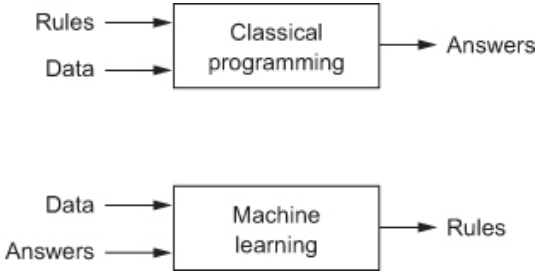
Language Models for Law and Social Science

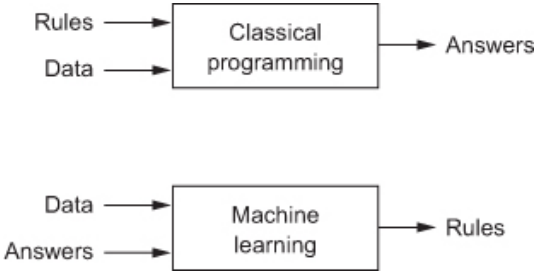
4. Supervised Learning with Text

What is machine learning?



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

What is machine learning?



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

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

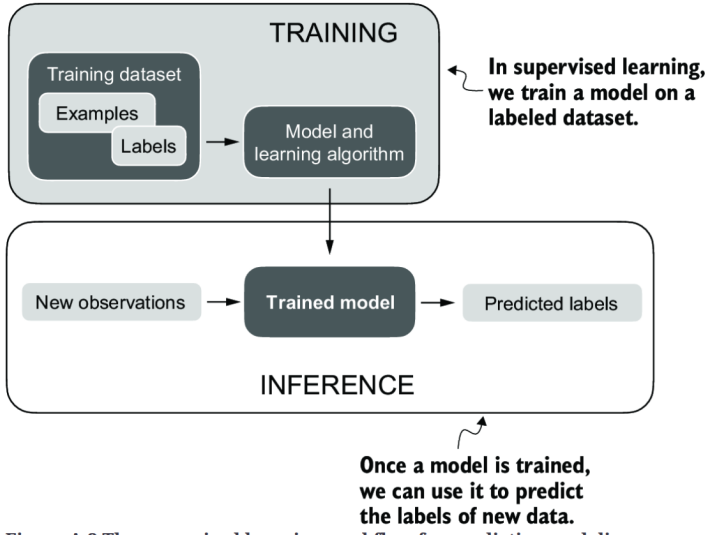


Figure A.3 The supervised learning workflow for predictive modeling consists of a training stage where a model is trained on labeled examples in a training dataset. The trained model can then be used to predict the labels of new observations.

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

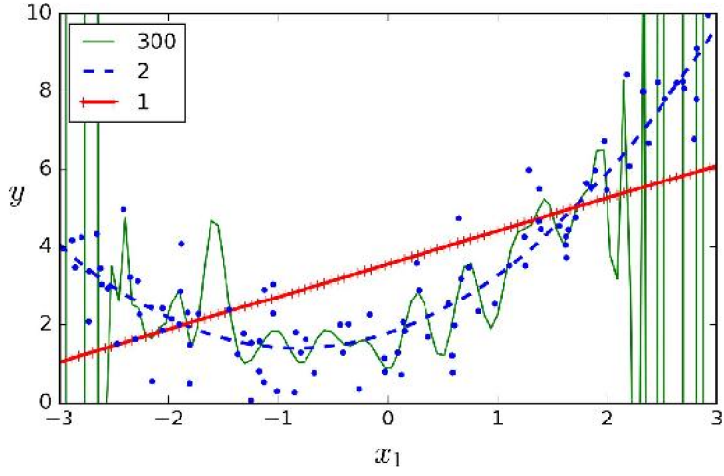


Figure 4-14. High-degree Polynomial Regression

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.

e.g., 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)

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

e.g., 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).

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

- 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.

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

- The partial derivative for feature j is

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

  - $\rightarrow$ 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.

Data Prep for Machine Learning

Data Prep for Machine Learning

- Data Pre-Processing: See Geron Chapter 2 for pandas and sklearn syntax:
  - imputing missing values.
  - feature scaling (often helpful/necessary for ML models to work well)
    - if predictors are sparse (e.g. bag-of-words), use `StandardScaler(with_mean=False)`.
  - encoding categorical variables.
  - **Best practice: reproducible data pipeline**.

Data Prep for Machine Learning

- Data Pre-Processing: See Geron Chapter 2 for pandas and sklearn syntax:
  - imputing missing values.
  - feature scaling (often helpful/necessary for ML models to work well)
    - if predictors are sparse (e.g. bag-of-words), use `StandardScaler(with_mean=False)`.
  - encoding categorical variables.
  - **Best practice: reproducible data pipeline**.
- 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.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
```

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.

Text Features as ML Features

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

Text Features as ML Features

- Each document is a sequence of symbols d_i, while (standard) ML algorithms work on numbers.
- The solution: all the methods from Weeks 1, 2, 3 for extracting informative numerical information from documents:
  - style features
  - counts over dictionary patterns
  - tokens
  - n-grams
  - principal components
  - topic shares
  - etc.
- Represent documents as a matrix $\boldsymbol{x}$ with $n_x \geq 1$ features.

Supervised Feature Selection

- N-grams and many other featurizers produce a lot of features that are not useful for the task.
- If dimensionality is a problem, use supervised feature selection (**sklearn.feature_selection**) to select informative predictors.

Supervised Feature Selection

- N-grams and many other featurizers produce a lot of features that are not useful for the task.
- If dimensionality is a problem, use supervised feature selection (**sklearn.feature_selection**) to select informative predictors.
- e.g., for classification (discrete labels), use chi2 (chi-squared metric)
  - very fast, works on sparse matrices
  - features must be non-negative (with neg predictors, use `f_classif`)
- For regression tasks (continuous outcome), use `f_regression`.
- **Remember: only in the training set**

Three Types of (Standard) Machine Learning Problems

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

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

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.

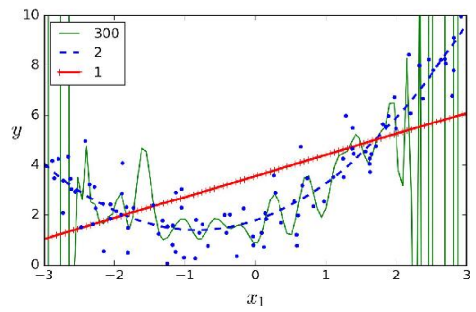


Figure 4-14. High-degree Polynomial Regression

- ***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 coefficents 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.

- **Elastic Net**: $R_{\text{enet}} = \lambda_1 R_1 + \lambda_2 R_2$

Evaluating Regression Models

```
1  from sklearn.metrics import mean_squared_error, r2_score
2
3  # model evaluation for training set
4  y_train_predict = house_model.predict(X_train)
5  rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
6  r2 = r2_score(y_train, y_train_predict)
7
8  print('RMSE for training is ', rmse)
9  print('R2 score for training is ', r2)
10
11 y_test_predict = house_model.predict(X_test)
12 rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))
13 r2 = r2_score(y_test, y_test_predict)
14
15 print('RMSE for testing is ', rmse)
16 print('R2 score for testing is ',r2)
```

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.

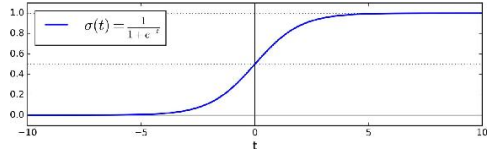
Binary Outcome $\leftrightarrow$ Binary Classification

- 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:

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

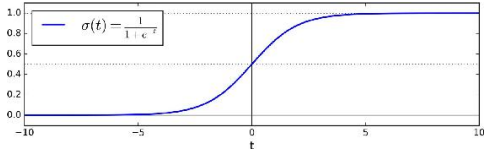
- In **logistic regression** we use a sigmoid transformation:

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



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

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

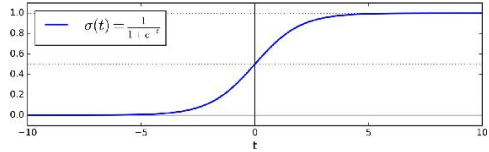


- Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

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

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

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



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

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

  - does not have a closed form solution, but we can use the loss gradient

$$\nabla_\theta L(\theta) = \sum_{i=1}^{n} \left(\sigma(\boldsymbol{x}_i \cdot \theta) - y_i\right) \boldsymbol{x}_i$$

    and gradient descent to find the global minimum (since the loss function is convex) .

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

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

		Predicted Class	
		Negative	Positive
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
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
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
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:

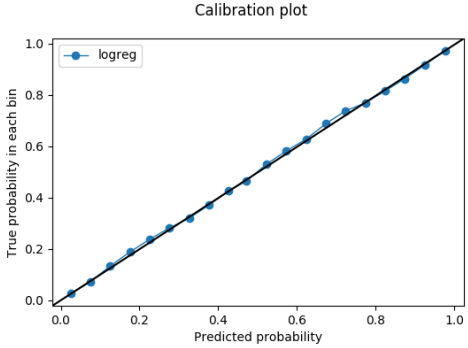
$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

- 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.

```
from seaborn import regplot
regplot(y_test, y_pred, x_bins=20)
```

Application: Predicting Political Party from Text

Andrew Peterson and Arthur Spirling, "Classification accuracy as a substantive quantity of interest: Measuring polarization in Westminster systems," *Political Analysis* (2018).

Application: Predicting Political Party from Text

Andrew Peterson and Arthur Spirling, "Classification accuracy as a substantive quantity of interest: Measuring polarization in Westminster systems," *Political Analysis* (2018).

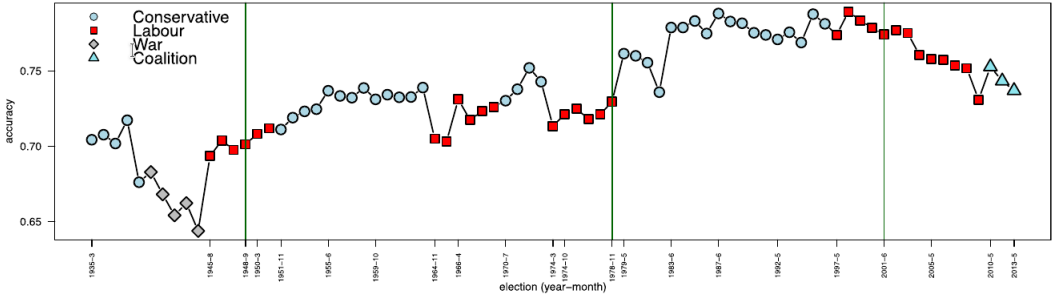
- Machine Learning Problem:
  - Corpus D = 3.5M U.K. parliament speeches, 1935-2013.
  - Label Y = party of speaker (Conservative or Labour)

Application: Predicting Political Party from Text

Andrew Peterson and Arthur Spirling, "Classification accuracy as a substantive quantity of interest: Measuring polarization in Westminster systems," *Political Analysis* (2018).

- Machine Learning Problem:
  - Corpus $D = 3.5$M U.K. parliament speeches, 1935-2013.
  - Label $Y =$ party of speaker (Conservative or Labour)

In years that classifier is more accurate, speech is more polarized:



Multi-Class Classification

Multi-Class Classification

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

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

Multi-Class Classification

- 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 outputing 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:

$$\tilde{y} = \arg\max_k \hat{y}_{[k]}$$

Categorical Cross Entropy

- The standard loss function in multinomial classification is **categorical cross entropy**:

$$L(\theta) = -\sum_{k=1}^{n_y} \boldsymbol{y}^k \log(\hat{y}^k(\boldsymbol{x}, \theta))$$

  - measures dissimilarity between the true label distribution $\boldsymbol{y}$ and the predicted label distribution $\hat{\boldsymbol{y}}$.

Categorical Cross Entropy

- The standard loss function in multinomial classification is **categorical cross entropy**:

$$L(\theta) = -\sum_{k=1}^{n_y} \boldsymbol{y}^k \log(\hat{y}^k(\boldsymbol{x}, \theta))$$

  - measures dissimilarity between the true label distribution $\boldsymbol{y}$ and the predicted label distribution $\hat{\boldsymbol{y}}$.

- Since there is just one true class ($y = 1$ for one class k^*, and zero for others), simplifies to

$$L(\theta) = -\log(\hat{y}^{k^*}(\boldsymbol{x}, \theta))$$

  - Rewards putting higher probability on the true class, ignores distribution of probabilities on other classes.

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

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

- λ = 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
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
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

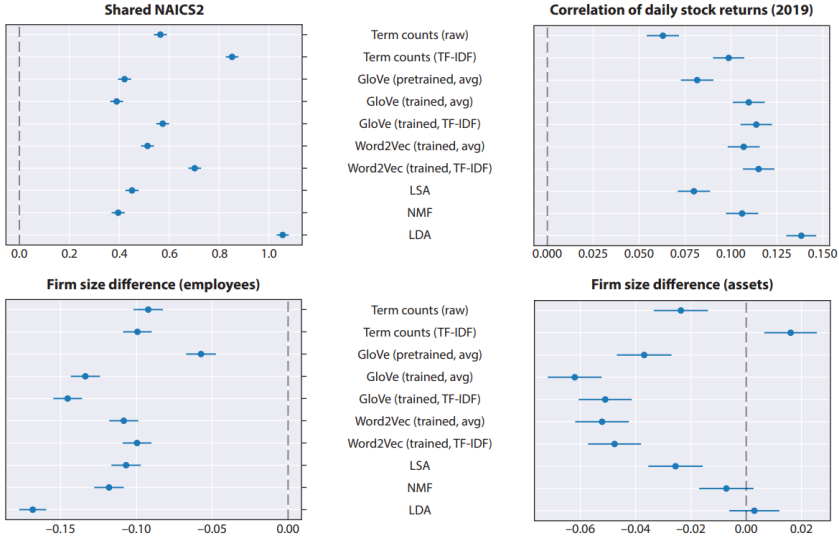


Figure 4

Impact of algorithm on downstream regression coefficient estimates. We draw 50,000 random pairs of firms among the population for which we can retrieve a 2019 Risk Factors section and a stock price for every trading day in 2019 from the Center for Research in Security Prices. For each algorithm, we then compute the pairwise similarity between each firm's texts, which we regress on a dummy variable for shared NAICS2 sector; the correlation between daily returns in 2019; the absolute log ratio of employees; and the absolute log ratio of total assets. The data on sector and firm size come from COMPUSTAT. The panels in the figure display the point estimates and 95% confidence intervals for each regression coefficient and each algorithm. In all regressions, continuous covariates are expressed in standard deviation units. Abbreviations: LDA, latent Dirichlet allocation; LSA, latent semantic analysis; NAICS2, North American Industry Classification System 2; NMF, nonnegative matrix factorization; TF-IDF, term frequency-inverse document frequency.

Overview of Text Analysis Methods (Osnabruegge et al 2021)

	Dictionaries (Custom)	Dictionaries (Generic)	Topic Modeling	Within-Domain Supervised Learning	Cross-Domain Supervised Learning
Design/Annotation Costs	High	Low	Low	High	Moderate
Specificity	High	Moderate	Low	High	Moderate
Interpretability	High	High	Moderate	High	High
Validatability	Low	Low	Low	High	High

Review: NLP "Bias" is statistical bias

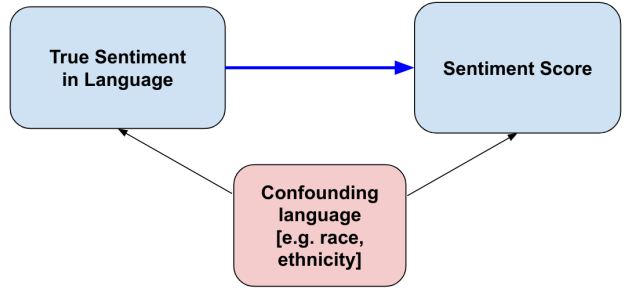
- Supervised learning algorithms trained on annotated datasets learn both the annotated label and other language information that is correlated with the label (confounders).

Review: NLP "Bias" is statistical bias

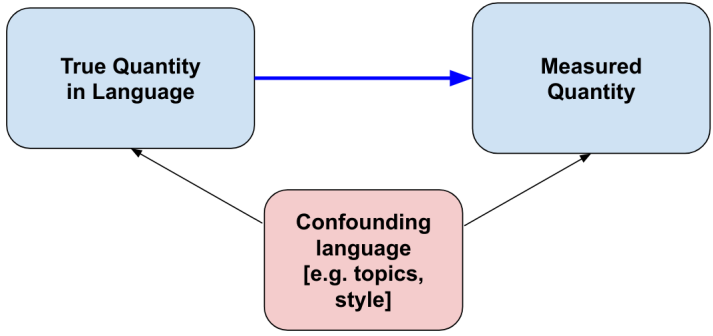
- Supervised learning algorithms trained on annotated datasets learn both the annotated label and other language information that is correlated with the label (confounders).
- e.g., sentiment scores that are trained on annotated datasets also learn from the correlated non-sentiment information:

```
text_to_sentiment("Let's go get Italian food")
2.0429166109
text_to_sentiment("Let's go get Chinese food")
1.4094033658
text_to_sentiment("Let's go get Mexican food")
0.3880198556
```

```
text_to_sentiment("My name is Emily")
2.2286179365
text_to_sentiment("My name is Heather")
1.3976291151
text_to_sentiment("My name is Yvette")
0.9846380213
text_to_sentiment("My name is Shaniqua")
-0.4704813178
```



Review: Classifiers Learn Confounders



- supervised models (classifiers, regressors) learn features that are correlated with the label being annotated.
  - when taken to new data, the predicted label might reflect the presence of the confounders, rather than the presence of the true label.

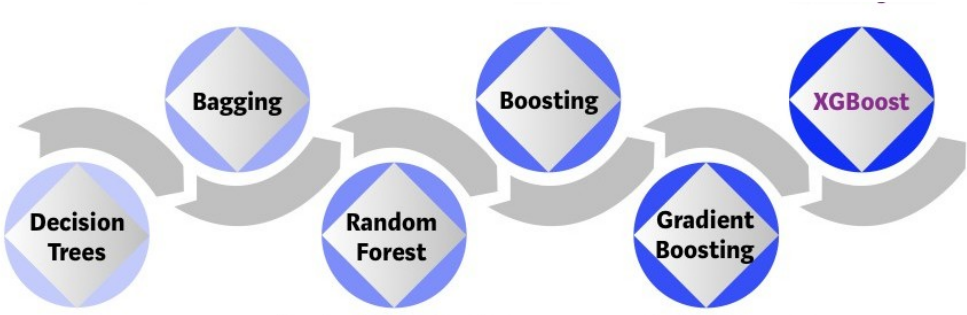
Machine Learning Essentials

Ensemble Learning with XGBoost

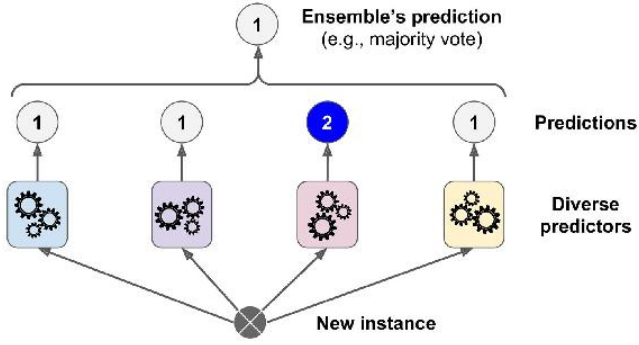
Media Slant is Contagious

Deep Learning Essentials

XGBoost: Overview



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.

Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X,y)
```

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

- 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

```
from xgboost import XGBClassifier
model = XGBClassifier()

model.fit(X_train, y_train,
          early_stopping_rounds=10,
          eval_metric="logloss",
          eval_set=[(X_eval, y_eval)]
          )

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Simple machine learning with long documents

1. Take TF-IDF bigrams as inputs X.
   - if too high-dimensional, use supervised feature selection and/or filter on parts of speech
2. Select a machine learning model for predicting outcome y:
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
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!

Machine Learning Essentials

Ensemble Learning with XGBoost

Media Slant is Contagious

Deep Learning Essentials

Media Slant is Contagious

80 Pages • Posted: 2 Dec 2020 • Last revised: 17 Feb 2025

Philine Widmer
Paris School of Economics (PSE); University of St.Gallen

Clémentine Abed Meraim
ETH Zürich

Sergio Galletta
ETH Zürich

Elliott Ash
ETH Zürich

This Paper

- **Does media slant spread to other outlets?**
- **Empirical context:** We ask whether the media slant of cable news TV channels spreads to local newspapers in the United States.
- **Measuring slant:** Based on a machine learning model that predicts, for a given body of text, whether it resembles the language by the relatively conservative network (FNC), rather than language by the relatively liberal networks (CNN or MSNBC)
- **Empirical strategy:** we use IV instrumenting county-level TV viewership by the positioning of station in channel system across counties (Martin and Yurukoglu AER, 2017)
- **Media slant is contagious**: Higher exposure to Fox News Channel (relative to CNN/MSNBC) causally increases the linguistic similarity between local newspaper content and FNC content

Data Overview (2005-2008)

- **Observation unit:** newspaper i in county j in state s
  - 305 unique local newspaper titles
  - On average, 12.4 counties matched to a newspaper
  - $\rightarrow$ 3,781 newspaper-county observations
- **Main regressor:** instrumented cable TV viewership
  - FNC viewership relative to CNN/MSNBC viewership
  - Channel positions and viewership from Nielsen (Martin and Yurukoglu AER, 2017)
  - Aggregated at the county level Weighted by TV-watching households in the zip code
- **Outcome:** $Slant_{ijs}$ is the language similarity between newspaper text and FNC(vs CNN/MSNBC) transcripts

NLP Methods Overview

What we want: Predicted textual similarity between newspaper texts i to FNC(vs CNN/MSNBC) $\widehat{FNC}_i$

What we start with

- 16M newspaper article snippets from NewsLibrary
  - Contains outlet name, article date, byline, headline, and the first 80 words of the article
- 40K cable news episodes from LexisNexis
  - Contains text and metadata; text segmented into 1.6M 80-word snippets

Text Pre-Processsing

- Pre-processing of newspaper and transcripts snippets
  - Convert to lower case
  - Remove non-meaningful stopwords and non-letter characters
  - Replace tokens by their stem (Porter algorithm)
- Form bigrams from processed tokens (65K bigrams)
  - $\rightarrow$ Which of them are predictive for FNC or CNN/MSNBC?
  - Minimum frequency threshold (20x per network) to filter out uninformative content (e.g., host names)

Preparing the Datasets for Machine Learning

- Oversample FNC transcripts to get a balanced sample (50% FNC and 50% CNN/MSNBC)
- Shuffle dataset into 80% training set and 20% test set
- Supervised feature selection in training set
  - Rank all features by correlation (χ^2) with Fox vs. CNN/MSNBC
  - Keep the 2,000 most predictive bigrams

Processed Bigrams Associated with FNC or CNN/MSNBC

Table: Bigrams associated with FNC or CNN/MSNBC (χ^2)

Fox News	**CNN/MSNBC**
american troop	gay communiti
crime punish	crime alleg
immigr author	politico report
nation economi	economi world
support gun	woman black

Logistic Regression Model

- Parametrize probability that transcript m is from FNC based on bigrams

$$\widehat{FNC}_m = \Pr[FNC_m = 1 | B_m] = \frac{1}{1 + \exp(-\psi' B_m)}$$

  - B_m: 2000-vector of bigram frequencies in m
  - ψ: 2000-vector of coefficients from Logistic regression

- Test-set accuracy is 73%, better than guessing (50%) and similar to human judgement more

Test Set Accuracy

	Predicted CNN	Predicted FNC
Actual CNN	**38.3%** (235K)	11.7% (72K)
Actual FNC	15.0% (92K)	**35.0%** (215K)

- Similar to accuracy of human judgment $\approx$ 76%.
- In a sample of 1000 80-word snippets, three human coders agreed on the correct outcome 58% of the time.

back

Predict Slant of Newspaper Texts

Take model trained on **TV transcripts** to predict textual similarity with Fox News for **newspaper article** snippets

- A_n: 2000-vector of bigram frequencies in snippet n
- For n, predict textual similarity to Fox News as

$$\widehat{Fox}_n = \Pr[Fox = 1|A_n] = \frac{1}{1+\exp(-\hat{\psi}' A_n)}$$

- Aggregate $\widehat{Fox}_n$ to newspaper-county level: $\widehat{Fox}_{ijs}$

s

Predict Slant of Newspaper Texts

Table: Articles with high/low similarity to FNC shows

Similar with FNC: The Sacramento Bee (CA), 97% similarity to FNC

Don Kercell thinks he's earned a second chance. The Contractors State License Board does not agree. And therein lies a tale of choices and consequences; crime and punishment; addiction and rehabilitation; public protection and personal redemption – and second chances. Kercell is a 48-year-old resident of Rio Linda. In his youth, he discovered two things. One was that he had a talent for working with concrete. The other was methamphetamine. [...]

Similar with CNN/MSNBC: The Sun (San Bernardino, CA), 3% similarity to FNC

REDLANDS - A week after a state judge ruled that banning gay marriage is unconstitutional, students at University of Redlands will celebrate the milestone along with continued efforts to raise awareness of the gay community.The PRIDE Alliance, a campus group devoted to promoting tolerance on campus for gay, lesbian, bisexual and transgender students, will celebrate PRIDE Week at the university through Friday. A series of events is scheduled to raise awareness on campus and in the [...]

Identification Strategy

OLS:

$$\text{Slant}_{ijs} = \alpha_s + \theta \text{Viewership}_{js} + X_{ijs}\beta + \epsilon_{ijs} \quad (1)$$

First-stage regression:

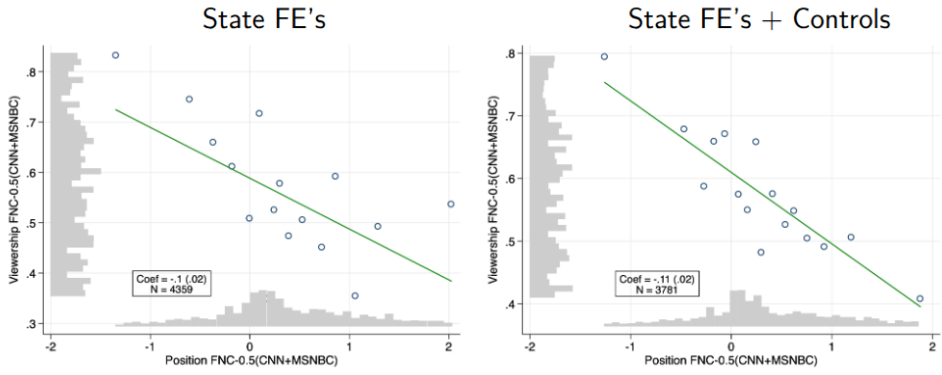
$$\text{Viewership}_{js} = \alpha_s + \delta \text{Position}_{js} + X_{ijs}\beta + \eta_{ijs} \quad (2)$$

Two-stage least squares (2SLS):

$$\text{Slant}_{ijs} = \alpha_s + \theta \widehat{\text{Viewership}}_{js} + X_{ijs}\beta + \epsilon_{ijs} \quad (3)$$

First Stage: FNC Channel Position and Viewership

Figure: FNC Position and FNC Viewership



Notes: Binned scatterplots (16 bins) of standardized viewership of FNC-0.5(CNN+MSNBC) against standardized position of FNC-0.5(CNN+MSNBC). Cross-section with newspaper-county-level observations weighted by newspaper circulation in each county. On the left, state fixed effects are included. On the right, state fixed effects, as well as demographic controls, channel controls, and generic newspaper language controls are included.

Identification Assumptions

- Exclusion Restriction: Channel position affects FNC/newspaper similarity only through shifting viewership
- Exogeneity: Channel position not related to other local factors that influence FNC/newspaper similarity
- Empirical checks consistent with assumptions
  - Channel position not associated with demographics
  - Placebo: No 2SLS effect of FNC viewership on similarities of newspaper articles in 1995/1996 (before introduction of FNC) with FNC language

Channel Position not Associated with Demographics Predictive of Newspaper Content

Table: Identification checks: Instrument Uncorrelated with Relevant Covariates

	Reduced form			
	Viewership FNC* (absolute)	$Slant_{ijs}$*	Viewership FNC* (rel. to CNN/MSNBC)	$Slant_{ijs}$*
	(1)	(2)	(3)	(4)
FNC position (absolute)	-0.022	0.017		
	(0.043)	(0.013		
FNC position (rel. to CNN/MSNBC)			0.006	-0.006
			(0.025)	(0.006)
N observations	3781	3781	3781	3781
State FE	X	X	X	X

Standard errors are multiway-clustered at the county and at the newspaper level (in parenthesis): * $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$.

Main Results

Table: Cable News Effects on Newspaper Content (2SLS)

Dep. variable: Slant_{ijs}=Pr(*FNC*|Text_{ijs})	(1)	(2)	(3)
FNC Viewership (rel. to CNN/MSNBC)	0.314***	0.311***	0.318**
	(0.114)	(0.113)	(0.126)
K-P First-Stage F-stat	36.553	36.298	34.147
N observations	3781	3781	3781
State FE	X	X	X
Demographic controls	X	X	X
Channel controls		X	X
Newspaper language controls			X

Standard errors are multiway-clustered at the county and at the newspaper level (in parenthesis): * $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$.

Robustness Checks

- Effects present when focusing on newspaper headquarters counties
- Similar patterns for absolute FNC viewership and FNC viewership relative to CNN and MSNBC separately
- Alternative matching of newspapers to counties
- Relative instead of absolute circulation weights and by Nielsen users weights
- Standard error clustering at state level
- Controlling for pre-FNC endorsements
- Dropping AP articles
- Dropping each state or newspaper individually

Further Results

- Cable network slant is also contagious for local news more
- Contagious slant polarizes local newspapers more
- Importantly: no effects of TV channels on generic language features (vocab size, average word length, average sentence length, article length)
- Framing rather than topic choice: effects largely unchanged when controlling for LDA topic shares

Take Aways

- Partisan cable news influenced the content of local news channels
- New evidence on how partisan media influences not just voting and policies
- Media outlets or technologies should not be considered independent from each other
- Future work must allow for secondary indirect effects of partisan media

Machine Learning Essentials

Ensemble Learning with XGBoost

Media Slant is Contagious

Deep Learning Essentials

- Neural networks $\leftrightarrow$ deep learning models
  - solve machine learning problems, just like logistic regression or gradient boosted machines
  - use torch, rather than sklearn.

- Neural networks $\leftrightarrow$ deep learning models
  - solve machine learning problems, just like logistic regression or gradient boosted machines
  - use torch, rather than sklearn.
- **why use neural nets?**
  - sometimes outperform standard ML techniques on standard problems
  - greatly outperform standard ML techniques on some problems, e.g. language modeling

- Neural networks $\leftrightarrow$ deep learning models
  - solve machine learning problems, just like logistic regression or gradient boosted machines
  - use torch, rather than sklearn.
- **why use neural nets?**
  - sometimes outperform standard ML techniques on standard problems
  - greatly outperform standard ML techniques on some problems, e.g. language modeling
- **why not use neural nets?**
  - usually worse than standard ML on standard problems, and harder to implement.
  - Computational constraints: they require specialized processors, can be expensive to train

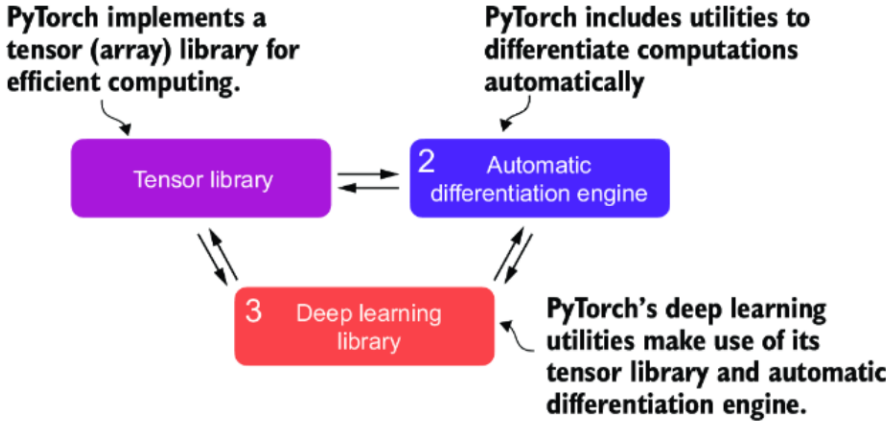
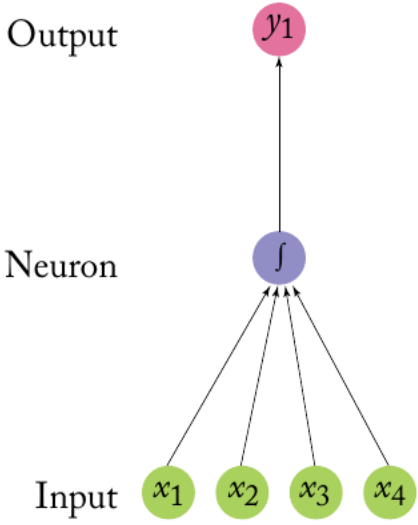


Figure A.1 PyTorch's three main components include a tensor library as a fundamental building block for computing, automatic differentiation for model optimization, and deep learning utility functions, making it easier to implement and train deep neural network models.

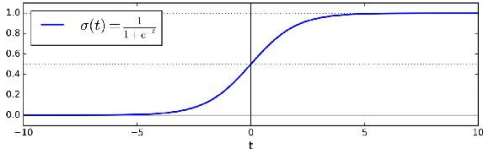
A "Neuron"



- applies dot product to vector of numerical inputs:
  - multiplies each input by a learned weight (parameter or coefficient)
  - sums these products
- applies a non-linear "activation function" to the sum
  - (e.g., the $\int$ shape indicates a sigmoid transformation)
- passes the output.

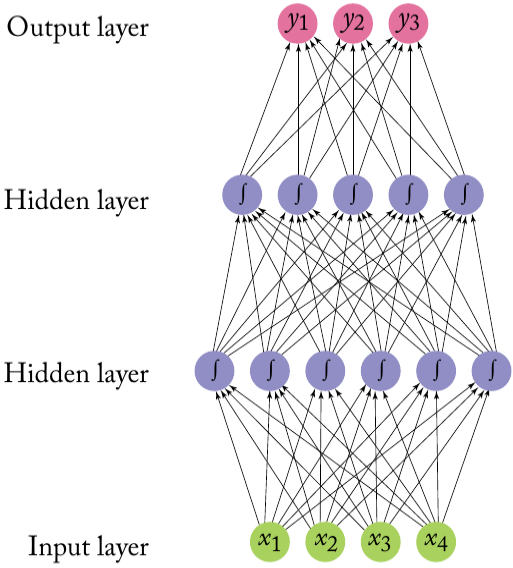
"Neuron" = Logistic Regression

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



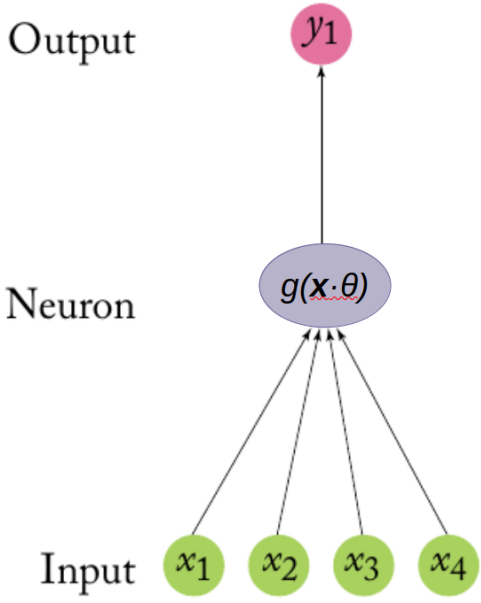
- applies dot product to vector of numerical inputs:
  - multiplies each input by a learned weight (parameter or coefficient)
  - sums these products
- applies a non-linear "activation function" (sigmoid) to the sum
- passes the output.

Multi-Layer Perceptron (MLP)



- A multilayer perceptron (also called a feed-forward network or sequential model) stacks neurons horizontally and vertically.
- alternatively, think of it as a stacked ensemble of logistic regression models.
- this vertical stacking is the "deep" in "deep learning"!

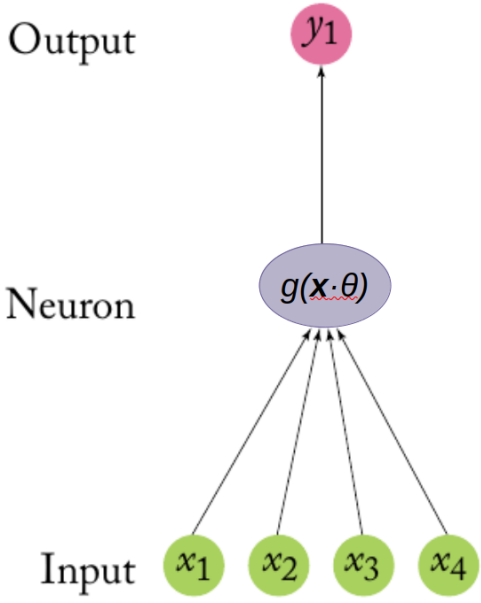
Activation functions $g(\boldsymbol{x} \cdot \theta)$



Previously we had

$$\mathrm{g}(\boldsymbol{x} \cdot \theta) = \mathrm{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$

Activation functions $g(\boldsymbol{x} \cdot \theta)$

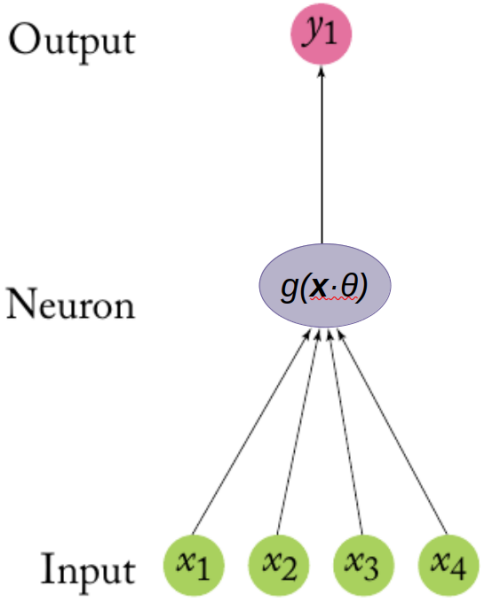


Previously we had

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

It turns out that sigmoid does not work well in hidden layers, mainly because gradient is flat except around zero.

Activation functions $g(\boldsymbol{x} \cdot \theta)$



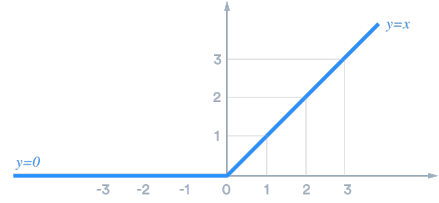
Previously we had

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

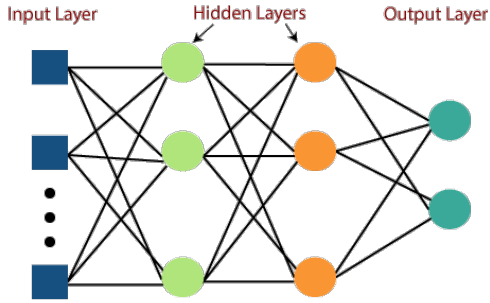
It turns out that sigmoid does not work well in hidden layers, mainly because gradient is flat except around zero.

ReLU (rectified linear unit) function:

$$g(\boldsymbol{x} \cdot \theta) = \text{ReLU}(\boldsymbol{x} \cdot \theta) = \max\{0, \boldsymbol{x} \cdot \theta\}$$



Equation Notation: Multi-Layer Perceptron



- An multi-layer perceptron (MLP) with two hidden layers is

$$\boldsymbol{y} = \boldsymbol{g}_2(\boldsymbol{g}_1(\boldsymbol{x} \cdot \boldsymbol{\omega}_1) \cdot \boldsymbol{\omega}_2) \cdot \boldsymbol{\omega}_y$$

$$\boldsymbol{y} \in \{0,1\}^{n_y}, \boldsymbol{x} \in \mathbb{R}^{n_x}, \boldsymbol{\omega}_1 \in \mathbb{R}^{n_x \times n_1}, \boldsymbol{\omega}_2 \in \mathbb{R}^{n_1 \times n_2}, \boldsymbol{\omega}_y \in \mathbb{R}^{n_2 \times n_y}$$

  - n_1, n_2 = dimensionality in first and second hidden layer.
  - $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_y$ = set of learnable weights for the first hidden, second hidden, and output layer
  - $\boldsymbol{g}_1(\cdot), \boldsymbol{g}_2(\cdot)$ = element-wise non-linear functions (e.g. ReLU) for first and second layer.

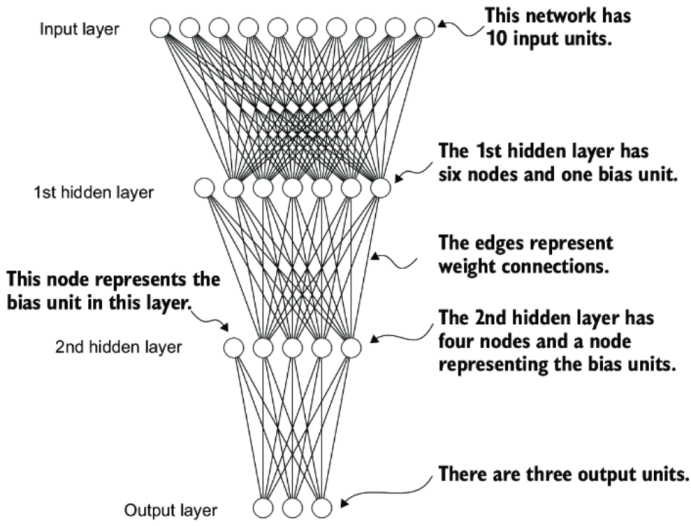


Figure A.9 A multilayer perceptron with two hidden layers. Each node represents a unit in the respective layer. For illustration purposes, each layer has a very small number of nodes.

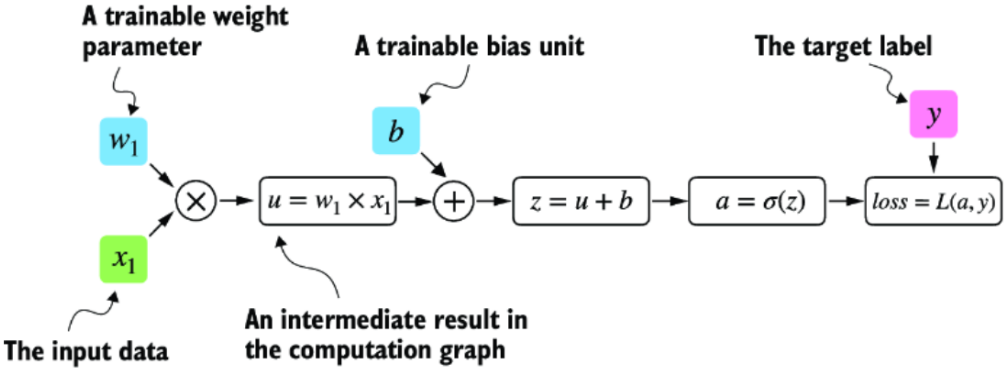


Figure A.7 A logistic regression forward pass as a computation graph. The input feature x_1 is multiplied by a model weight w_1 and passed through an activation function s after adding the bias. The loss is computed by comparing the model output a with a given label y.

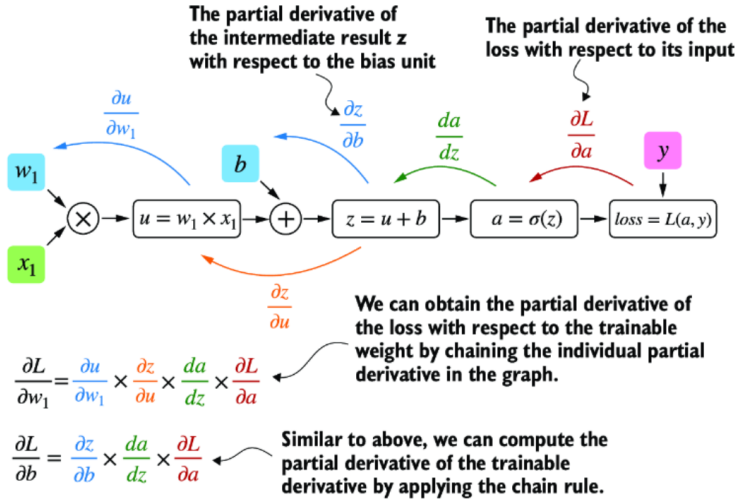


Figure A.8 The most common way of computing the loss gradients in a computation graph involves applying the chain rule from right to left, also called reverse-model automatic differentiation or backpropagation. We start from the output layer (or the loss itself) and work backward through the network to the input layer. We do this to compute the gradient of the loss with respect to each parameter (weights and biases) in the network, which informs how we update these parameters during training.

Listing A.9 Neural network training in PyTorch

```
import torch.nn.functional as F

torch.manual_seed(123)
model = NeuralNetwork(num_inputs=2, num_outputs=2)    #1
optimizer = torch.optim.SGD(
    model.parameters(), lr=0.5
)           #2

num_epochs = 3
for epoch in range(num_epochs):

    model.train()
    for batch_idx, (features, labels) in enumerate(train_loader):
        logits = model(features)

        loss = F.cross_entropy(logits, labels)

        optimizer.zero_grad()           #3
        loss.backward()         #4
        optimizer.step()        #5

        ### LOGGING
        print(f"Epoch: {epoch+1:03d}/{num_epochs:03d}"
              f" | Batch {batch_idx:03d}/{len(train_loader):03d}"
              f" | Train Loss: {loss:.2f}")

    model.eval()
    # Insert optional model evaluation code
```

#1 The dataset has two features and two classes.

#2 The optimizer needs to know which parameters to optimize.

#3 Sets the gradients from the previous round to 0 to prevent unintended gradient accumulation

#4 Computes the gradients of the loss given the model parameters

#5 The optimizer uses the gradients to update the model parameters.

Dropout

An elegant regularization technique:

- at every training step, every neuron has some probability (typically $p = 0.5$) of being temporarily dropped out, so that it will be ignored at this step.
- at test time, neurons dont get dropped anymore but coefficients are down-weighted by p.

Dropout

An elegant regularization technique:

- at every training step, every neuron has some probability (typically $p = 0.5$) of being temporarily dropped out, so that it will be ignored at this step.
- at test time, neurons dont get dropped anymore but coefficients are down-weighted by p.

Why it works:

- Neurons cannot co-adapt with neighbors; they must be independently useful.
- Layers cannot rely excessively on just a few inputs.
- Approximates an ensemble of N models (where N is the number of neurons).

Early Stopping

- A second elegant regularization technique, used in both xgboost and in neural nets:
  - gradually train the model gradients while checking fit in a held-out validation set.
  - when model starts overfitting, stop training.
- Requires user to specify two additional parameters:
  - validation set: a third sample of the data, separate from training set and test set
  - early stopping rounds: stop training if we have done this many epochs without improving validation set performance.