Natural Language Processing for Law and Social Science

4. Supervised Learning with Text

Response Essay RE01 Due Friday, March 24th

- ▶ Critically read and review an application paper.
- ▶ 300 words is the minimum for a passing grade (2+) but 500+ words would be expected for high grade (4-5).
- ▶ Anonymize your submission – do not include your name anywhere in the document. Submit as TXT or PDF to EduFlow.

What can I write about?

- Any "Applications" reading from RE01 list (see "Response Essays" page, linked from syllabus).
- Can read/respond to an off-syllabus paper if it applies tools from one of the first four lectures.
  - dictionary methods, text complexity, n-grams, document similarity, topic models, machine learning.
  - Please confirm off-syllabus readings with me by email.
  - No papers allowed from the in-class presentaiton list either
- Note:
  - if a paper has an appendix, you are responsible for reading it!

What to think about

See response essays page.

- Apply what we have been doing in lecture and in the in-class presentations.
- Focus on identifying problems with the paper, rather than summarizing it.
- Example essays from previous years are available from homework page.
- We will continue to practice criticizing the required reading next week.
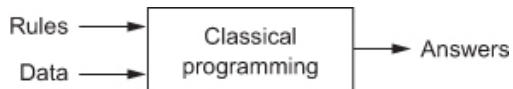
In-Class Presentation: Hoyle et al
"Unsupervised discovery of gendered language through latent-variable modeling"
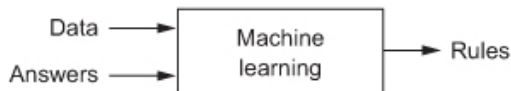
Machine Learning Essentials

Ensemble Learning with XGBoost
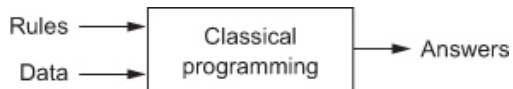
Deep Learning Essentials

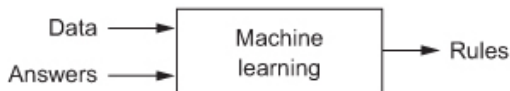# What is machine learning?

- ▶ 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...
```
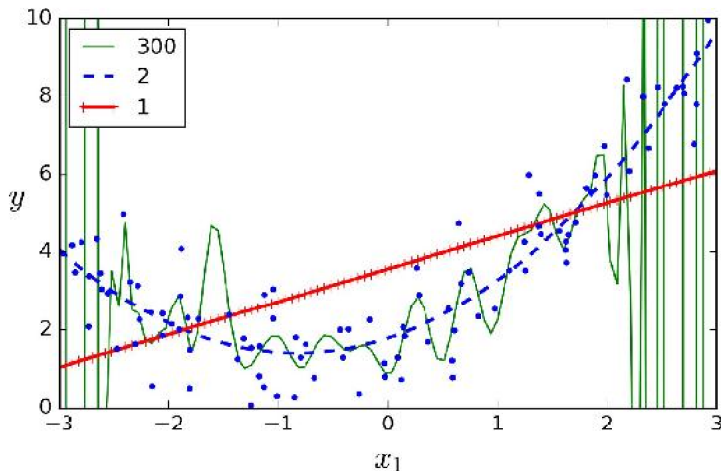
What do ML Algorithms do? Fit a function to data points

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

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

   ▶ 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

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

  ▶ $\rightarrow$ 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):

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

  ▶ $\eta = $ 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 $\mathbf{y}_i$ with dimensions $n_y \geq 1$

▶ Some documents are labeled and some are unlabeled $\rightarrow$
  ▶ we would like to learn a function $\hat{\mathbf{y}}(d_i)$ 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 ↔ 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:
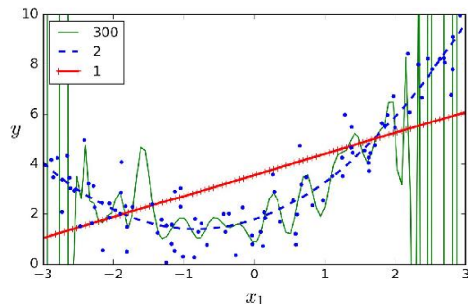 ▶ tends to over-fit training data.
  ▶ cannot handle multicollinearity.

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

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

▶ $R(\theta)$ 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 coeffcients 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

```python
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 ↔ 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:

$$L(\theta) = \underbrace{-\frac{1}{n_D}}_{\text{negative}} \sum_{i=1}^{n_D} [\underbrace{y_i}_{y_i=1} \underbrace{\log(\hat{y}_i)}_{\log \text{ prob} y_i=1} + \underbrace{(1-y_i)}_{y_i=0}\underbrace{\log(1-\hat{y}_i)}_{\log \text{ prob} y_i=0}]$$
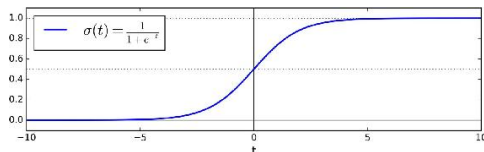
▶ In **logistic regression** we use a sigmoid transformation:
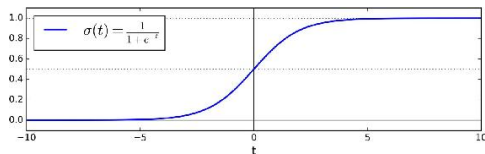
$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$

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

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

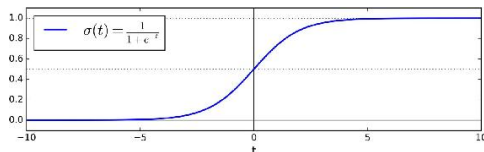▶ Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

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

  ▶ 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).

▶ 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:

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

▶ → 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:

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

▶ $\rightarrow$ 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:

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

▶ → 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

Calibration plot
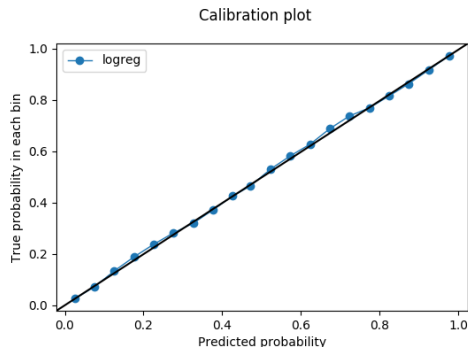
▶ Plotting the binned fraction in a category (Y axis) against the predicted probability in a category (X axis):

▶ 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 $D = 3.5$M 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 = 3.5M 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:
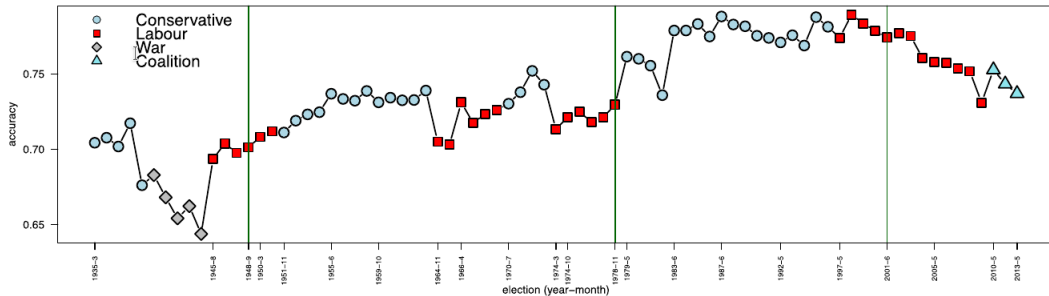
Multi-Class Classification

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

$$\hat{y}_k(\mathbf{x}_i) = \Pr(y_i = k) = \frac{\exp(\theta'_k \mathbf{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta'_l \mathbf{x}_i)}$$

▶ softmax is the multiclass generalization of sigmoid → can then interpret $\hat{y}$ as probabilities.

▶ n_x features and n_y output classes → 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

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

▶ 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

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

▶ 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

In-Class Presentation: Osnabruegge, Ash, and Morelli (2022)

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

	Dictionaries (Custom)	Dictionaries (Generic)	Topic Modeling	Within-Domain Supervised Learning	Cross-Domain Supervised Learning
Specificity	High	Moderate	Low	High	Moderate
Interpretability	High	High	Moderate	High	High

	Dictionaries (Custom)	Dictionaries (Generic)	Topic Modeling	Within-Domain Supervised Learning	Cross-Domain Supervised Learning
Validatability	Low	Low	Low	High	High

Machine Learning Essentials

Ensemble Learning with XGBoost

Deep Learning Essentials

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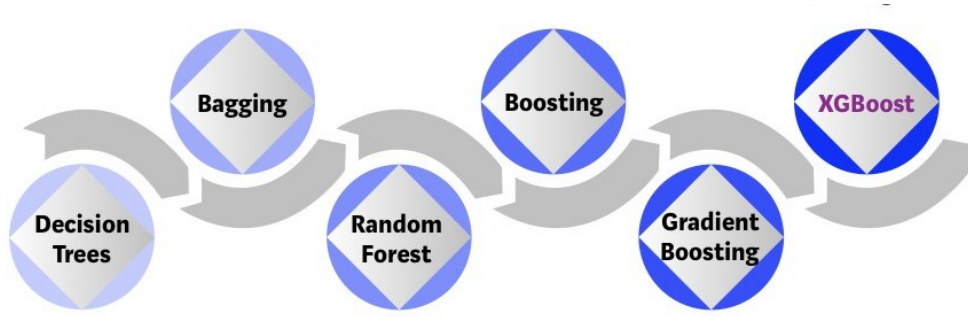
Voting Classifiers
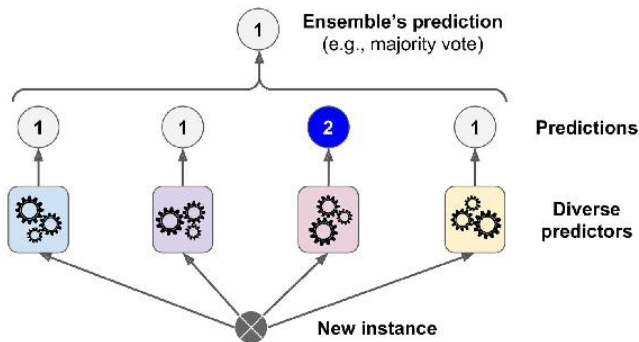
- ▶ voting classifiers (ensembles of different models that vote on the prediction) generally out-perform the best classifier in the ensemble.
  - ▶ more diverse algorithms will make different types of errors, and improve your ensemble's robustness.

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

```python
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)
```

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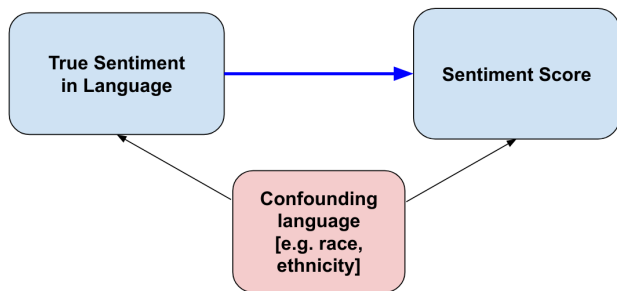
Review: Classifiers Learn Confounders
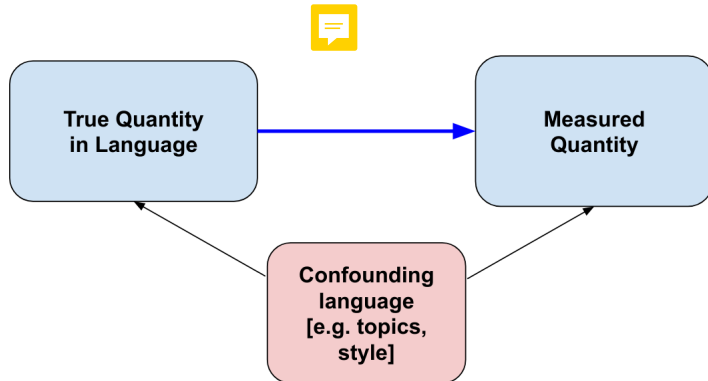
- supervised models (classifiers, regressors) learn features that are correlated with the label being annotated.
  - when taken to new data, they predicted label might reflect the presence of the confounders, rather than the presence of the true label.

In-Class Presentation:
Youyou, Wang, and Uzzi (2023)
A discipline-wide investigation of the replicability of Psychology papers over the past two decades

- ▶ Neural networks ↔ 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

A "Neuron"
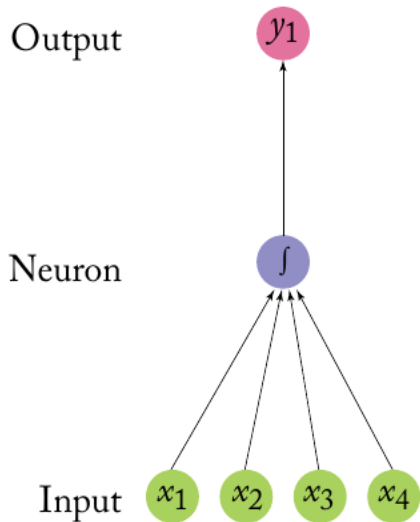
Output

Neuron

Input

- ▶ 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

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$
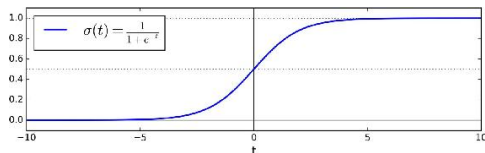
- ▶ applies dot product to vector of numerical inputs:
  - ▶ multiplies each input by a learned weight (parameter or coefficient)
  - ▶ sums these products
- ▶ applies a non-linear "activation function" (sigmoid) to the sum
- ▶ passes the output.

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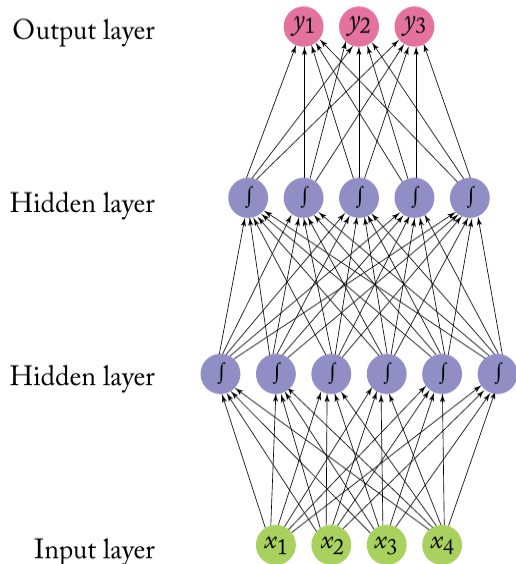
- ▶ A multilayer perceptron (also called a feed-forward network or sequential model) stacks neurons horizontally and vertically.
- ▶ alternatively, think of it as a stacked ensemble of logistic regression models.
- ▶ this vertical stacking is the "deep" in "deep learning"!

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

Output: y_1

Neuron: $g(\boldsymbol{x} \cdot \theta)$

Input: x_1 x_2 x_3 x_4
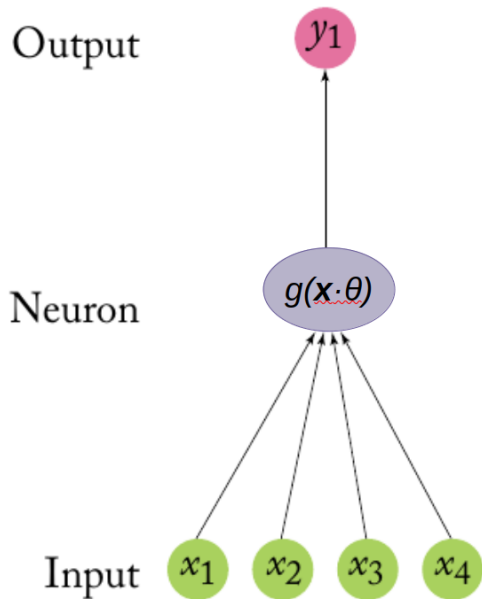
Previously we had

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

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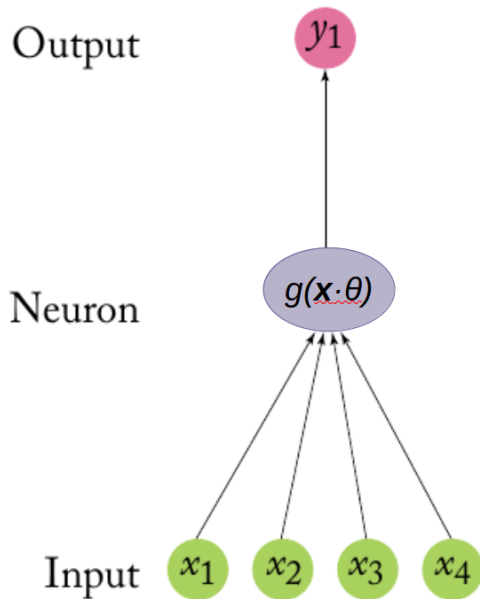
Output

Neuron

Input

Previously we had

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

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

Activation functions $g(\boldsymbol{x} \cdot \theta)$
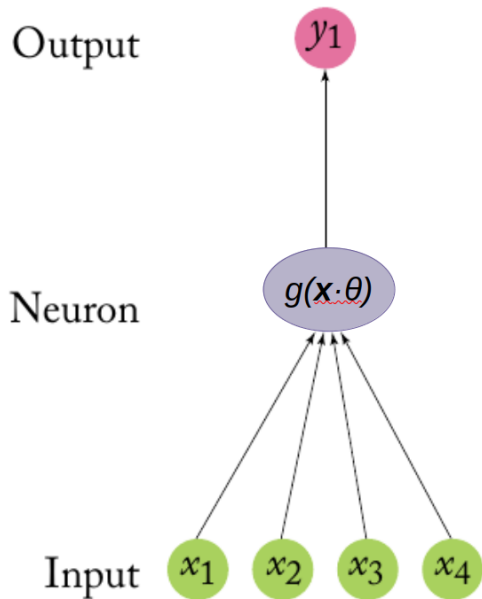
Output y_1

Neuron $g(\boldsymbol{x} \cdot \theta)$

Input x_1 x_2 x_3 x_4

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:

$$g(\boldsymbol{x} \cdot \theta) = \text{ReLU}(\boldsymbol{x} \cdot \theta) = \max\{0, \boldsymbol{x} \cdot \theta\}$$
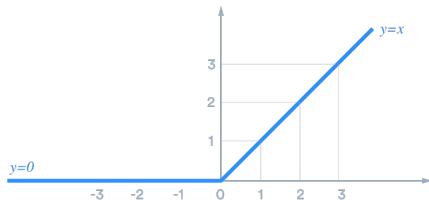
Equation Notation: Multi-Layer Perceptron
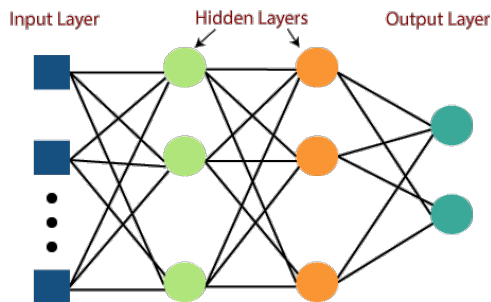
▶ An multi-layer perceptron (MLP) with two hidden layers is

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

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

  ▶ 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.

Google Developers Guide for Text Classification:

- with relatively few, longer documents, use an MLP:

    - $x =$ **tf-idf-weighted bigrams** as a baseline specification for text classification tasks.

        - select 20,000 features using supervised feature selection in training set.

    - $f(\cdot) =$ MLP with two hidden layers.

- A simple MLP is one of the models tried by the U.K. parliament paper (Peterson and Spirling 2018).
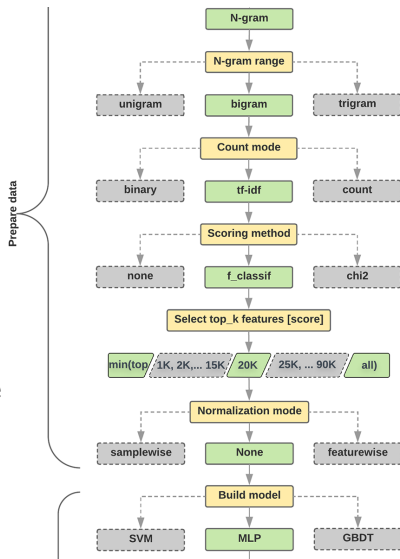
Dropout

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.

True/False Quiz

1. Good performance with stochastic gradient descent requires that the loss function is convex, so that is why it is used for training neural nets rather than regular (non-stochastic) gradient descent.

2. Feature scalers and feature selectors should be fitted in the training set and applied in the test set.

3. Mean squared error (MSE) is preferred to R-squared in evaluating/selecting regression models.

4. An advantage of xgboost over elastic net or neural nets is that it is less likely to learn from confounding text characteristics, and therefore generalize better out of domain.

5. Rectified linear unit (ReLU) should be used as the activation function in MLPs.

6. Neural nets tend to out-perform xgboost on text classification tasks.

7. Early stopping means splitting into three sets (train, validation, test), and training the model until performance starts decreasing in the validation set.

8. Number of hidden layers, and number of neurons per layer, are hyperparameters that can be learned by cross-validation in the training set.