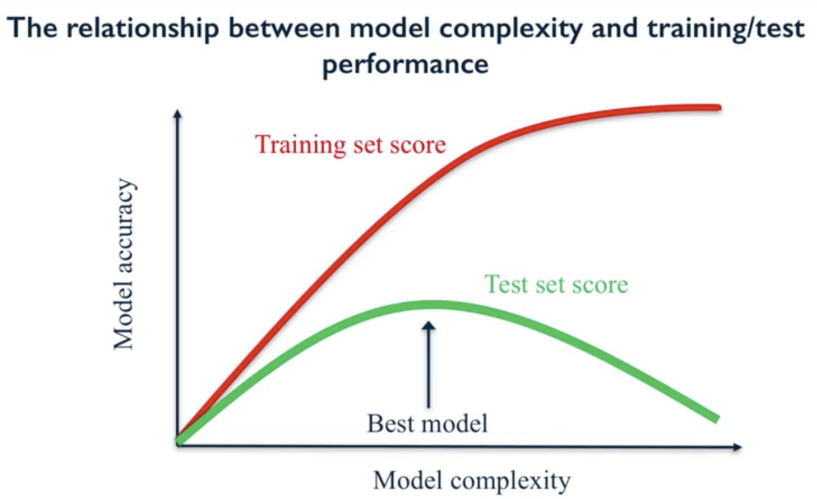
Link to course

Assignments

- Assignment 1
- Assignment 2

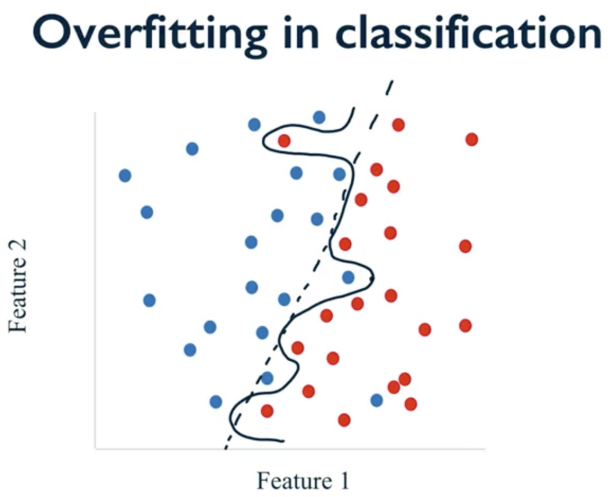
Week 2

- It is important to split the data set into training and testing data sets to properly analyze your model.
    - Only use the training data set for training
    - Only use the testing data set for assessing how good your model is
    - The ability of a model to predict results for data that it didn't see before is called "generalization"
- The following plot shows how increasing the complexity of the model (ex, no. of neighbors for knn) does not always result in a more accurate model



- "**Overfitting**" is when we want to train a complex model, but we don't have enough training data
    - due to that, the model will learn only local patterns, and won't be able to generalize to more global patterns since there is not enough training data points
    - in this case, the accuracy of the training set will be optimistic, and in reality the model will not be able to generalize to new data correctly

- plots below show an example of overfitting



- "**Underfitting**" is when we we have a very simplistic model for the data we have
  - in this case, the model will not perform well even for the training data
- "**K-Nearest Neighbors**"
  - Learns the training data set
  - For each data point to predict, finds the k most similar neighbors
  - Predicts the final label based on the labels of these neighbors (usually by majority rule)
  - Most important parameters:
    - n_neighbors (default = 5)
    - distance function between data points (default = minkowski distance with power parameter p = 2 ie Euclidean distance)
- "**Linear Regression**"
  - Predicted value is a weighted sum of all the feature values, plus a constant
  - There are multiple ways to learn the weights
    - Least Squares (default)
    - Ridge Regression
      - same as least squares but adds a penalty for large weights
        - this means that a model with smaller weights is preferred to one with larger weights
      - this is called regularization and usually prevents over-fitting
        - controlled by a parameter called alpha
        - higher alpha means more regularization (and smaller weights)

$$RSS_{RIDGE}(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2 + \alpha \sum_{\{j=1\}}^{p} w_j^2$$

      - Regularization helps more when you have a small amount of data and lots of features. The more the data, the less useful the regularization.
      - **IMPORTANT**: for ridge regression to work properly, all features need to be normalized to the same scale
      - In Turi, ridge regression alpha is the same as the l2_penalty parameter.
      - When to use:
        - when lots of variables have small or medium effect on your output variable
    - Lasso regression:

- Similar to ridge in that it adds a penalty on large weights

$$RSS_{LASSO}(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2 + \alpha \sum_{\{j=1\}}^{p} |w_j|$$

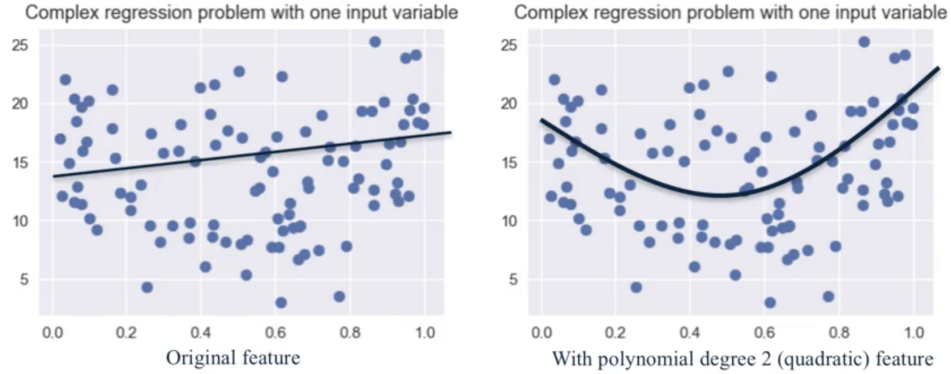
- Can have very different results from ridge regression
- A subset of the coefficients are forced to be precisely 0
- In Turi, lasso regression alpha is the same as the l1_penalty parameter
- When to use:
  - when a few variables have a medium or large effect on your output variable

- Polynomial features:
  - Create a combination of the given features and solve using linear regression
  - For example, given two features, x0 and x1, create new features x0^2, x0x1, x1^2, then solve using LR

$$\boldsymbol{x} = (x_0, x_1) \Rightarrow \boldsymbol{x}' = (x_0, x_1, x_0^2, x_0 x_1, x_1^2)$$

$$\hat{y} = \hat{w}_0 x_0 + \hat{w}_1 x_1 + \hat{w}_{00} x_0^2 + \hat{w}_{01} x_0 x_1 + \hat{w}_{11} x_1^2 + b$$

  - This adds the effect of interactions between the original feature variables that we didn't have before
  - Allows us to fit a polynomial to the data, instead of a straight line

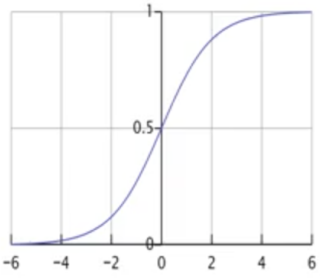


  - Can result in over-fitting a complex model
    - To help, we should combine with ridge regression
  - Similarly, other non-linear functions can be used to combine original features (called non-linear basis functions outside scope of this course)

- "**Feature Scaling**"
  - "MinMax" scaling:

$$x_i' = (x_i - x_i^{MIN}) / (x_i^{MAX} - x_i^{MIN})$$

  - both training data and test data should use identical scaling
- "**Logistic Regression**"
  - Used for classification: binary or multi-class
  - In linear regression, y is the sum of the weighted features. In logistic regression, instead of sum, we use an s-shaped logistic function



$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

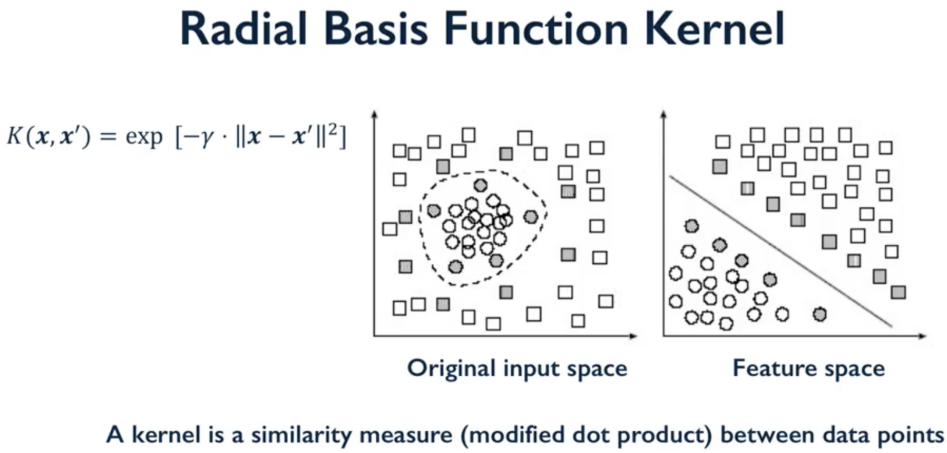
$$= \frac{1}{1 + \exp\left[-(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)\right]}$$

  - Like ridge and lasso linear regression, we can apply a regularization penalty to logistic regression
    - Called parameter C (default = 1.0) and is same as L2 (ridge) regularization
    - it can be important to normalize features when using regularization
- "**Support Vector Machines**"

- Linear classifier:
    - use linear regression as before, but then use a sign() function (or some other step function) to classify results based on some threshold
        - ex, if predicted value is positive, then class = 1, otherwise class = 0
    - "classifier margin" is the max width the decision boundary area can increase before hitting a data point
        - ie, how much separation we have between the data and the decision boundary
        - the best classifier has the max margin
            - this classifier is called the**Linear Support Vector Machine**(**LSVM**)
    - Regularization is controlled via parameter C
        - larger C means less regularization

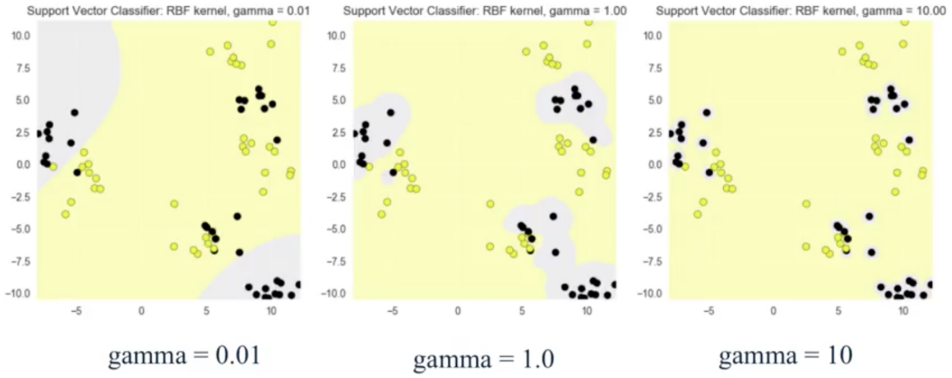
Linear Models	
Pros	**Cons**
Simple and easy to train	For lower-dimensional models, other models are better
Fast prediction	For classification, data may not be linearly separable
Scale well to very large datasets	
Work well with sparse data sets	
Relatively easy to interpret reasons for prediction	

- "**Multi Class Classification**"
    - Essentially breaks down the problem into a set of binary classifications, then it chooses the one that has the highest score
        - For example, if we have labels (a, b, c, d), it creates 4 binary classification models
            - a vs rest
            - b vs rest
            - c vs rest
            - d vs rest
        - when predicting, all 4 classifiers are run, and the one that gives the highest score is chosen as the predicted label
- "**Kernelized SVM**"
    - can be used for classification and regression
    - transforms original input data space into a higher dimensional data space that can be linearly classified
    - important to normalize input data
    - Radial Basis Function (RBF) Kernel



    - gamma controls how far the influence of a single training datum reaches, which affects how tightly the decision boundaries end up surrounding points in input space
    - KSVMs are very sensitive to value of gamma
        - small gamma means larger similar radius so that points further away are considered similar
            - this results in more points being grouped together and smoother decision boundaries
            - C (regularization) will have a big impact
        - bigger gamma means only points very close to each other are considered similar
            - results in more complex and tightly constrained decision boundaries
            - C (regularization) will have little to no effect

The effect of the RBF gamma parameter on decision boundaries



- Polynomial Kernel

Kernelized Support Vector Machines: pros and cons

Pros:

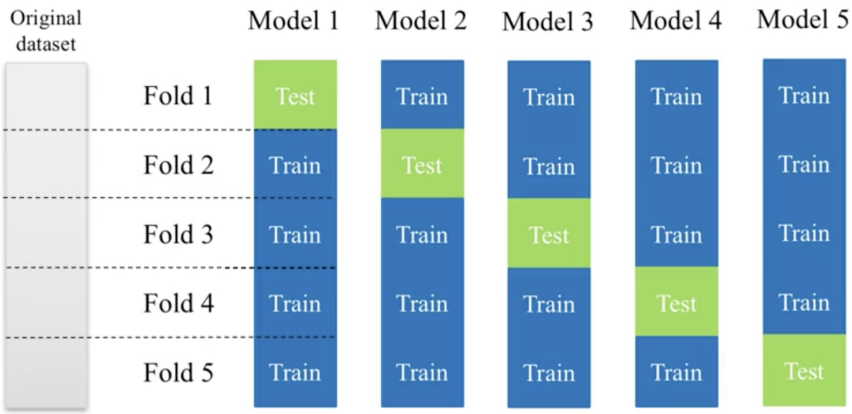
- Can perform well on a range of datasets.
- Versatile: different kernel functions can be specified, or custom kernels can be defined for specific data types.
- Works well for both low- and high-dimensional data.

Cons:

- Efficiency (runtime speed and memory usage) decreases as training set size increases (e.g. over 50000 samples).
- Needs careful normalization of input data and parameter tuning.
- Does not provide direct probability estimates (but can be estimated using e.g. Platt scaling).
- Difficult to interpret why a prediction was made.

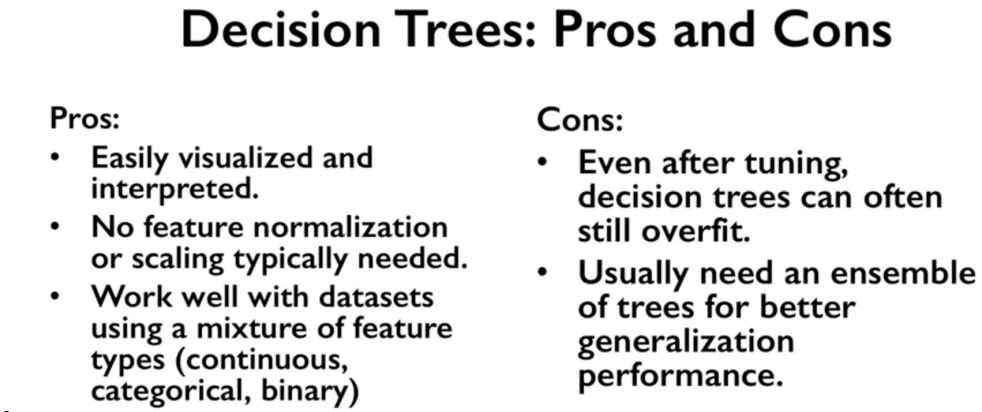
- "**Cross Validation**"
  - Used for evaluating a model, not tuning it
  - The random nature of the train/test split can give very different model parameters depending on which portion of the data set was used for training
  - The idea of cross validation is to generate multiple training/test data sets and validate the model using them and averaging the results
  - "K-fold cross validation"
    - K is usually 5 or 10
    - split the data into K equal parts

Cross-validation Example (5-fold)



  - For K=5, each model is trained with 4 folds, and tested with 1 fold. There will be 5 models.
  - Averaging out the scores of all models will give a good indication of what the model accuracy would be
  - If the scores of different models are very different, then this indicates that the model is very sensitive to the training data set
  - If the data set is sorted by label, then the cross validation can choose folds that have the same behavior and not representative of the overall data set

        - solution is called "**Stratified Cross Validation**" which chooses different data sets based on label
    - "**Leave-one-out Cross Validation**"
        - Each test fold contains only one item, and the rest of the data is part of the training set
        - computationally expensive, but gives best insight
- "**Decision Trees**"
    - Learn an explicit set of if-then rules on feature values that give the correct target values
    - Represented as a binary tree with yes/no answers to each rule
    - The goal is to find the fewest set of decisions that would give the most accurate answers
    - Each decision splits the data into two based on some threshold called the "split point" (below or above)
    - A good "informative" split is one that does a good job in separating the data
    - Multiple decision splits are made to create the tree
    - There can be multiple elements at the same leaf node
        - When using the model to predict, the tree is traversed according to the feature values of the element being predicted until we reach a leaf node
            - For the case of classification, the majority label of all the elements in the leaf node is chosen as the predicted label
            - For the case of regression, the mean value of the target values of all the elements in the leaf node is chosen as the predicted target value
    - Overfitting can be a real issue and we need to prevent that by forcing the tree to not become too complex (pruning)
        - max_depth
        - max_leaf_nodes max number of leaf nodes
        - min_samples_leaf min number of elements per leaf node
        - in practice, only need to adjust one of the above parameters to reduce overfitting (they impact each other)
    - The "feature importance" is good to look at to get some quick insight into the model
        - a feature with low importance doesn't mean it's not useful. It means that it wasn't used early on to split the data
        - it does not give insight into how features are related
        - a feature with high importance means that it was used early on to split data
        - when performing cross validation, it's important to average out the feature importance for each of the models



Week 3

- "**Accuracy**" is not always the best measure for evaluating a model
    - you have to choose the correct evaluation metric for your problem and your goals
    - For example, for classifying cancerous tumors, we might want to have the classifier err on the side of caution, and classify non-cancerous tumors as cancerous, in order to increase survival rate
- **"Imbalanced classes"**
    - in the case of binary classification, one of the two classes might have many more hits that the other in the training set
    - for example, classifying credit card transactions. Most transactions are good, and the bad ones are very few in comparison
    - another example is a product recommendation system. Most product are not relevant, while only a small number of them are relevant to previous purchases
    - this happens a lot
    - in this case, "accuracy" is a bad metric to evaluate a classifier because if the classifier simply always chooses the dominant class, then it will have a very high accuracy
- **Dummy regressor/classifier**is one that always outputs a constant value that can be controlled
    - most_frequent, stratified, uniform,mean, median, user_controlled, constant
    - can use a dummy regressor to compare its performance to your best model, and see if your features are effective or erroneous or whether you are using the wrong kernel or model parameters. it could also mean that you have a large class imbalance

- **Confusion Matrix**

Confusion Matrix for Binary Prediction Task

	Predicted negative	Predicted positive
True negative	TN = 356	FP = 51
True positive	FN = 38	TP = 5

N = 450

- Every test instance is in exactly one box (integer counts).
- Breaks down classifier results by error type.
- Thus, provides more information than simple accuracy.
- Helps you choose an evaluation metric that matches project goals.
- Not a single number like accuracy.. but there are many possible metrics that can be derived from the confusion matrix.

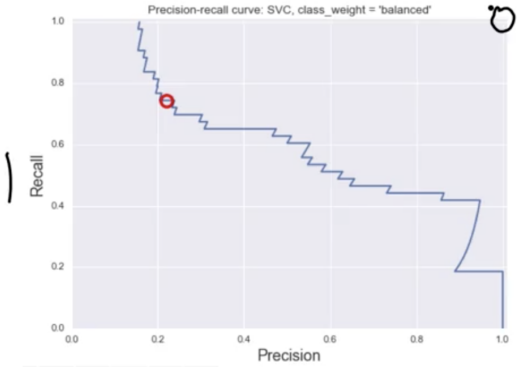
- Other evaluation metrics based on confusion matrix:
  - **Accuracy**= (TN + TP) / (TN + TP + FN + FP)
  - **Classification error**= 1 - Accuracy
  - **Recall**or**True Positive Rate**(TPR) = TP / (TP + FN)
    - what fraction of all positive instances are classified correctly?
    - also known as 'Sensitivity' and 'Probability of Detection'
  - **Precision**: TP / (TP + FP)
    - avoid false positives. what fraction of positive predictions are correct?
    - to improve precision, we have to either improve the number of correct positive predictions (TP) or reduce the number of incorrect positive predictions (FP)
  - **False Positive Rate**(FPR): FP / (TN + FP)
    - what fraction of negatives does the classifier incorrectly identify as positives?
    - also known as specificity
  - You have to choose the correct metric based on your goal
- **Decision Functions**
  - decision_function
    - tells us how confidently a classifier will predict the positive class or the negative class
    - for positive instances, this should be a large positive value, while for negative instances this should be a large negative value
  - predict_proba
    - predicted probability of class membership
- **Precision-Recall Curves**
  - Very widely used evaluation method
  - Scikit has a built-in way to generate those curves

Precision-Recall Curves

X-axis: Precision
Y-axis: Recall

Top right corner:
- The "ideal" point
- Precision = 1.0
- Recall = 1.0

"Steepness" of P-R curves is important:
- Maximize precision
- while maximizing recall



- **ROC Curves**

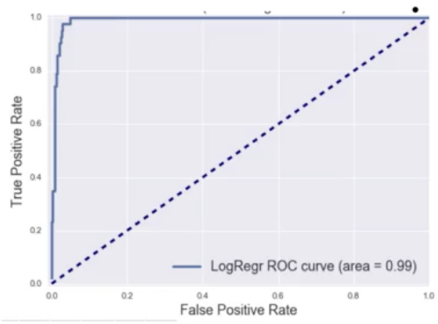
- Used to evaluate binary classifiers

ROC Curves

X-axis: False Positive Rate
Y-axis: True Positive Rate

Top left corner:
- The "ideal" point
- False positive rate of zero
- True positive rate of one

"Steepness" of ROC curves is important:
- Maximize the true positive rate
- while minimizing the false positive rate



- Alternatively, we want to maximize the area under the curve (AUC)

- **Multi-class evaluation** is a straight-forward extension of 2-class (binary) evaluation
  - Evaluation metrics are averages across classes
  - Important to take into account the number of instances in each class (for imbalanced classes)
  - "Multi-label classification" is when one instance can have multiple labels
    - not covered in this course
  - The multi-class confusion matrix is a k-by-k matrix, where k is the number of classes
    - correct classifications are along the diagonal
    - wrong classifications are off the diagonal
  - Macro average precision
    - each *class* has equal weight
    - compute metric within each class
    - average resulting metrics across classes
  - Micro average precision
    - Each *instance* has equal weight
    - If there is a dominant class in the data, then this class will be given much more influence than other classes due to the number of instances
  - If the classes have about the same number of instances, then macro- and micro-averages will be around the same
  - If some classes have much more instances than others, then
    - micro-averaging will be weighted toward the largest classes
    - macro-averaging will be weighted toward the smallest classes
  - If micro-average << macro-average, then examine larger classes for poor metric performance
  - If macro-average << micro-average, then examine smaller classes for poor metric performance

```
print('Micro-averaged precision = {:.2f} (treat instances equally)'
      .format(precision_score(y_test_mc, svm_predicted_mc, average = 'micro')))
print('Macro-averaged precision = {:.2f} (treat classes equally)'
      .format(precision_score(y_test_mc, svm_predicted_mc, average = 'macro')))
```

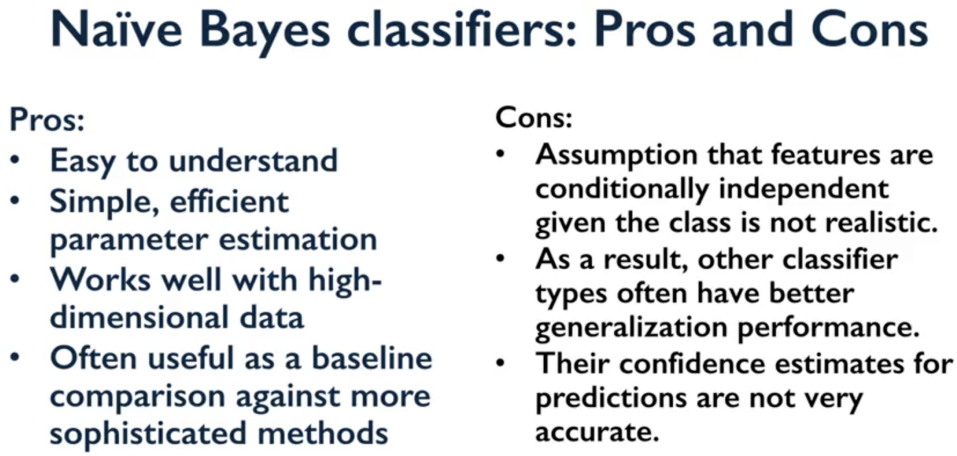
```
Micro-averaged precision = 0.49 (treat instances equally)
Macro-averaged precision = 0.91 (treat classes equally)
```

- "**Regression Evaluation**"
  - In most cases, the r2_score number is enough
    - between 0 and 1, but can be negative
  - Other metrics:
    - mean_absolute_error
    - mean_squared_error
    - median_absolute_error (robust to outliers)
- "**Model Selection**"

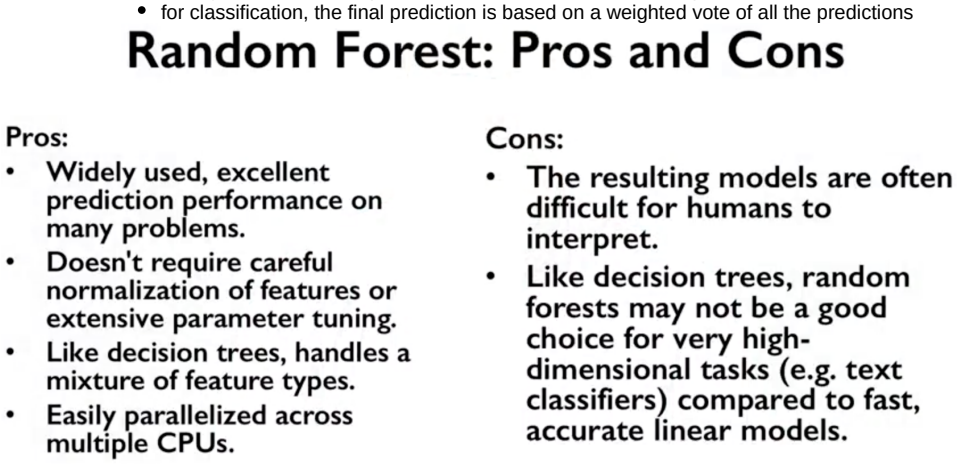
Concluding Notes

- Accuracy is often not the right evaluation metric for many real-world machine learning tasks
  - False positives and false negatives may need to be treated very differently
  - Make sure you understand the needs of your application and choose an evaluation metric that matches your application, user, or business goals.
- Examples of additional evaluation methods include:
  - Learning curve: How much does accuracy (or other metric) change as a function of the amount of training data?
  - Sensitivity analysis: How much does accuracy (or other metric) change as a function of key learning parameter values?

- "**Naive Bayes Classifiers**"
  - called "naive" because they make the assumption that features are independent of each other
  - types include:
    - bernoulli
      - binary features (feature exists or not in this instance)
    - multinomial
      - discrete features (feature is a count of occurrences of something)
    - gaussian
      - continuous values
      - during training: estimates the mean and std-dev of the feature for each class
      - during prediction: finds the class that best matches the feature mean/std-dev of the given instance features
  - Bernoulli and multinomial are usually used for text classification
    - work well with sparse data sets
  - Gaussian
    - Good for use when you have 100s and 1000s of features (higher dimensional data)



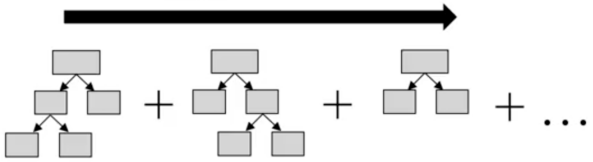
- "**Random Forests**"
  - An "ensemble" is a method that uses multiple learning models and aggregating them into one model that is better than any of the individual models since it will be immune to weaknesses of any one model.
  - RFs are an ensemble of trees
    - contains 100s or more of trees
    - introduces random variations into tree-building that improves stability
      - two types of randomness:
        1. the data used to build each tree is selected randomly
           - bootstrap samples use random selection with replacement
             - if we have N data samples, we select N samples from them, allowing us to choose the same data row again.
        2. features used in each split randomly are randomly selected
           - in decision trees, the best feature is used for each split from all the features available
           - in each RF tree, a random subset of features are chosen for each split, then the best one among those features in chosen for that split
           - the number of features in this subset is controllable (max_features parameter) and RFs are highly sensitive to it
    - When predicting, the RF predicts the output of every tree
      - for regression, the final prediction is the mean of all the tree predictions
      - for classification, the final prediction is based on a weighted vote of all the predictions



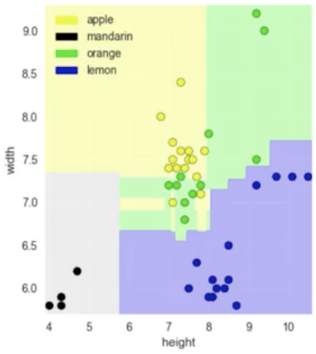
- "**Gradient Boosted Decision Trees**"
  - An ensemble of decision trees
  - Creates a series of trees where each tree is trained to attempt to correct mistakes in the previous tree
    - in contrast, random forests create a set of trees in parallel

Gradient Boosted Decision Trees (GBDT)

- Training builds a series of small decision trees.
- Each tree attempts to correct errors from the previous stage.



- The learning rate controls how hard each new tree tries to correct remaining mistakes from previous round.
  - High learning rate: more complex trees
  - Low learning rate: simpler trees



GBDT decision regions on two-feature fruits dataset

GBDT: Pros and Cons

Pros:

- **Often best off-the-shelf accuracy on many problems.**
- **Using model for prediction requires only modest memory and is fast.**
- **Doesn't require careful normalization of features to perform well.**
- **Like decision trees, handles a mixture of feature types.**

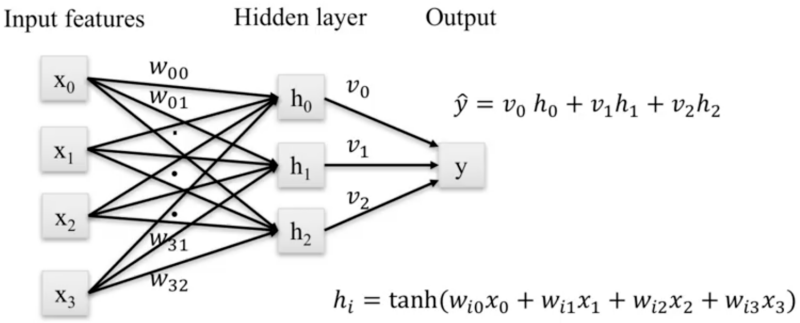
Cons:

- **Like random forests, the models are often difficult for humans to interpret.**
- **Requires careful tuning of the learning rate and other parameters.**
- **Training can require significant computation.**
- **Like decision trees, not recommended for text classification and other problems with very high dimensional sparse features, for accuracy and computational cost reasons.**

- **"Neural Networks"**

Multi-layer Perceptron with One Hidden Layer

(and tanh activation function)



- Neural networks need more training data and more computation to train as compared to linear models
- Activation functions:
  - tanh
  - relu (Default in scikit)
    - rectified linear unit function
    - y = 0 for x < 0
    - y = x for x >= 0
  - logistic function
- normalization of input features can be critical

Neural Networks: Pros and Cons

Pros:

- They form the basis of state-of-the-art models and can be formed into advanced architectures that effectively capture complex features given enough data and computation.

Cons:

- Larger, more complex models require significant training time, data, and customization.
- Careful preprocessing of the data is needed.
- A good choice when the features are of similar types, but less so when features of very different types.

- "**Deep Learning**"

Deep Learning Summary

- Deep learning architectures combine a sophisticated automatic feature extraction phase with a supervised learning phase.
- The feature extraction phase uses a hierarchy of multiple feature extraction layers.
- Starting from primitive, low-level features in the initial layer, each feature layer's output provides the input features to the next higher feature layer.
- All features are used in the final supervised learning model.

Pros and Cons of Deep Learning

- Pros:
  - *Powerful: deep learning has achieved significant gains over other machine learning approaches on many difficult learning tasks, leading to state-of-the-art performance across many different domains.*
  - *Does effective automatic feature extraction, reducing the need for guesswork and heuristics on this key problem.*
  - *Current software provides flexible architectures that can be adapted for new domains fairly easily.*
- Cons:
  - *Can require huge amounts of training data.*
  - *Can require huge amounts of computing power.*
  - *Architectures can be complex and often must be highly tailored to a specific application.*
  - *The resulting models may not be easily interpretable.*

- "**Data Leakage**"
  - When data about the target variable is included in the training data
    - this data would not be normally available during actual model usage
  - For example
    - including target data in input features
    - including test data in training data
  - If your model's performance is too good to be true, then it's probably due to data leakage

Detecting Data Leakage

- **Before building the model**
  - *Exploratory data analysis to find surprises in the data*
  - *Are there features very highly correlated with the target value?*
- **After building the model**
  - *Look for surprising feature behavior in the fitted model.*
  - *Are there features with very high weights, or high information gain?*
  - *Simple rule-based models like decision trees can help with features like account numbers, patient IDs*
  - *Is overall model performance surprisingly good compared to known results on the same dataset, or for similar problems on similar datasets?*
- **Limited real-world deployment of the trained model**
  - *Potentially expensive in terms of development time, but more realistic*
  - *Is the trained model generalizing well to new data?*

- "**Unsupervised Machine Learning**"

Introduction: Unsupervised Learning

- **Unsupervised learning involves tasks that operate on datasets without labeled responses or target values.**
- **Instead, the goal is to capture interesting structure or information.**

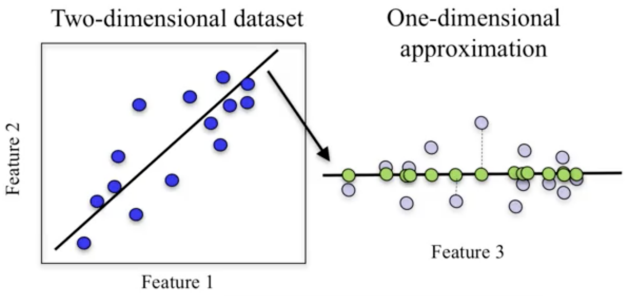
Applications of unsupervised learning:

- **Visualize structure of a complex dataset.**
- **Density estimation to predict probabilities of events.**
- **Compress and summarize the data.**
- **Extract features for supervised learning.**
- **Discover important clusters or outliers.**

- Two types of UL:
  - Transformations
    - Density estimation (ex: Kernel density estimation to create heat maps from geospatial data)
    - Dimensionality reduction (ex: PCA)

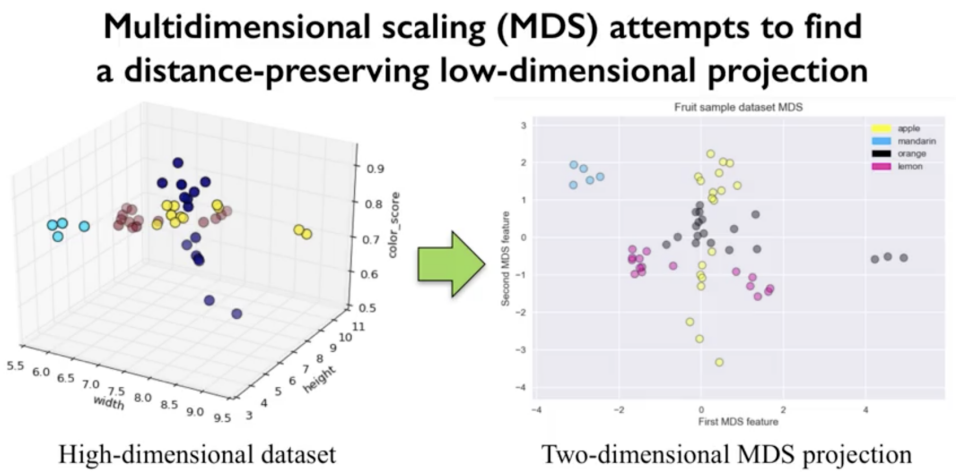
Dimensionality Reduction

- **Finds an approximate version of your dataset using fewer features**
- **Used for exploring and visualizing a dataset to understand grouping or relationships**
- **Often visualized using a 2-dimensional scatterplot**
- **Also used for compression, finding features for supervised learning**



The one-dimensional approximation is obtained by projecting the original points onto the diagonal line and using their position on that line as the new single feature.

    - Manifold Learning
      - More sophisticated than PCA



- t-SNE
  - finds a 2d representation of your data such that the distances between data points in the 2d space match as closely as possible to the distances in the original higher-dimensional space
  - gives much more weight to preserving distances between neighbors

- Clustering
  - Divide data into groups or clusters based on similarity
  - Some clustering models can predict clusters of new data points. This is similar to classification but without training using labeled examples.
  - Can be hard to interpret meaning of the clustering. Humans expertise is often needed.
  - K-means Clustering
    - You have to tell it how many clusters (K) to create, it then classifies the data into that many clusters
    - It then randomly picks k locations to serve as initial guess for cluster centers
    - Then it assigns each data point to the nearest cluster center
    - Each cluster center is then replaced with the mean of all the points assigned to it
    - The two previous steps repeated until convergence
    - Different init locations can result in different solutions
      - SciKit solves this by running 10 different K-means runs and picks the best
    - Important to normalize features using MinMax scaling
    - Works well when clusters are roughly same size and doesn't work well when the sizes are irregular
  - Agglomerative Clustering
    1. Each point is put in its own cluster
    2. The two most similar clusters are merged
       a. There are different ways of merging: Ward's method, average linkage and complete linkage
    3. Continue until the desired number of clusters is reached
  - DBSCAN Clustering
    - Density Based Spatial Clustering of Applications with Noise
    - don't need to specify the number of clusters before hand
    - works well with clusters of different shapes
    - can identify outliers that shouldn't be assigned to any cluster
    - efficient: can be used on large data sets
    - main idea: clusters represent areas that are more dense, separated by less dense areas