Text Data in Business and Economics

Basel University – Autumn 2024

5. Supervised and Unsupervised Learning from Text

Different Goals, Different Methods

- Supervised Learning
  - pursuing a known goal, e.g., predicting whether a political speech is from a Democrat or a Republican.
  - machine learns to replicate labels for new data points

Different Goals, Different Methods

- Supervised Learning
  - pursuing a known goal, e.g., predicting whether a political speech is from a Democrat or a Republican.
  - machine learns to replicate labels for new data points
- Unsupervised Learning
  - algorithm discovers themes/patterns in text (or other high-dimensional data)
  - human interprets the results (e.g. inspect content of topics or clusters)

Different Goals, Different Methods

- Supervised Learning
  - pursuing a known goal, e.g., predicting whether a political speech is from a Democrat or a Republican.
  - machine learns to replicate labels for new data points
- Unsupervised Learning
  - algorithm discovers themes/patterns in text (or other high-dimensional data)
  - human interprets the results (e.g. inspect content of topics or clusters)
- Both strategies amplify human effort, each in different ways.

Different Goals, Different Methods

- Supervised Learning
  - pursuing a known goal, e.g., predicting whether a political speech is from a Democrat or a Republican.
  - machine learns to replicate labels for new data points
- Unsupervised Learning
  - algorithm discovers themes/patterns in text (or other high-dimensional data)
  - human interprets the results (e.g. inspect content of topics or clusters)
- Both strategies amplify human effort, each in different ways.
- Distinctions are not clear-cut:
  - supervised learning models can be used to discover themes/patterns
  - unsupervised learning models can be used in service of prediction or known goals.

Objectives: Social-Science Research using Unsupervised Learning

1. **What is the research question?**

Objectives: Social-Science Research using Unsupervised Learning

1. **What is the research question?**
2. Corpus and Data:
   - obtain, clean, preprocess, and link.
   - Produce descriptive visuals and statistics on the text and metadata

Objectives: Social-Science Research using Unsupervised Learning

1. **What is the research question?**
2. Corpus and Data:
   - obtain, clean, preprocess, and link.
   - Produce descriptive visuals and statistics on the text and metadata
3. Unsupervised learning:
   - **What are we trying to measure?**

Objectives: Social-Science Research using Unsupervised Learning

1. **What is the research question?**
2. Corpus and Data:
   - obtain, clean, preprocess, and link.
   - Produce descriptive visuals and statistics on the text and metadata
3. Unsupervised learning:
   - **What are we trying to measure?**
   - Select a model and train it.
   - Probe sensitivity to hyperparameters.
   - Validate that the model is measuring what we want.

Objectives: Social-Science Research using Unsupervised Learning

1. **What is the research question?**
2. Corpus and Data:
   - obtain, clean, preprocess, and link.
   - Produce descriptive visuals and statistics on the text and metadata
3. Unsupervised learning:
   - **What are we trying to measure?**
   - Select a model and train it.
   - Probe sensitivity to hyperparameters.
   - Validate that the model is measuring what we want.
4. Empirical analysis
   - Produce statistics or predictions with the trained model.
   - **Answer the research question.**

Outline

Dimensionality Reduction

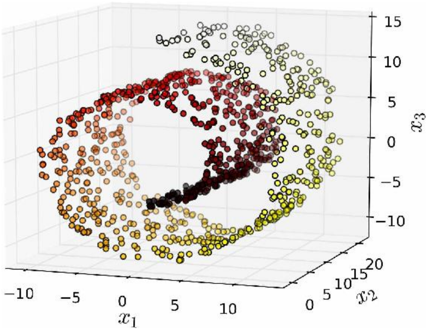
Topic Models

Supervised Learning
- Overview
- Regression / Regularization
- Binary Classification
- Multi-Class Models

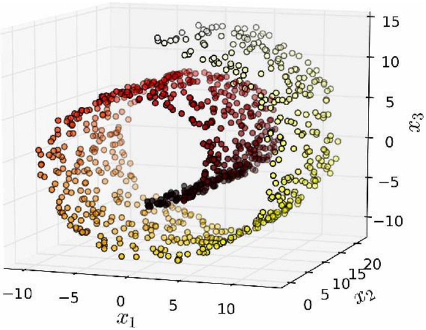
Ensemble Learning with XGBoost

- Datasets are not distributed uniformly across the feature space.
- They have a lower-dimensional latent structure – a **manifold** – that can be learned.

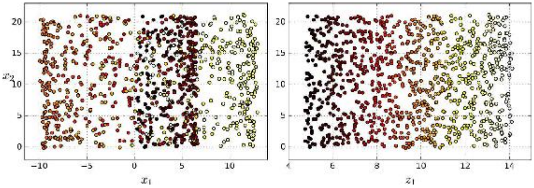
“The Swiss Roll”



“The Swiss Roll”



- Datasets are not distributed uniformly across the feature space.
- They have a lower-dimensional latent structure – a **manifold** – that can be learned.



- **Dimensonality reduction** makes data more interpretable – for example by projecting down to two dimensions for visualization.
- improves computational tractability.
- can improve model performance.

What dimension reductions have you already tried in this class?

The Document-Term Matrix is high-dimensional

The **document-term matrix $\boldsymbol{X}$**:

- each row d represents a **document,** while each column w represents a word (or term more generally, e.g. n-grams).
  - A matrix entry $\boldsymbol{X}_{[d,w]}$ quantifies the strength of association between a document and a word, generally its count or frequency

The Document-Term Matrix is high-dimensional

The **document-term matrix $\boldsymbol{X}$**:

- each row d represents a **document,** while each column w represents a word (or term more generally, e.g. n-grams).
  - A matrix entry $\boldsymbol{X}_{[d,w]}$ quantifies the strength of association between a document and a word, generally its count or frequency
- each document/row $\boldsymbol{X}_{[d,:]}$ is a distribution over terms
  - term vocabularies can be in the hundreds of thousands

The Document-Term Matrix is high-dimensional

The **document-term matrix $\boldsymbol{X}$**:

- each row d represents a **document,** while each column w represents a word (or term more generally, e.g. n-grams).
  - A matrix entry $\boldsymbol{X}_{[d,w]}$ quantifies the strength of association between a document and a word, generally its count or frequency
- each document/row $\boldsymbol{X}_{[d,:]}$ is a distribution over terms
  - term vocabularies can be in the hundreds of thousands
- each word/column $\boldsymbol{X}_{[:,w]}$ is a distribution over documents.
  - many interesting corpora have millions of documents

The Document-Term Matrix is high-dimensional

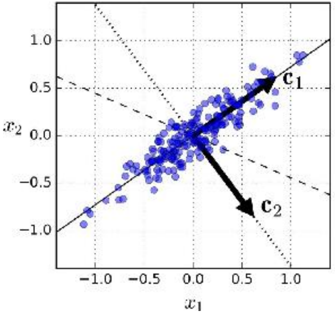
The **document-term matrix** $\boldsymbol{X}$:

- each row d represents a **document,** while each column w represents a word (or term more generally, e.g. n-grams).
  - A matrix entry $\boldsymbol{X}_{[d,w]}$ quantifies the strength of association between a document and a word, generally its count or frequency
- each document/row $\boldsymbol{X}_{[d,:]}$ is a distribution over terms
  - term vocabularies can be in the hundreds of thousands
- each word/column $\boldsymbol{X}_{[:,w]}$ is a distribution over documents.
  - many interesting corpora have millions of documents

$\rightarrow$ $\boldsymbol{X}$ often has billions of cells.

PCA (principal component analysis)

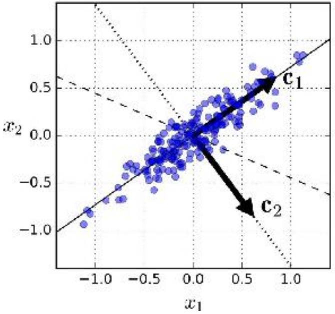
PCA (principal component analysis)



- PCA computes the dimension in data explaining most variance.

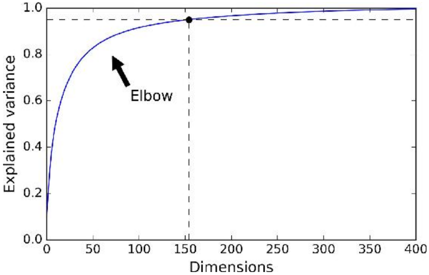
```
from sklearn.decomposition import PCA
pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train)
```

PCA (principal component analysis)



- PCA computes the dimension in data explaining most variance.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train)
```



- after the first component, subsequent components learn the (orthogonal) dimensions explaining most variance in dataset after projecting out first component.

PCA and LSA

The document-term matrix $\boldsymbol{X}$ can be reduced by projecting down to first principal component dimensions.

- This is known as "latent semantic analysis"
- Distance metrics between observations (e.g. cosine similarity) are approximately preserved.

PCA and LSA

The document-term matrix $\boldsymbol{X}$ can be reduced by projecting down to first principal component dimensions.

- This is known as "latent semantic analysis"
- Distance metrics between observations (e.g. cosine similarity) are approximately preserved.

- PCA factors are not interpretable.
  - Hoffman (1999) fixes this and puts LSA on firmer foundations by assuming a generative model of text – the word counts in a document are generated by a multinomial distribution.
  - For non-negative data (e.g. counts or frequencies), **Non-negative Matrix Factorization (NMF)** provides more interpretable factors than PCA.

Outline

Topic Models in Social Science

- Core methods for topic models were developed in computer science and statistics
  - summarize unstructured text
  - use words within document to infer subject
  - useful for dimension reduction

Topic Models in Social Science

- Core methods for topic models were developed in computer science and statistics
  - summarize unstructured text
  - use words within document to infer subject
  - useful for dimension reduction
- Social scientists use topics as a form of measurement
  - how observed covariates drive trends in language
  - tell a story not just about what, but how and why

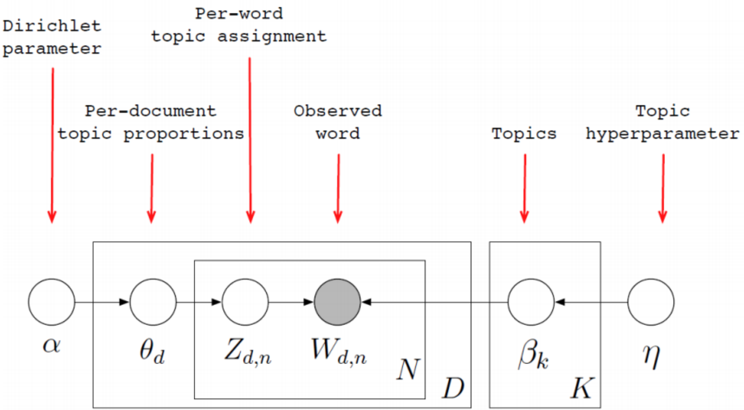
Topic Models in Social Science

- Core methods for topic models were developed in computer science and statistics
  - summarize unstructured text
  - use words within document to infer subject
  - useful for dimension reduction
- Social scientists use topics as a form of measurement
  - how observed covariates drive trends in language
  - tell a story not just about what, but how and why
  - **topic models are more interpretable** than other dimension reduction methods, such as PCA.

- Latent Dirichlet Allocation (LDA):
  - Each topic is a distribution over words.
  - Each document is a distribution over topics.

- Latent Dirichlet Allocation (LDA):
  - Each topic is a distribution over words.
  - Each document is a distribution over topics.
- Input: $N \times M$ document-term count matrix X
- Assume: there are K topics (tunable hyperparameter, use coherence).
- Like PCA or NMF, LDA works by factorizing X into:
  - an $N \times K$ document-topic matrix
  - an $K \times M$ topic-term matrix.

- Latent Dirichlet Allocation (LDA):
  - Each topic is a distribution over words.
  - Each document is a distribution over topics.
- Input: $N \times M$ document-term count matrix X
- Assume: there are K topics (tunable hyperparameter, use coherence).
- Like PCA or NMF, LDA works by factorizing X into:
  - an $N \times K$ document-topic matrix
  - an $K \times M$ topic-term matrix.



Using an LDA Model

Once trained, can easily get topic proportions for a corpus.

- for any document – doesn't have to be in training corpus.
- main topic is the highest-probability topic

Using an LDA Model

Once trained, can easily get topic proportions for a corpus.

- for any document – doesn't have to be in training corpus.
- main topic is the highest-probability topic
- documents with highest share in a topic work as representative documents for the topic.

Using an LDA Model

Once trained, can easily get topic proportions for a corpus.

- for any document – doesn't have to be in training corpus.
- main topic is the highest-probability topic
- documents with highest share in a topic work as representative documents for the topic.

Can then use the topic proportions as variables in a social science analysis.

- e.g., Catalinac (2016) shows that after a Japanese political reform that reduced intraparty competition, candidate platforms reduced pork-barrel policies and increased national ones.

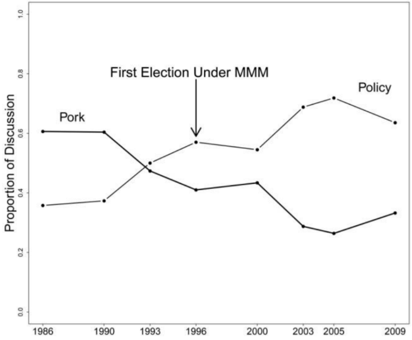


Table 1 A Summary of Common Assumptions and Relative Costs Across Different Methods of Discrete Text Categorization

	Method				
A. Assumptions	***Reading***	***Human Coding***	***Dictionaries***	***Supervised Learning***	***Topic Model***
Categories are known	No	Yes	Yes	Yes	No
Category nesting, if any, is known	No	Yes	Yes	Yes	No
Relevant text features are known	No	No	Yes	Yes	Yes
Mapping is known	No	No	Yes	No	No
Coding can be automated	No	No	Yes	Yes	Yes
B. Costs					
Preanalysis Costs					
Person-hours spent conceptualizing	Low	High	High	High	Low
Level of substantive knowledge	Moderate/High	High	High	High	Low
Analysis Costs					
Person hours spent per text	High	High	Low	Low	Low
Level of substantive knowledge	Moderate/High	Moderate	Low	Low	Low
Postanalysis Costs					
Person-hours spent interpreting	High	Low	Low	Low	Moderate
Level of substantive knowledge	High	High	High	High	High

Recommended: read this part of Quinn, Monroe, Colaresi, Crespin, and Radev (2010).

Structural Topic Model = LDA + Metadata

Roberts, Stewart, and Tingley

STM provides two ways to include contextual information:

- Topic prevalence can vary by metadata
  - e.g. Republicans talk about military issues more then Democrats

Structural Topic Model = LDA + Metadata

Roberts, Stewart, and Tingley

STM provides two ways to include contextual information:

- Topic prevalence can vary by metadata
  - e.g. Republicans talk about military issues more then Democrats
- Topic content can vary by metadata
  - e.g. Republicans talk about military issues more patriotically than Democrats.

Structural Topic Model = LDA + Metadata

Roberts, Stewart, and Tingley

STM provides two ways to include contextual information:

- Topic prevalence can vary by metadata
  - e.g. Republicans talk about military issues more then Democrats
- Topic content can vary by metadata
  - e.g. Republicans talk about military issues more patriotically than Democrats.

- Structural topic model is not a prediction model:
  - it will tell you which topics or features correlate with an outcome, but it will not provide an in-sample or out-of-sample prediction for an outcome

Structural Topic Model = LDA + Metadata

Roberts, Stewart, and Tingley

STM provides two ways to include contextual information:

- Topic prevalence can vary by metadata
  - e.g. Republicans talk about military issues more then Democrats
- Topic content can vary by metadata
  - e.g. Republicans talk about military issues more patriotically than Democrats.

- Structural topic model is not a prediction model:
  - it will tell you which topics or features correlate with an outcome, but it will not provide an in-sample or out-of-sample prediction for an outcome
- It actually uses another distribution of the priors (*not* Dirichlet) such that without covariates it replicates the correlated topic model (Blei and Lafferty, 2005)

Recent Advances in Topic Model

- Keyword-Assisted topic model (Eshima, Imai, and Sasaki, 2024)
  - allows *semi-supervised* creation of topics
  - input seed dictionaries and labeled topics

Recent Advances in Topic Model

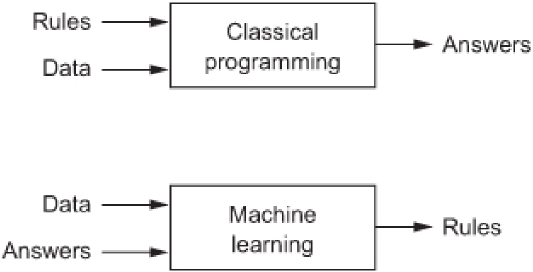
- Keyword-Assisted topic model (Eshima, Imai, and Sasaki, 2024)
  - allows *semi-supervised* creation of topics
  - input seed dictionaries and labeled topics
- Generalized topic model (Gauthier, Widmer, and Ash, 2024)
  - includes covariates and auxiliary outcomes, allowing for these to affect the priors
  - allows for the use of both text and images

Recent Advances in Topic Model

- Keyword-Assisted topic model (Eshima, Imai, and Sasaki, 2024)
  - allows *semi-supervised* creation of topics
  - input seed dictionaries and labeled topics
- Generalized topic model (Gauthier, Widmer, and Ash, 2024)
  - includes covariates and auxiliary outcomes, allowing for these to affect the priors
  - allows for the use of both text and images
- Problems with unstructured data and casual inference (Battaglia et al., 2024)
  - shows that two-steps strategy leads to invalid inference
  - propose solutions with bias correction and a one-step strategy

Outline

What is machine learning?



- In classical computer programming, humans input the rules and the data, and the computer provides answers.
- In machine learning, humans input the data and the answers, and the computer learns the rules.

What do ML Algorithms do? Fit a function to data points

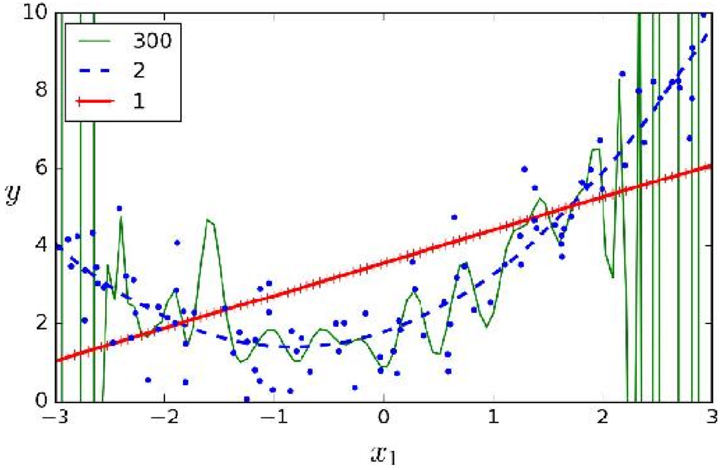


Figure 4-14. High-degree Polynomial Regression

What do ML Algorithms do? Minimize a cost function

- A typical cost function (or loss function) for regression problems is Mean Squared Error (MSE):

$$\text{MSE}(\theta) = \frac{1}{n_D} \sum_{i=1}^{n_D} (h(x_i; \theta) - y_i)^2$$

  - n_D, the number of rows/observations
  - x, the matrix of predictors, with row x_i
  - y, the vector of outcomes, with item y_i
  - $h(x_i; \theta) = \hat{y}$ the model prediction (hypothesis)

The **data** (x, y) are taken as given, and the ML algorithm searches for **parameters** θ to minimize the cost function.

Linear Regression is Machine Learning

- Ordinary Least Squares Regression (OLS) assumes the functional form $f(x;\theta) = x_i'\theta$ and minimizes the mean squared error (MSE)

$$\min_{\hat{\theta}} \frac{1}{n_D} \sum_{i=1}^{n_D} (x_i'\hat{\theta} - y_i)^2$$

Linear Regression is Machine Learning

- Ordinary Least Squares Regression (OLS) assumes the functional form $f(x;\theta) = x_i'\theta$ and minimizes the mean squared error (MSE)

$$\min_{\hat{\theta}} \frac{1}{n_D} \sum_{i=1}^{n_D} (x_i'\hat{\theta} - y_i)^2$$

- This minimand has a closed form solution

$$\hat{\theta} = (\boldsymbol{x}'\boldsymbol{x})^{-1}\boldsymbol{x}'\boldsymbol{y}$$

  - most machine learning models do **not** have a closed form solution $\rightarrow$ use numerical optimization instead (gradient descent).

$$\mathrm{MSE}(\theta) = \frac{1}{n_D}\sum_{i=1}^{n_D}(h(\theta; \boldsymbol{x}_i) - y_i)^2$$

- The partial derivative for feature j is

$$\frac{\partial \mathrm{MSE}}{\partial \theta_j} = \frac{2}{n_D}\sum_{i=1}^{n_D}(\underbrace{h(\theta; \boldsymbol{x}_i) - y_i}_{\text{error for this obs}}) \underbrace{\frac{\partial h(\theta; \boldsymbol{x}_i)}{\partial \theta_j}}_{\text{how } \theta_j \text{ shifts } h(\cdot)}$$

  - $\rightarrow$ estimates how changing θ_j would reduce the error across the whole dataset.

$$\text{MSE}(\theta) = \frac{1}{n_D}\sum_{i=1}^{n_D}(h(\theta; \boldsymbol{x}_i) - y_i)^2$$

- The partial derivative for feature j is

$$\frac{\partial \text{MSE}}{\partial \theta_j} = \frac{2}{n_D}\sum_{i=1}^{n_D}\underbrace{(h(\theta; \boldsymbol{x}_i) - y_i)}_{\text{error for this obs}} \underbrace{\frac{\partial h(\theta; \boldsymbol{x}_i)}{\partial \theta_j}}_{\text{how } \theta_j \text{ shifts } h(\cdot)}$$

  - $\rightarrow$ estimates how changing θ_j would reduce the error across the whole dataset.

- The ***gradient*** ∇ gives the vector of these partial derivatives for all features:

$$\nabla_\theta \text{MSE} = \begin{bmatrix} \frac{\partial \text{MSE}}{\partial \theta_1} \\ \frac{\partial \text{MSE}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \text{MSE}}{\partial \theta_{n_x}} \end{bmatrix}$$

- ***Gradient descent*** nudges θ against the gradient (the direction that reduces MSE):

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \text{MSE}$$

- η = learning rate

$$\text{MSE}(\theta) = \frac{1}{n_D}\sum_{i=1}^{n_D}(h(\theta; \boldsymbol{x}_i) - y_i)^2$$

- The partial derivative for feature j is

$$\frac{\partial \text{MSE}}{\partial \theta_j} = \frac{2}{n_D}\sum_{i=1}^{n_D}\underbrace{(h(\theta; \boldsymbol{x}_i) - y_i)}_{\text{error for this obs}}\underbrace{\frac{\partial h(\theta; \boldsymbol{x}_i)}{\partial \theta_j}}_{\text{how } \theta_j \text{ shifts } h(\cdot)}$$

  - $\rightarrow$ estimates how changing θ_j would reduce the error across the whole dataset.

- The ***gradient*** ∇ gives the vector of these partial derivatives for all features:

$$\nabla_\theta \text{MSE} = \begin{bmatrix} \frac{\partial \text{MSE}}{\partial \theta_1} \\ \frac{\partial \text{MSE}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \text{MSE}}{\partial \theta_{n_x}} \end{bmatrix}$$

- ***Gradient descent*** nudges θ against the gradient (the direction that reduces MSE):

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \text{MSE}$$

- η = learning rate

- If the cost function is convex, gradient descent is guaranteed to find the global minimum.

Machine Learning with Text Data

- We have a corpus (or dataset) D of $n_D \geq 1$ documents d_i (or data points).

Machine Learning with Text Data

- We have a corpus (or dataset) D of $n_D \geq 1$ documents d_i (or data points).
- Each document i has an associated outcome or label $\boldsymbol{y}_i$ with dimensions $n_y \geq 1$

Machine Learning with Text Data

- We have a corpus (or dataset) D of $n_D \geq 1$ documents d_i (or data points).
- Each document i has an associated outcome or label $\boldsymbol{y}_i$ with dimensions $n_y \geq 1$
- Some documents are labeled and some are unlabeled $\rightarrow$
  - we would like to learn a function $\hat{\boldsymbol{y}}(d_i)$ based on the labeled data ...
  - ... to machine-classify the unlabeled data.

First Problem

- Each document is a sequence of symbols d_i, while (standard) ML algorithms work on numbers.

First Problem

- Each document is a sequence of symbols d_i, while (standard) ML algorithms work on numbers.
- The solution: all the methods from previous lectures for extracting informative numerical information from documents:
  - style features
  - counts over dictionary patterns
  - tokens
  - n-grams
  - principal components
  - topic shares
  - etc.
- documents can thus be **featurized** – represented as a matrix of vectors $\boldsymbol{x}$ with $n_x \geq 1$ features.

Three Types of (Standard) Machine Learning Problems

Determined by the data type of the outcome variable (or label):

- **Binary classification**: two choices, normalized to zero and one.
  - e.g., guilty or innocent

Three Types of (Standard) Machine Learning Problems

Determined by the data type of the outcome variable (or label):

- **Binary classification**: two choices, normalized to zero and one.
  - e.g., guilty or innocent
- **Regression**: a one-dimensional, continuous, real-valued outcome.
  - e.g., number of days of prison assigned

Three Types of (Standard) Machine Learning Problems

Determined by the data type of the outcome variable (or label):

- **Binary classification**: two choices, normalized to zero and one.
  - e.g., guilty or innocent
- **Regression**: a one-dimensional, continuous, real-valued outcome.
  - e.g., number of days of prison assigned
- **Multinomial Classification**: Three or more discrete, un-ordered outcomes.
  - e.g., predict what judge is assigned to a case: Alito, Breyer, or Cardozo

Loss functions, more generally

- The loss function $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ assigns a score based on prediction and truth:
  - Should be bounded from below, with the minimum attained only for cases where the prediction is correct.
- The average loss for the test set is

$$\mathcal{L}(\theta) = \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i)$$

- The estimated parameter matrix θ solves

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta)$$

$\hookrightarrow$ optimizes over parameter space; treats the data as constants.

Gradient Descent

- even when cost function is not convex (eg neural nets), gradient descent often gets decent results.
- ***Stochastic* gradient descent (SGD)** computes the gradient for a single randomly sampled instance (at each iteration).
  - Much faster, still works well.

Use Cross-Validation During Model Training

- Train/Test Split:
  - ML models can achieve arbitrarily high accuracy in-sample, so performance should be evaluated out-of-sample.

Use Cross-Validation During Model Training

- Train/Test Split:
  - ML models can achieve arbitrarily high accuracy in-sample, so performance should be evaluated out-of-sample.
  - standard approach: randomly sample 80% training dataset to learn parameters, form predictions in 20% testing dataset for evaluating performance.
- Within the training set:
  - Use cross-validation with grid search to get model performance metrics across subsets of data using different hyperparameter specs.
  - Find the best hyperparameters for out-of-fold prediction in the training set.
- Then evaluate model performance in the test set using these hyperparameters.

Model Evaluation in Test Set

Evaluating a "good" model is context-dependent. Here are some basics.

Regression:

- mean squared error (MSE)
- R-squared (same ranking as MSE, but units are more interpretable)
- mean absolute error (MAE, $\sum |\hat{y}(\theta) - y|$) is less sensitive to outliers.

Model Evaluation in Test Set

Evaluating a "good" model is context-dependent. Here are some basics.

Regression:

- mean squared error (MSE)
- R-squared (same ranking as MSE, but units are more interpretable)
- mean absolute error (MAE, $\sum |\hat{y}(\theta) - y|$) is less sensitive to outliers.

Classification:

- more complicated, but accuracy is a good baseline:
  accuracy $=$ ($\#$ correct test-set predictions) / ($\#$ of test-set observations)

Model Evaluation in Test Set

Evaluating a "good" model is context-dependent. Here are some basics.

Regression:

- mean squared error (MSE)
- R-squared (same ranking as MSE, but units are more interpretable)
- mean absolute error (MAE, $\sum |\hat{y}(\theta) - y|$) is less sensitive to outliers.

Classification:

- more complicated, but accuracy is a good baseline:
  accuracy $=$ ($\#$ correct test-set predictions) / ($\#$ of test-set observations)
- What if one of the outcomes is over-represented – e.g., 19 out of 20? Then I can guess the modal class and get 95% accuracy.
  - Some alternative classifier metrics designed to address class imbalance (more below).

Regression models $\leftrightarrow$ Continuous outcome

- If the outcome is continuous (e.g., $Y =$ tax revenues collected, or criminal sentence imposed in months of prison):
  - Need a regression model.
- Problems with OLS:
  - tends to over-fit training data.
  - cannot handle multicollinearity.

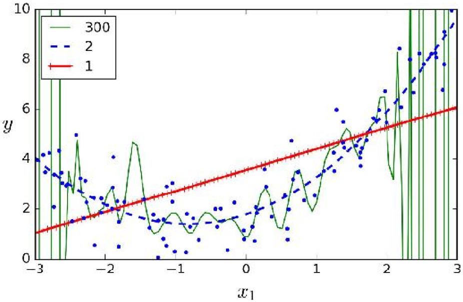


Figure 4-14. High-degree Polynomial Regression

Regression models $\leftrightarrow$ Continuous outcome

- If the outcome is continuous (e.g., $Y =$ tax revenues collected, or criminal sentence imposed in months of prison):
  - Need a regression model.
- Problems with OLS:
  - tends to over-fit training data.
  - cannot handle multicollinearity.

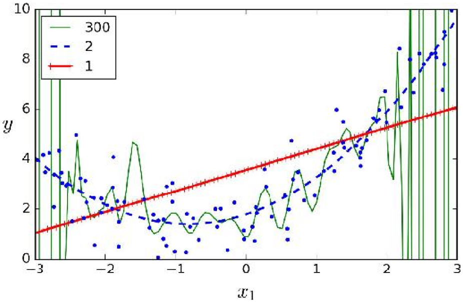


Figure 4-14. High-degree Polynomial Regression

- ***Regularization***: model training methods designed to reduce/prevent over-fitting.

Regularized Loss Function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i;\theta), \boldsymbol{y}_i) + \lambda R(\theta)$$

- $R(\theta)$ is a "regularization function" or "regularizer", designed to reduce over-fitting.
- λ is a hyperparameter where higher values increase regularization.

Regularized Loss Function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i) + \lambda R(\theta)$$

- $R(\theta)$ is a "regularization function" or "regularizer", designed to reduce over-fitting.
- λ is a hyperparameter where higher values increase regularization.

In particular:

- "Lasso" (or L1) penalty:

$$R_1 = \|\theta\|_1 = \sum_{j=1}^{n_x} |\theta_j|$$

  - shrinks coeffecents toward zero. automatically performs feature selection and outputs a sparse model.

Regularized Loss Function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i) + \lambda R(\theta)$$

- $R(\theta)$ is a "regularization function" or "regularizer", designed to reduce over-fitting.
- λ is a hyperparameter where higher values increase regularization.

In particular:

- "Lasso" (or L1) penalty:

$$R_1 = \|\theta\|_1 = \sum_{j=1}^{n_x} |\theta_j|$$

  - shrinks coefficents toward zero. automatically performs feature selection and outputs a sparse model.
- "Ridge" (or L2) penalty:

$$R_2 = \|\theta\|_2^2 = \sum_{j=1}^{n_x} (\theta_j)^2$$

  - shrinks coefficients toward zero and helps select between collinear predictors.

Regularized Loss Function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i) + \lambda R(\theta)$$

- $R(\theta)$ is a "regularization function" or "regularizer", designed to reduce over-fitting.
- λ is a hyperparameter where higher values increase regularization.

In particular:

- "Lasso" (or L1) penalty:

$$R_1 = \|\theta\|_1 = \sum_{j=1}^{n_x} |\theta_j|$$

  - shrinks coefficents toward zero. automatically performs feature selection and outputs a sparse model.
- "Ridge" (or L2) penalty:

$$R_2 = \|\theta\|_2^2 = \sum_{j=1}^{n_x} (\theta_j)^2$$

  - shrinks coefficients toward zero and helps select between collinear predictors.
- Elastic Net: $R_{\text{enet}} = \lambda_1 R_1 + \lambda_2 R_2$

Binary Outcome $\leftrightarrow$ Binary Classification

- Binary classifiers try to match a boolean outcome $y \in \{0, 1\}$.
  - The standard approach is to apply a transformation (e.g. sigmoid/logit) to normalize $\hat{y} \in [0, 1]$.
  - Prediction rule is 0 for $\hat{y} < .5$ and 1 otherwise.

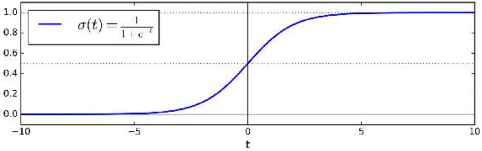
Binary Outcome $\leftrightarrow$ Binary Classification

- Binary classifiers try to match a boolean outcome $y \in \{0, 1\}$.
  - The standard approach is to apply a transformation (e.g. sigmoid/logit) to normalize $\hat{y} \in [0, 1]$.
  - Prediction rule is 0 for $\hat{y} < .5$ and 1 otherwise.
- The binary cross-entropy (or log loss) is:

$$L(\theta) = \underbrace{-\frac{1}{n_D}}_{\text{negative}} \sum_{i=1}^{n_D} [\underbrace{y_i}_{y_i=1} \underbrace{\log(\hat{y}_i)}_{\text{log prob} y_i=1} + \underbrace{(1 - y_i)}_{y_i=0} \underbrace{\log(1 - \hat{y}_i)}_{\text{log prob} y_i=0}]$$

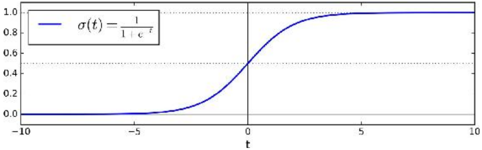
- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



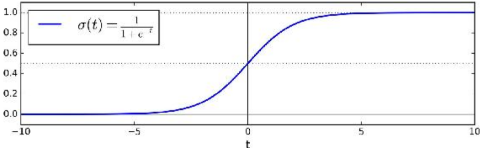
- Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

$$\min_{\theta} \sum_{i=1}^{n_D} -y_i \log(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta)) - [1 - y_i] \log(1 - \text{sigmoid}(\boldsymbol{x}_i \cdot \theta))$$

  - does not have a closed form solution, but it is convex (guaranteeing that gradient descent will find the global minimum).

- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



- Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

$$\min_{\theta} \sum_{i=1}^{n_D} -y_i \log(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta)) - [1 - y_i] \log(1 - \text{sigmoid}(\boldsymbol{x}_i \cdot \theta))$$

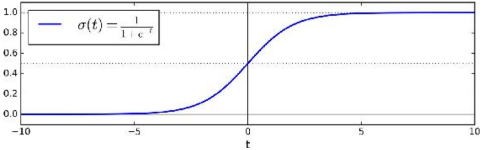
  - does not have a closed form solution, but it is convex (guaranteeing that gradient descent will find the global minimum).

- The gradient for one data point is

$$\frac{\partial L(\theta)}{\partial \theta_j} = \underbrace{(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta) - y_i)}_{\text{error for obs } i} \underbrace{x_i^j}_{\text{input } j}$$

- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



- Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

$$\min_{\theta} \sum_{i=1}^{n_D} -y_i \log(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta)) - [1 - y_i] \log(1 - \text{sigmoid}(\boldsymbol{x}_i \cdot \theta))$$

  - does not have a closed form solution, but it is convex (guaranteeing that gradient descent will find the global minimum).

- The gradient for one data point is

$$\frac{\partial L(\theta)}{\partial \theta_j} = \underbrace{(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta) - y_i)}_{\text{error for obs } i} \underbrace{x_i^j}_{\text{input } j}$$

- Like linear regression, logistic regression can be regularized with L1 or L2 penalties.

A **Confusion Matrix** is a nice way to visualize classifier performance:

		Predicted Class	
		Negative	Positive
2***True Class**	Negative	**# True Negatives**	# False Positives
	Positive	# False Negatives	**# True Positives**

- Cell values give counts in the test set.

A **Confusion Matrix** is a nice way to visualize classifier performance:

		Predicted Class	
		Negative	Positive
2***True Class**	Negative	**# True Negatives**	# False Positives
	Positive	# False Negatives	**# True Positives**

- Cell values give counts in the test set.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives} + \text{True Negatives}}$$

A **Confusion Matrix** is a nice way to visualize classifier performance:

		Predicted Class	
		Negative	Positive
2***True Class**	Negative	**# True Negatives**	# False Positives
	Positive	# False Negatives	**# True Positives**

- Cell values give counts in the test set.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives} + \text{True Negatives}}$$

$$\text{Precision (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- Precision decreases with false positives. "When I guess this outcome, I tend to guesses correctly."

A **Confusion Matrix** is a nice way to visualize classifier performance:

		Predicted Class	
		Negative	Positive
2***True Class**	Negative	**# True Negatives**	# False Positives
	Positive	# False Negatives	**# True Positives**

- Cell values give counts in the test set.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives} + \text{True Negatives}}$$

$$\text{Precision (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- Precision decreases with false positives. "When I guess this outcome, I tend to guesses correctly."

$$\text{Recall (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- Recall decreases with false negatives. "When this outcome occurs, I don't miss it."

If labels are (almost) balanced, then accuracy is a decent metric.

- If not (say 90% in one category), accuracy will be uninformative/misleading.

If labels are (almost) balanced, then accuracy is a decent metric.

- If not (say 90% in one category), accuracy will be uninformative/misleading.

Balanced accuracy = the average recall in both classes:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}}\right)$$

- $\rightarrow$ equal to accuracy when classes are balanced, or when performance is the same across classes.

If labels are (almost) balanced, then accuracy is a decent metric.

- If not (say 90% in one category), accuracy will be uninformative/misleading.

Balanced accuracy = the average recall in both classes:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}}\right)$$

- $\rightarrow$ equal to accuracy when classes are balanced, or when performance is the same across classes.

F_1 **score** = the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

- penalizes both false positives and false negatives; still ignores true negatives.

If labels are (almost) balanced, then accuracy is a decent metric.

- If not (say 90% in one category), accuracy will be uninformative/misleading.

Balanced accuracy = the average recall in both classes:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}}\right)$$

- $\rightarrow$ equal to accuracy when classes are balanced, or when performance is the same across classes.

F_1 **score** = the harmonic mean of precision and recall:

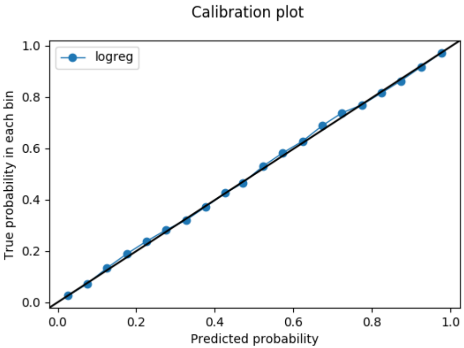
$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

- penalizes both false positives and false negatives; still ignores true negatives.

AUC-ROC = Area Under the Receiver Operating Characteristic Curve

- provides an aggregate measure of performance across all possible classification thresholds.
- Interpretation: randomly sample one positive and one negative example. AUC = probability that the model correctly guesses which is which.

Evaluating Classification Models: Calibration Curves



- Plotting the binned fraction in a category (Y axis) against the predicted probability in a category (X axis):
- Provides evidence of whether the classifer is replicating the conditional distribution of the outcome.

Multiple Classes: Setup

- The outcome is $y_i \in \{1, ..., k, ..., n_y\}$ output classes, which can also be represented as a one-hot vector

$$\boldsymbol{y}_i = \{\mathbf{1}[y_i = 1], ..., \mathbf{1}[y_i = n_y]\}$$

Multiple Classes: Setup

- The outcome is $y_i \in \{1, ..., k, ..., n_y\}$ output classes, which can also be represented as a one-hot vector
$$\boldsymbol{y}_i = \{\mathbf{1}[y_i = 1], ..., \mathbf{1}[y_i = n_y]\}$$
- We want to learn a vector function
$$\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \theta)$$
taking text features $\boldsymbol{x}$ as inputs and outputting a vector of probabilities across outcome classes:
$$\hat{\boldsymbol{y}} = \{\hat{y}^1, ..., \hat{y}^{n_y}\}, \sum_{k=1}^{n_y} \hat{y}^k = 1, \hat{y}^k \geq 0 \ \forall k$$
- for prediction step, can select the highest-probability class:
$$\tilde{y} = \arg\max_k \hat{y}_{[k]}$$
- The standard loss function in multinomial classification is **categorical cross entropy**

Multinomial Logistic Regression

Multinomial logistic regression computes probabilities for each class k using the softmax transformation

$$\hat{y}_k(\boldsymbol{x}_i) = \Pr(y_i = k) = \frac{\exp(\theta_k' \boldsymbol{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta_l' \boldsymbol{x}_i)}$$

- softmax is the multiclass generalization of sigmoid $\rightarrow$ can then interpret $\hat{y}$ as probabilities.
- n_x features and n_y output classes $\rightarrow$ there is a $n_y \times n_x$ parameter matrix Θ, where the parameters for each class θ_k are stored as rows.

Multinomial Logistic Regression

Multinomial logistic regression computes probabilities for each class k using the softmax transformation

$$\hat{y}_k(\boldsymbol{x}_i) = \Pr(y_i = k) = \frac{\exp(\theta_k' \boldsymbol{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta_l' \boldsymbol{x}_i)}$$

- softmax is the multiclass generalization of sigmoid $\rightarrow$ can then interpret $\hat{y}$ as probabilities.
- n_x features and n_y output classes $\rightarrow$ there is a $n_y \times n_x$ parameter matrix Θ, where the parameters for each class θ_k are stored as rows.

The **L2-penalized logistic regression** has loss function

$$\mathcal{L}(\theta) = -\frac{1}{n_D} \sum_{i=1}^{n_D} \log \frac{\exp(\theta_{k*}' \boldsymbol{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta_l' \boldsymbol{x}_i)} + \lambda \sum_{j=1}^{n_x} \sum_{k=1}^{n_y} (\theta_{[j,k]})^2$$

- $\lambda =$ strength of L2 penalty (could also add lasso penalty)
  - as before, predictors should be scaled to the same variance.

		Predicted Class		
		Class A	Class B	Class C
3***True Class**	Class A	Correct A	A, classed as B	A, classed as C
	Class B	B, classed as A	Correct B	B, classed as C
	Class C	C, classed as A	C, classed as B	Correct C

More generally, with **multi-class confusion matrix** M with items M_{ij} (row i, column j):

$$\text{Precision for } k = \frac{\text{True Positives for } k}{\text{True Positives for k} + \text{False Positives for } k} = \frac{M_{kk}}{\sum_l M_{lk}}$$

$$\text{Recall for } k = \frac{\text{True Positives for } k}{\text{True Positives for } k + \text{False Negatives for } k} = \frac{M_{kk}}{\sum_l M_{kl}}$$

$$F_1(k) = 2 \times \frac{\text{precision}(k) \times \text{recall}(k)}{\text{precision}(k) + \text{recall}(k)}$$

		Predicted Class		
		Class A	Class B	Class C
3***True Class**	Class A	Correct A	A, classed as B	A, classed as C
	Class B	B, classed as A	Correct B	B, classed as C
	Class C	C, classed as A	C, classed as B	Correct C

More generally, with **multi-class confusion matrix** M with items M_{ij} (row i, column j):

$$\text{Precision for } k = \frac{\text{True Positives for } k}{\text{True Positives for k} + \text{False Positives for } k} = \frac{M_{kk}}{\sum_l M_{lk}}$$

$$\text{Recall for } k = \frac{\text{True Positives for } k}{\text{True Positives for } k + \text{False Negatives for } k} = \frac{M_{kk}}{\sum_l M_{kl}}$$

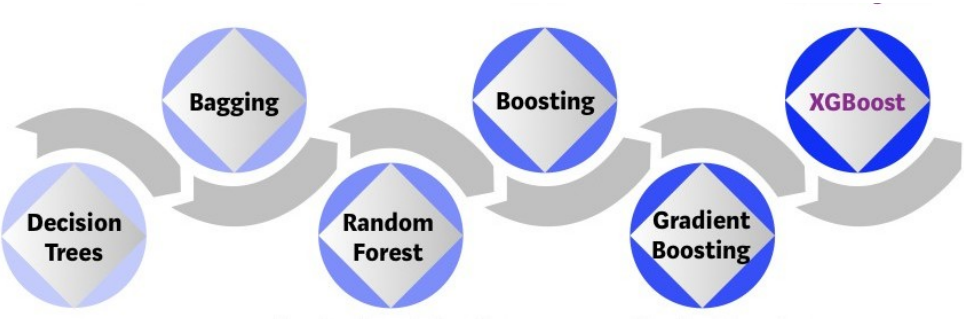
$$F_1(k) = 2 \times \frac{\text{precision}(k) \times \text{recall}(k)}{\text{precision}(k) + \text{recall}(k)}$$

Can average these metrics across classes to get aggregate metrics.

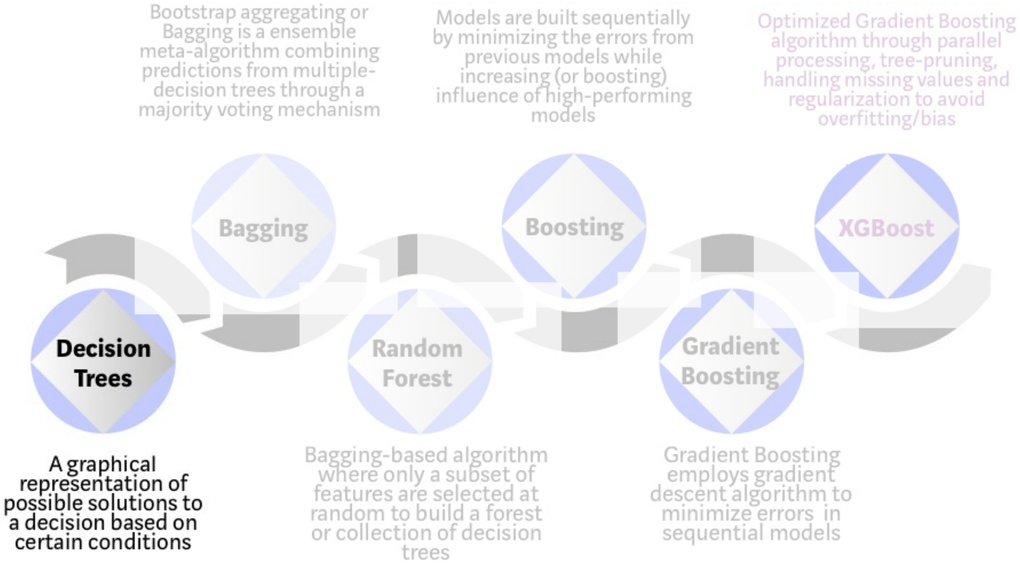
- e.g., balanced accuracy = unweighted average of recalls across classes.
- can weight classes by their frequency in dataset

Outline

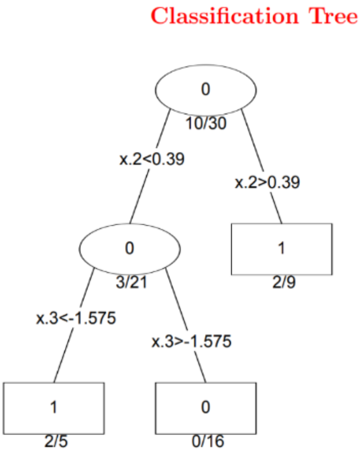
XGBoost: Overview



XGBoost Ingredients: Decision Trees

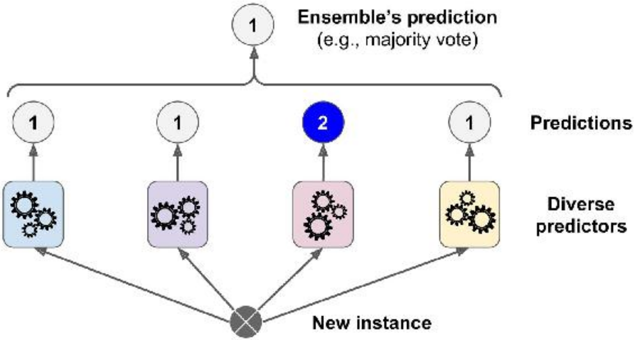


Decision Trees



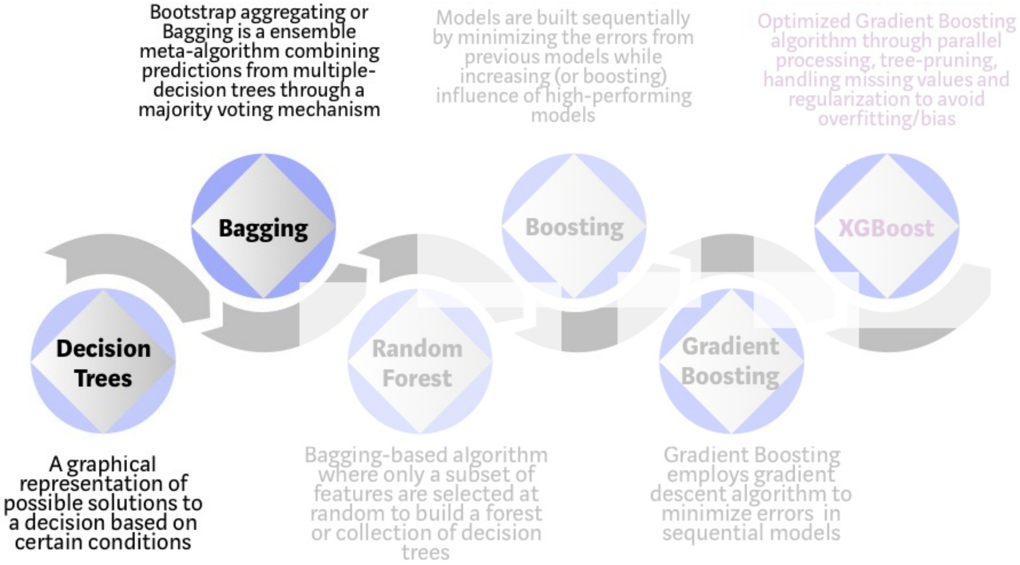
- Decision trees learn a series of binary splits in the data based on hard thresholds.
  - if yes, go right; if no, go left.
- Can have additional splits as you move through the tree.
- fast and interpretable, but performance is often poor.

Voting Classifiers



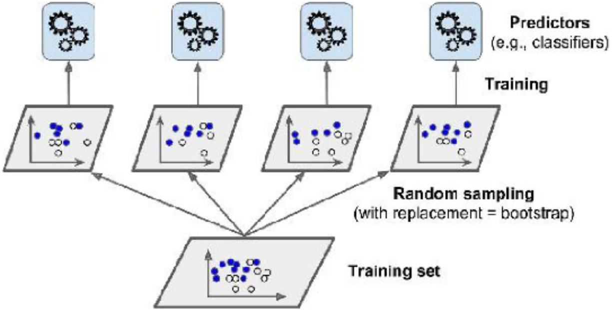
- voting classifiers (ensembles of different models that vote on the prediction) generally out-perform the best classifier in the ensemble.
  - more diverse algorithms will make different types of errors, and improve your ensemble's robustness.

XGBoost Ingredients: Bootstrapping



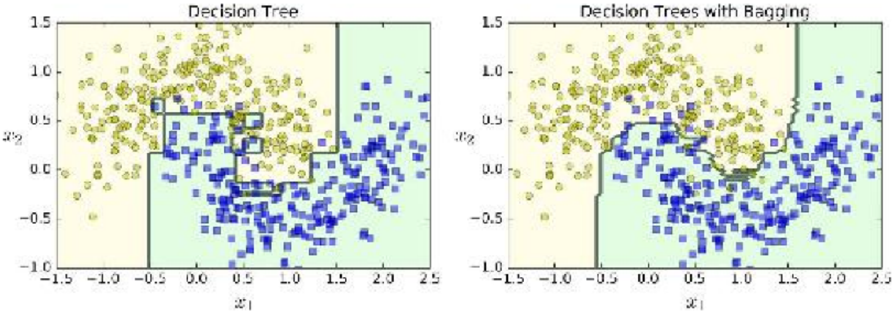
Bootstrapping

- Rather than use the same data on different classifiers, one can use different subsets of the data on the same classifier:



  - can also use different subsets of features across subclassifiers.

Bootstrapping Benefits



- A bootstraped ensemble generally has a similar bias but lower variance than a single predictor trained on all the data.
- Predictors can be trained in parallel using separate CPU cores.

XGBoost Ingredients: Random Forests

Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism

Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models

Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias

A graphical representation of possible solutions to a decision based on certain conditions

Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees

Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

1. Each voting tree gets its own sample of data.

Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

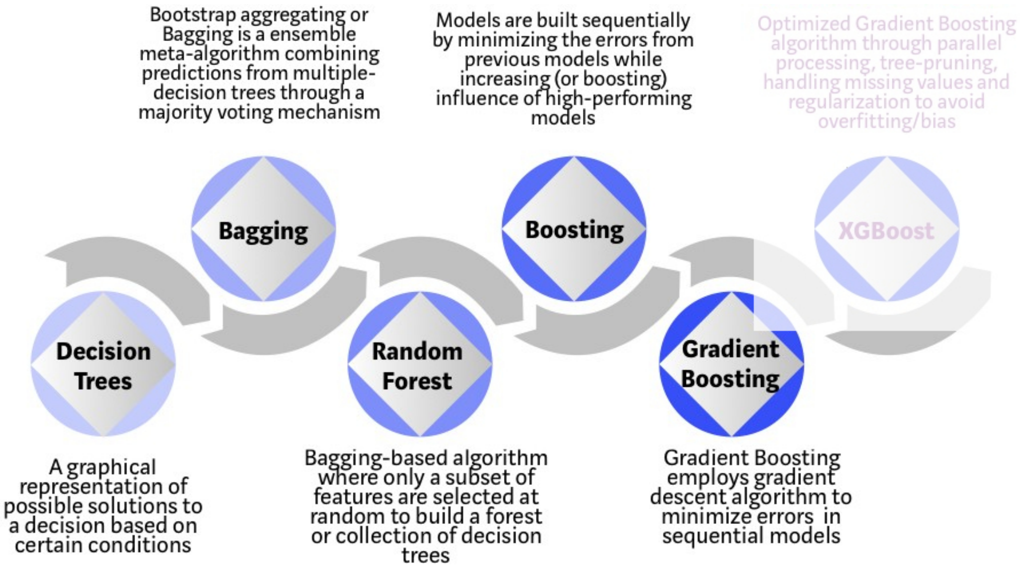
1. Each voting tree gets its own sample of data.
2. At each tree split, a random sample of features is drawn, only those features are considered for splitting.

Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

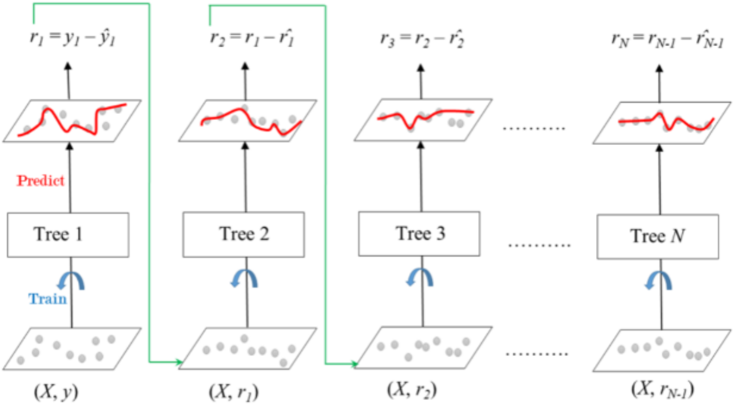
1. Each voting tree gets its own sample of data.
2. At each tree split, a random sample of features is drawn, only those features are considered for splitting.
3. For each tree, error rate is computed using data outside its bootstrap sample.

XGBoost Ingredients: Gradient Boosting



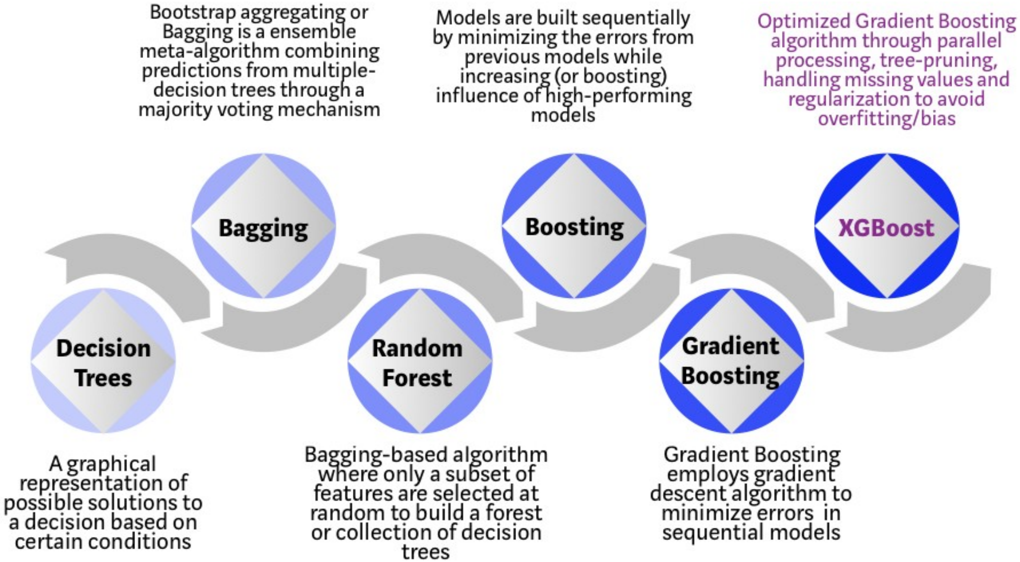
Gradient Boosting Machines

- Gradient boosting refers to an additive ensemble of trees:



- Adds additional layers of trees to fit the residuals of the first layers

XGBoost Ingredients

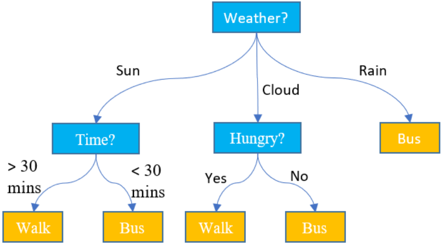


XGBoost

- Feurer et al (2018) find that XGBoost beats a sophisticated AutoML procedure with grid search over 15 classifiers and 18 data preprocessors.
- A good starting point for any machine learning task.
- easy to use
- actively developed
- efficient / parallelizable
- provides model explanations
- takes sparse matrices as input

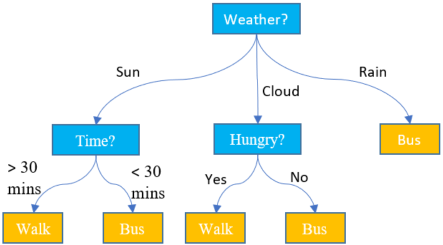
Tree Ensembles are Black Boxes

- Small decision trees have the advantage of being highly interpretable.



Tree Ensembles are Black Boxes

- Small decision trees have the advantage of being highly interpretable.



- Larger trees and ensembles (e.g. XGBoost) lose this nice feature.
- Best-performing ML models are hard to interpret because they use lots of features and exploit non-linearities and interactions.

Interpreting Tree Ensembles

XGBoost's Feature Importance Metric:

- At each decision node, compute **information gain** for feature j **(change in predicted probability**).
- Average across all nodes for each j.

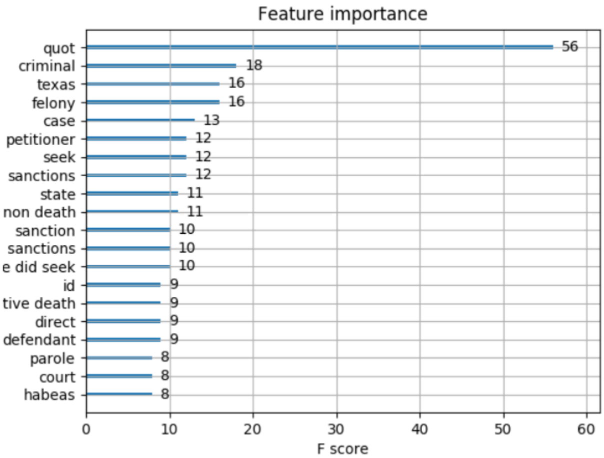
Ranks predictors by their relative contributions.

```
from xgboost import plot_importance
plot_importance(xgb_reg, max_num_features=10)
```

Feature Importance

```python
from xgboost import plot_importance
plot_importance(xgb_reg, max_num_features=20)
```

```
<IPython.core.display.Javascript object>
```



A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - *(If y is more complicated, e.g. a sequence of words, we use deep learning.)*

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - *(If y is more complicated, e.g. a sequence of words, we use deep learning.)*
3. Use cross-validation grid search in training set to select model hyperparameters.
   - For classification, use cross entropy; for regression, use mean squared error.

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - *(If y is more complicated, e.g. a sequence of words, we use deep learning.)*
3. Use cross-validation grid search in training set to select model hyperparameters.
   - For classification, use cross entropy; for regression, use mean squared error.
4. Evaluate model in held-out test set:
   - For classification, use balanced accuracy, confusion matrix, and calibration plot.
   - For regression, use R squared and binscatter plot.

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - *(If y is more complicated, e.g. a sequence of words, we use deep learning.)*
3. Use cross-validation grid search in training set to select model hyperparameters.
   - For classification, use cross entropy; for regression, use mean squared error.
4. Evaluate model in held-out test set:
   - For classification, use balanced accuracy, confusion matrix, and calibration plot.
   - For regression, use R squared and binscatter plot.
5. Interpret the model predictions:
   - for gradient boosting, use feature importance ranking.
   - for linear models, examine coefficients
   - look at highest and lowest ranked documents for $\hat{y}$

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - *(If y is more complicated, e.g. a sequence of words, we use deep learning.)*
3. Use cross-validation grid search in training set to select model hyperparameters.
   - For classification, use cross entropy; for regression, use mean squared error.
4. Evaluate model in held-out test set:
   - For classification, use balanced accuracy, confusion matrix, and calibration plot.
   - For regression, use R squared and binscatter plot.
5. Interpret the model predictions:
   - for gradient boosting, use feature importance ranking.
   - for linear models, examine coefficients
   - look at highest and lowest ranked documents for $\hat{y}$
6. Answer the research question!