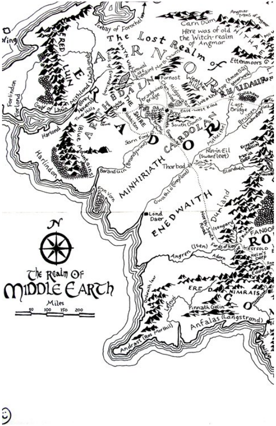
7A. Supervised Techniques IV

DS-GA 1015, Text as Data
Arthur Spirling

March 23, 2021

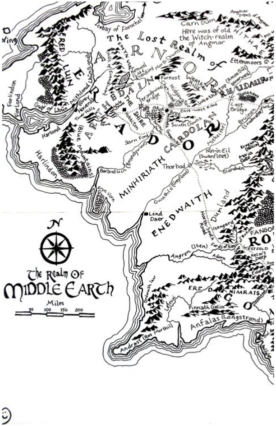
Where Are We?

Where Are We?



The Realm of
MIDDLE EARTH
Miles
50
100
150
200
The Lost Realm of
ARNOR
Here was of old
the Witch-realm
of Angmar
Carn Dûm
Angmar
Icebay of Forochel
ERED LUIN
Forlindon
Forlond
Gulf of Lhûn
Mithlond
(Grey Havens)
Lhûn (Lune)
Harlond
Harlindon
Tower Hills
(Blue Mountains)
ERIADOR
ARTHEDAIN
THE SHIRE
Hobbiton
Nenuial
(Lake Evendim)
North Downs
Fornost
Brandywine Bridge
Old Forest
Bree
Midgewater
Weather Hills
Weathertop
(Amon Sûl)
Ettenmoors
RHUDAUR
Trollshaws
Last Bridge
East-West Road
South Downs
Bruinen
(Loudwater)
Mitheithel
CARDOLAN
Sarn Ford
Baranduin (Brandywine)
MINHIRIATH
Tharbad
Nîn-in-Eilph
(Swanfleet)
Glanduin
Moria Gate
Eryn Vorn
Gwathló (Greyflood)
Lond Daer
ENEDWAITH
Dunland
North-South Road
Isengard
FANGORN
ROHAN
WESTFOLD
Fords of Isen
Angren (Isen)
Adorn
Helm's Deep
Edoras
ERED NIMRAIS
GONDOR
Pinnath Gelin
Anfalas (Langstrand)
Andrast (Ras Morthil)
Lefnui

Where Are We?

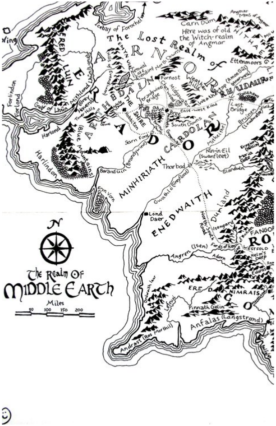


We've looked some supervised learning ideas for estimating the class of individual documents for e.g. the senate speech example:

Where Are We?

We've looked some supervised learning ideas for estimating the class of individual documents for e.g. the senate speech example: in particular, support vector machines.

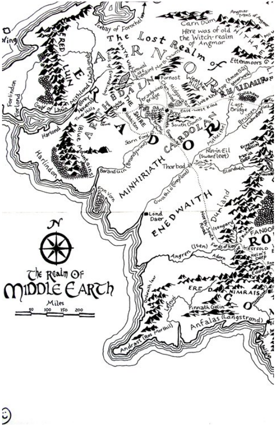
Where Are We?



We've looked some supervised learning ideas for estimating the class of individual documents for e.g. the senate speech example: in particular, support vector machines.

Continue with that today: KNN, CART etc.

Where Are We?



We've looked some supervised learning ideas for estimating the class of individual documents for e.g. the senate speech example: in particular, support vector machines.

Continue with that today: KNN, CART etc.

Plus ways to combine those techniques: ensembles.

Remember. . .

Remember...

Unsupervised techniques:

Remember...

Unsupervised techniques: learning (hidden or latent) structure in unlabeled data.

e.g. PCA of legislators's votes:

Remember...

Unsupervised techniques: learning (hidden or latent) structure in unlabeled data.

e.g. PCA of legislators's votes: want to see how they are organized—

Remember...

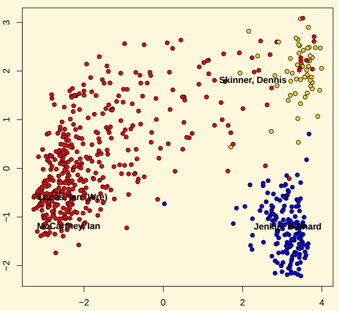
Unsupervised techniques: learning (hidden or latent) structure in unlabeled data.

e.g. PCA of legislators's votes: want to see how they are organized—by party? by ideology? by race?

Remember...

Unsupervised techniques: learning (hidden or latent) structure in unlabeled data.

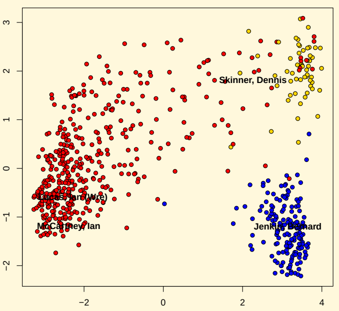
e.g. PCA of legislators's votes: want to see how they are organized—by party? by ideology? by race?



Remember...

Unsupervised techniques: learning (hidden or latent) structure in unlabeled data.

e.g. PCA of legislators's votes: want to see how they are organized—by party? by ideology? by race?

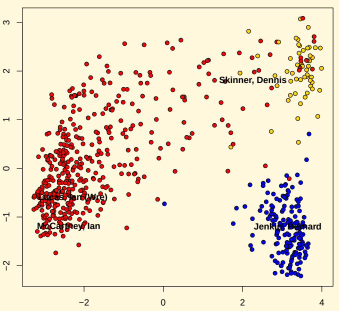


Supervised techniques:

Remember. . .

Unsupervised techniques: learning (hidden or latent) structure in unlabeled data.

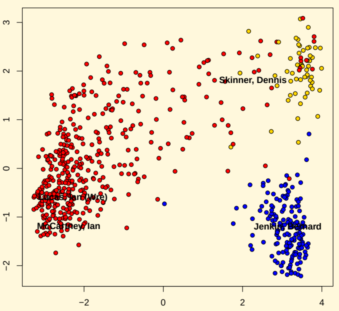
e.g. PCA of legislators's votes: want to see how they are organized—by party? by ideology? by race?



Supervised techniques: learning relationship between inputs and a labeled set of outputs.

Unsupervised techniques: learning (hidden or latent) structure in unlabeled data.

e.g. PCA of legislators's votes: want to see how they are organized—by party? by ideology? by race?



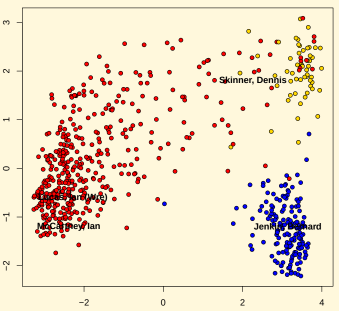
Supervised techniques: learning relationship between inputs and a labeled set of outputs.

e.g. opinion mining:

Remember...

Unsupervised techniques: learning (hidden or latent) structure in unlabeled data.

e.g. PCA of legislators's votes: want to see how they are organized—by party? by ideology? by race?



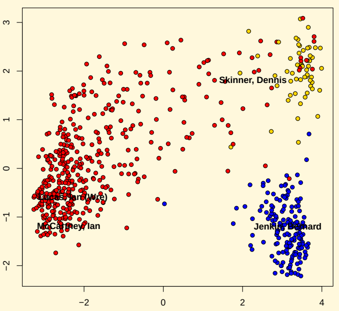
Supervised techniques: learning relationship between inputs and a labeled set of outputs.

e.g. opinion mining: what makes a critic like or dislike a movie ($y_i \in \{0, 1\}$)?

Remember. . .

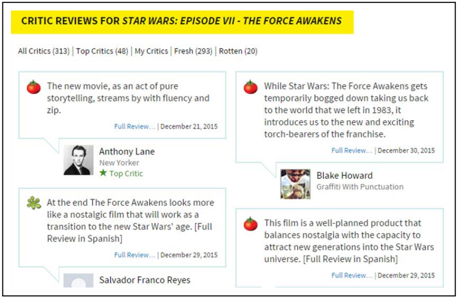
Unsupervised techniques: learning (hidden or latent) structure in unlabeled data.

e.g. PCA of legislators's votes: want to see how they are organized—by party? by ideology? by race?



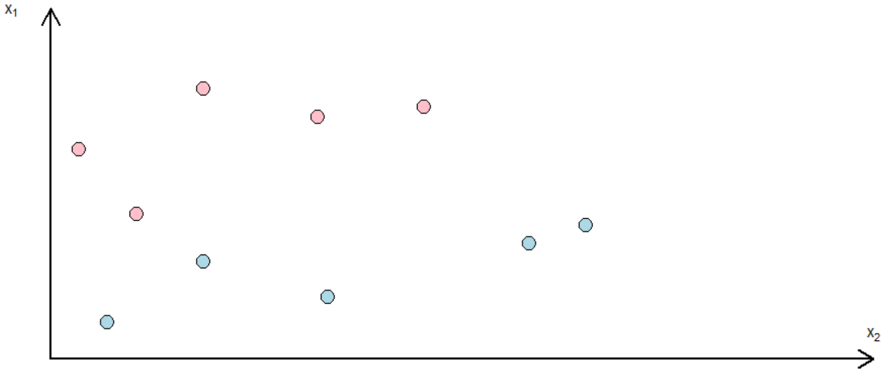
Supervised techniques: learning relationship between inputs and a labeled set of outputs.

e.g. opinion mining: what makes a critic like or dislike a movie ($y_i \in \{0, 1\}$)?

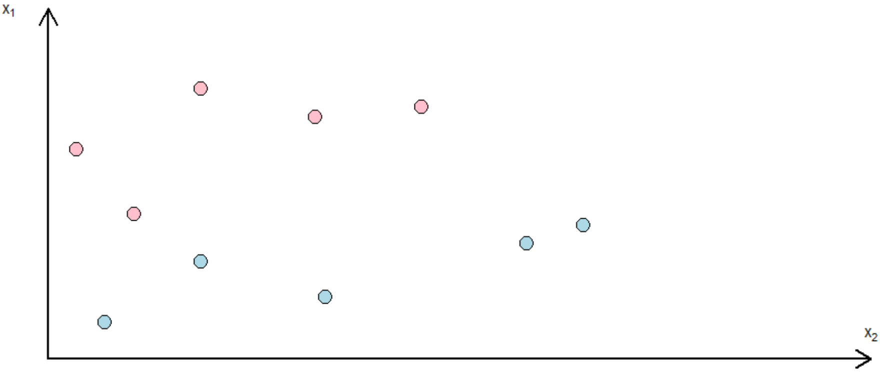


Reminder: The 10 Senators

Reminder: The 10 Senators



Reminder: The 10 Senators



k-nearest neighbors

k-nearest neighbors

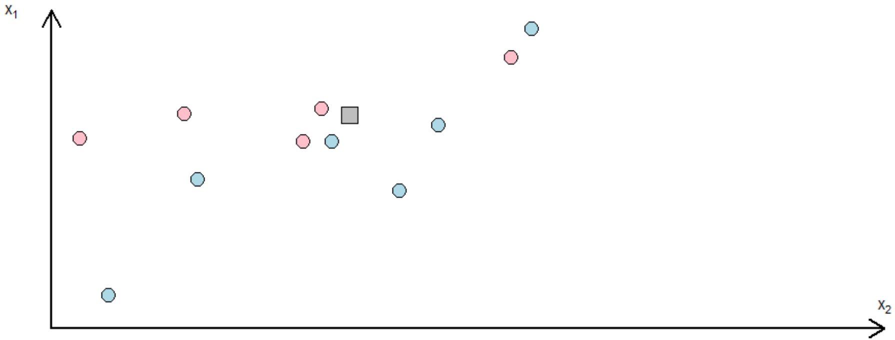
Variant of the Senate example:

k-nearest neighbors

Variant of the Senate example: now suppose we are interested in classifying a 'new' (test set) Senator (■) based on her feature values.

k-nearest neighbors

Variant of the Senate example: now suppose we are interested in classifying a 'new' (test set) Senator (■) based on her feature values.



x1
x2

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space,

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

e.g. $k = 3$:

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

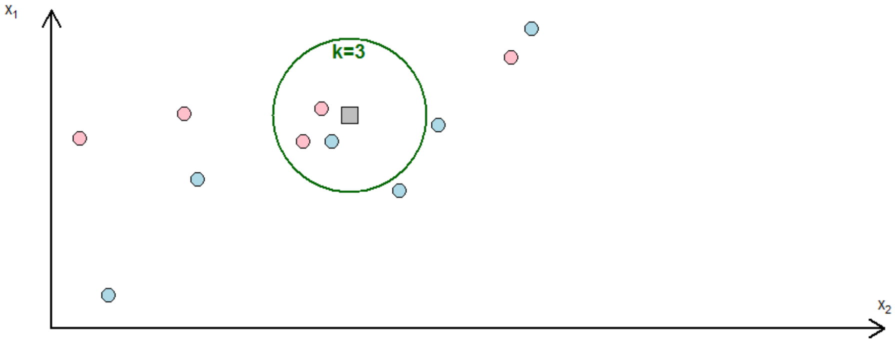
e.g. $k = 3$: she is assigned to the Republicans (2 vs 1)

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

e.g. $k = 3$: she is assigned to the Republicans (2 vs 1)



x_1
k=3
x_2

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a ‘new’ (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

e.g. $k = 3$: she is assigned to the Republicans (2 vs 1)

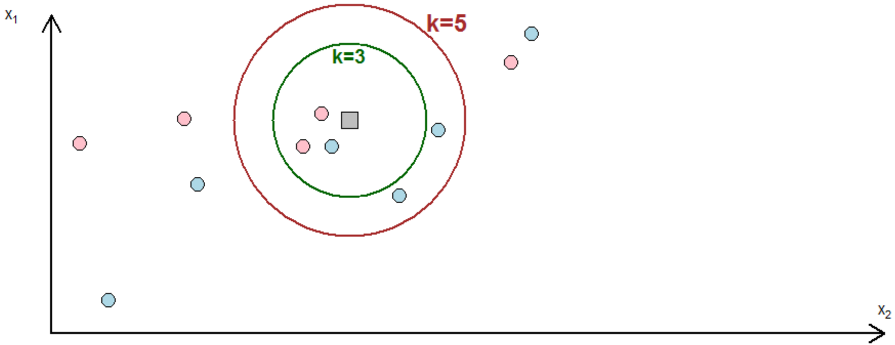
k-nearest neighbors

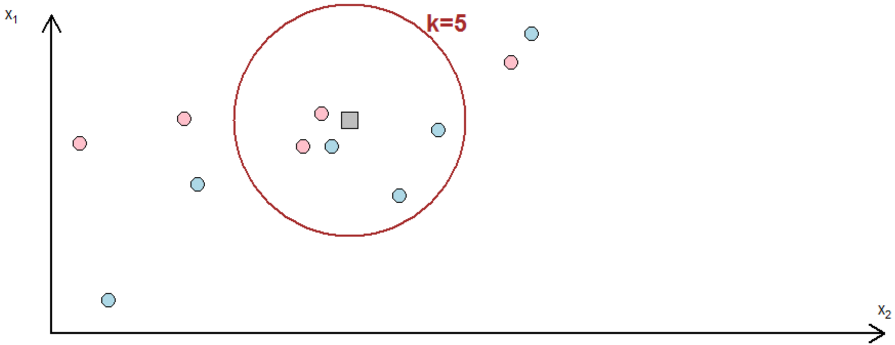
Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

e.g. $k = 3$: she is assigned to the Republicans (2 vs 1)

e.g. $k = 5$: she is assigned to the Democrats (3 vs 2)


x1
k=5
k=3
x2



x1
k=5
x2

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

e.g. $k = 3$: she is assigned to the Republicans (2 vs 1)

e.g. $k = 5$: she is assigned to the Democrats (3 vs 2)

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

e.g. $k = 3$: she is assigned to the Republicans (2 vs 1)

e.g. $k = 5$: she is assigned to the Democrats (3 vs 2)

→ Can use Euclidean distance (or some other metric for the neighbors), but need to be careful about imbalance in the training set.

k-nearest neighbors

Variant of the Senate example: now suppose we interested in classifying a 'new' (test set) Senator (■) based on her feature values.

One simple idea is to look at her k nearest neighbors from the training set in the feature space, and take a majority vote in terms of what party we should assign her to.

e.g. $k = 3$: she is assigned to the Republicans (2 vs 1)

e.g. $k = 5$: she is assigned to the Democrats (3 vs 2)

→ Can use Euclidean distance (or some other metric for the neighbors), but need to be careful about imbalance in the training set.

→ Choice of k can be optimized, but generally case that noise in data causes poor classification.

Advantages of *kNN*

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$.

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible.

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible. (But then, easy to overfit).

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible. (But then, easy to overfit).

Example of instance-based training ('lazy learning').

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible. (But then, easy to overfit).

Example of instance-based training ('lazy learning'). Everything uncovered about relationship between $\mathbb{E}(Y|X)$ is done locally.

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible. (But then, easy to overfit).

Example of instance-based training ('lazy learning'). Everything uncovered about relationship between $\mathbb{E}(Y|X)$ is done locally. So, no training and no model.

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible. (But then, easy to overfit).

Example of instance-based training ('lazy learning'). Everything uncovered about relationship between $\mathbb{E}(Y|X)$ is done locally. So, no training and no model. Can still inspect accuracy though.

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible. (But then, easy to overfit).

Example of instance-based training ('lazy learning'). Everything uncovered about relationship between $\mathbb{E}(Y|X)$ is done locally. So, no training and no model. Can still inspect accuracy though.

Works well so long as N is large,

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible. (But then, easy to overfit).

Example of instance-based training ('lazy learning'). Everything uncovered about relationship between $\mathbb{E}(Y|X)$ is done locally. So, no training and no model. Can still inspect accuracy though.

Works well so long as N is large, though this can be slow.

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible. (But then, easy to overfit).

Example of instance-based training ('lazy learning'). Everything uncovered about relationship between $\mathbb{E}(Y|X)$ is done locally. So, no training and no model. Can still inspect accuracy though.

Works well so long as N is large, though this can be slow.

Works with any types of features,

Advantages of *kNN*

It is non-parametric in sense that don't need any assumptions about functional form of $\mathbb{E}(Y|X)$. Don't need to divide the space in big 'chunks'—very flexible. (But then, easy to overfit).

Example of instance-based training ('lazy learning'). Everything uncovered about relationship between $\mathbb{E}(Y|X)$ is done locally. So, no training and no model. Can still inspect accuracy though.

Works well so long as N is large, though this can be slow.

Works with any types of features, though typically requires rescaling (normalizing) to ensure that one unit of one variable is not treated same as one unit of another (e.g. gender vs income: male is more different to female than \$10,000 is to \$10,001)

Disadvantages of *kNN*

Disadvantages of *kNN*

While interpretation of given decision is easy ("new observation is most like these other three")

Disadvantages of *kNN*

While interpretation of given decision is easy ("new observation is most like these other three") the interpretation of the model is hard. Don't typically have general lessons from *kNN*—there is no model.

Disadvantages of *kNN*

While interpretation of given decision is easy ("new observation is most like these other three") the interpretation of the model is hard. Don't typically have general lessons from *kNN*—there is no model.

Actually finding the nearest neighbours can be slow,

Disadvantages of *kNN*

While interpretation of given decision is easy ("new observation is most like these other three") the interpretation of the model is hard. Don't typically have general lessons from *kNN*—there is no model.

Actually finding the nearest neighbours can be slow, especially for long feature vectors and large N.

Disadvantages of *kNN*

While interpretation of given decision is easy ("new observation is most like these other three") the interpretation of the model is hard. Don't typically have general lessons from *kNN*—there is no model.

Actually finding the nearest neighbours can be slow, especially for long feature vectors and large N.

The curse of dimensionality.

Disadvantages of *kNN*

While interpretation of given decision is easy ("new observation is most like these other three") the interpretation of the model is hard. Don't typically have general lessons from *kNN*—there is no model.

Actually finding the nearest neighbours can be slow, especially for long feature vectors and large N.

The curse of dimensionality. As number of features grows large, points look more similar than they really are (especially if features are irrelevant to actual decision).

Disadvantages of *kNN*

While interpretation of given decision is easy ("new observation is most like these other three") the interpretation of the model is hard. Don't typically have general lessons from *kNN*—there is no model.

Actually finding the nearest neighbours can be slow, especially for long feature vectors and large N.

The curse of dimensionality. As number of features grows large, points look more similar than they really are (especially if features are irrelevant to actual decision). So, *kNN* often requires prior feature selection from theory,

Disadvantages of *kNN*

While interpretation of given decision is easy ("new observation is most like these other three") the interpretation of the model is hard. Don't typically have general lessons from *kNN*—there is no model.

Actually finding the nearest neighbours can be slow, especially for long feature vectors and large N.

The curse of dimensionality. As number of features grows large, points look more similar than they really are (especially if features are irrelevant to actual decision). So, *kNN* often requires prior feature selection from theory, or via automated methods.

Disadvantages of *kNN*

While interpretation of given decision is easy ("new observation is most like these other three") the interpretation of the model is hard. Don't typically have general lessons from *kNN*—there is no model.

Actually finding the nearest neighbours can be slow, especially for long feature vectors and large N.

The curse of dimensionality. As number of features grows large, points look more similar than they really are (especially if features are irrelevant to actual decision). So, *kNN* often requires prior feature selection from theory, or via automated methods. May also have to tune distance function manually (e.g weight up 'safety rating' for baby products)

Trees and Forests

Single-Tree Models

Single-Tree Models

- Basic idea: divide covariate space into B (non-overlapping) regions that are fairly homogenous with respect to Y. Then make a prediction (c_b) for all observations in region R_B.

Single-Tree Models

- Basic idea: divide covariate space into B (non-overlapping) regions that are fairly homogenous with respect to Y. Then make a prediction (c_b) for all observations in region R_B.

- a tree model for J covariates is a function

Single-Tree Models

- Basic idea: divide covariate space into B (non-overlapping) regions that are fairly homogenous with respect to Y. Then make a prediction (c_b) for all observations in region R_B.

- a tree model for J covariates is a function

$$f(X_i) = T(X_i; \Theta) \equiv \sum_{b=1}^{B} c_b I(X_i \in R_b)$$

Single-Tree Models

- Basic idea: divide covariate space into B (non-overlapping) regions that are fairly homogenous with respect to Y. Then make a prediction (c_b) for all observations in region R_B.

- a tree model for J covariates is a function

$$f(X_i) = T(X_i; \Theta) \equiv \sum_{b=1}^{B} c_b I(X_i \in R_b)$$

where Θ is the parameter vector, which contains tree depth (size), region definitions (how to split up the Xs), predicted values (c_b). And $I(\cdot)$ is the indicator function.

Single-Tree Models

- Basic idea: divide covariate space into B (non-overlapping) regions that are fairly homogenous with respect to Y. Then make a prediction (c_b) for all observations in region R_B.

- a tree model for J covariates is a function

$$f(X_i) = T(X_i; \Theta) \equiv \sum_{b=1}^{B} c_b I(X_i \in R_b)$$

where Θ is the parameter vector, which contains tree depth (size), region definitions (how to split up the Xs), predicted values (c_b). And $I(\cdot)$ is the indicator function. Thus, a given model takes a value of X_i and gives back a $\hat{Y}$ for that.

Choosing Θ

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i.

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible.

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible. A simple choice for $L(\cdot)$ is the sum of squared errors.

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible. A simple choice for $L(\cdot)$ is the sum of squared errors.

- This is non-trivial to do,

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

  where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible. A simple choice for $L(\cdot)$ is the sum of squared errors.

- This is non-trivial to do, so use heuristics to find optimal splits in practice.

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible. A simple choice for $L(\cdot)$ is the sum of squared errors.

- This is non-trivial to do, so use heuristics to find optimal splits in practice.
- But we rarely fit such trees as is:

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible. A simple choice for $L(\cdot)$ is the sum of squared errors.

- This is non-trivial to do, so use heuristics to find optimal splits in practice.
- But we rarely fit such trees as is: algorithm will fit a node to every observation.

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible. A simple choice for $L(\cdot)$ is the sum of squared errors.

- This is non-trivial to do, so use heuristics to find optimal splits in practice.
- But we rarely fit such trees as is: algorithm will fit a node to every observation. So, prune.

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible. A simple choice for $L(\cdot)$ is the sum of squared errors.

- This is non-trivial to do, so use heuristics to find optimal splits in practice.
- But we rarely fit such trees as is: algorithm will fit a node to every observation. So, prune.
- Very simple, and fast.

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

  $$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

  where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible. A simple choice for $L(\cdot)$ is the sum of squared errors.

- This is non-trivial to do, so use heuristics to find optimal splits in practice.
- But we rarely fit such trees as is: algorithm will fit a node to every observation. So, prune.
- Very simple, and fast. But sensitive to initial conditions,

Choosing Θ

- the optimal choice of Θ will correctly capture relationship between X and Y, but avoids overfitting. Specifically:

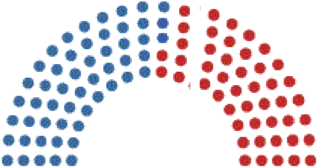
$$\hat{\Theta} = \arg\min_{\Theta} \sum_{b=1}^{B} \sum_{X_i \in R_b} L(y_i, c_b)$$

where $L(\cdot)$ is a loss function that tells us how far our predictions deviate from the true value of y_i. I want to find a Θ that makes the loss as small as possible. A simple choice for $L(\cdot)$ is the sum of squared errors.

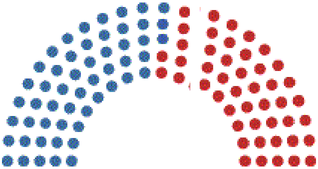
- This is non-trivial to do, so use heuristics to find optimal splits in practice.
- But we rarely fit such trees as is: algorithm will fit a node to every observation. So, prune.
- Very simple, and fast. But sensitive to initial conditions, and can't assess uncertainty.

Partitioning the Senators With Trees

Partitioning the Senators With Trees

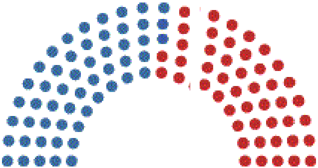


Partitioning the Senators With Trees



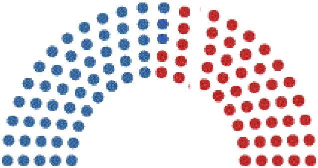
Idea our Senators are defined by their attributes.

Partitioning the Senators With Trees



Idea our Senators are defined by their attributes. Suppose we (optimally) split ('partition') the Senators with respect to x_1, such that we form two subsets of our training data.

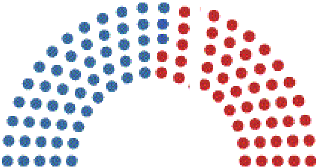
Partitioning the Senators With Trees



Idea our Senators are defined by their attributes. Suppose we (optimally) split ('partition') the Senators with respect to x_1, such that we form two subsets of our training data.

e.g suppose that Republicans generally use 'guns' more than Democrats,

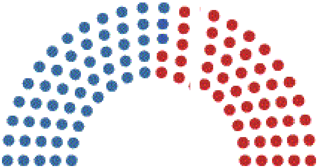
Partitioning the Senators With Trees



Idea our Senators are defined by their attributes. Suppose we (optimally) split ('partition') the Senators with respect to x_1, such that we form two subsets of our training data.

e.g suppose that Republicans generally use 'guns' more than Democrats, such that grabbing all the observations for which $x_{guns} > 0.621$ captures, say, 80% of the Republicans in our data.

Partitioning the Senators With Trees

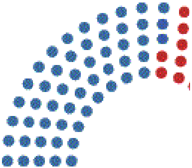


Idea our Senators are defined by their attributes. Suppose we (optimally) split ('partition') the Senators with respect to x_1, such that we form two subsets of our training data.

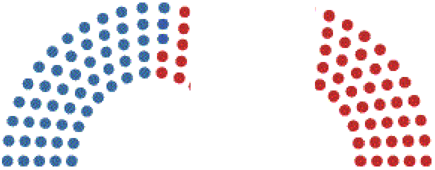
e.g suppose that Republicans generally use 'guns' more than Democrats, such that grabbing all the observations for which $x_{guns} > 0.621$ captures, say, 80% of the Republicans in our data.

Tree, stage 1

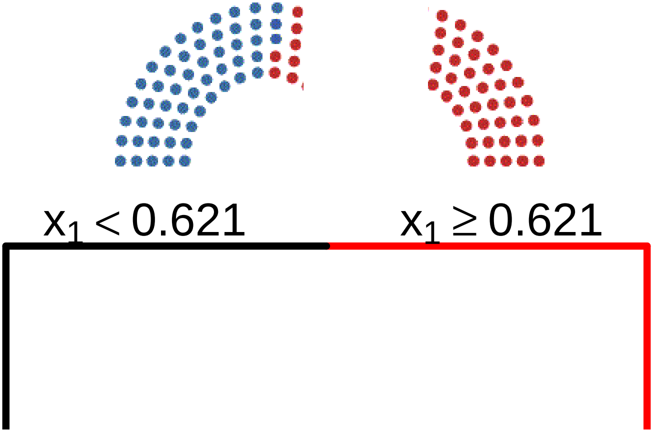
Tree, stage 1



Tree, stage 1



Tree, stage 1



Now, x_2 ...

Now, x_2 ...

$\rightarrow$ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

Now, x_2 ...

→ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

Now, x_2 . . .

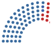
$\rightarrow$ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

and a subset that's still a mix of Republicans and Democrats.

Now, x_2 . . .

$\rightarrow$ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

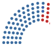
and a subset that's still a mix of Republicans and Democrats.



Now, x_2 ...

$\rightarrow$ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

and a subset that's still a mix of Republicans and Democrats.

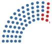


btw The set of Senators we've assigned to Republicans based on their x_1 values are called a leaf.

Now, x_2 ...

$\rightarrow$ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

and a subset that's still a mix of Republicans and Democrats.



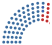
btw The set of Senators we've assigned to Republicans based on their x_1 values are called a leaf.

now suppose we take the mixed group remaining ('internal node') and split them based on x_2, which is their use of '`equality`'.

Now, x_2 ...

→ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

and a subset that's still a mix of Republicans and Democrats.



btw The set of Senators we've assigned to Republicans based on their x_1 values are called a leaf.

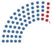
now suppose we take the mixed group remaining ('internal node') and split them based on x_2, which is their use of '`equality`'.

and it turns out that the (remaining) Republicans tend to use this less.

Now, x_2 . . .

→ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

and a subset that's still a mix of Republicans and Democrats.



btw The set of Senators we've assigned to Republicans based on their x_1 values are called a leaf.

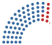
now suppose we take the mixed group remaining ('internal node') and split them based on x_2, which is their use of '`equality`'.

and it turns out that the (remaining) Republicans tend to use this less. So,

Now, x_2 ...

$\rightarrow$ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

and a subset that's still a mix of Republicans and Democrats.



btw The set of Senators we've assigned to Republicans based on their x_1 values are called a leaf.

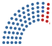
now suppose we take the mixed group remaining ('internal node') and split them based on x_2, which is their use of '`equality`'.

and it turns out that the (remaining) Republicans tend to use this less. So, when we partition according to, say, $x_2 \leq 0.56$ this enables us to perfectly divide this remaining subset into Democrats and Republicans.

Now, x_2 ...

$\rightarrow$ we now have two subsets of our training set: a bunch of Republicans (classified correctly based on x_1 alone)

and a subset that's still a mix of Republicans and Democrats.



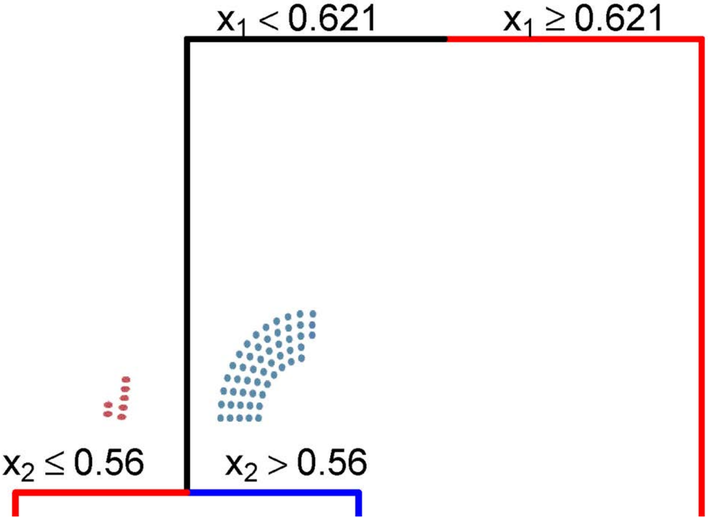
btw The set of Senators we've assigned to Republicans based on their x_1 values are called a leaf.

now suppose we take the mixed group remaining ('internal node') and split them based on x_2, which is their use of '`equality`'.

and it turns out that the (remaining) Republicans tend to use this less. So, when we partition according to, say, $x_2 \leq 0.56$ this enables us to perfectly divide this remaining subset into Democrats and Republicans.

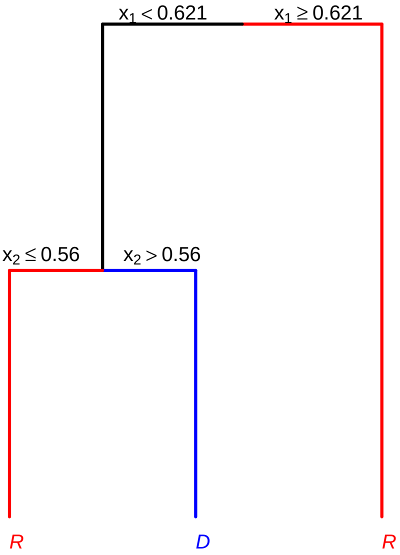
Tree, stage 2

Tree, stage 2



Complete Tree

Complete Tree



Notes

This classifier is known as a tree,

Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add 'much' to our classification ability.

Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add 'much' to our classification ability.

→ typically not the case that we can (or want to) classify perfectly into the leaves!

Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add ‘much’ to our classification ability.

$\rightarrow$ typically not the case that we can (or want to) classify perfectly into the leaves!

At each node,

Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add ‘much’ to our classification ability.

$\rightarrow$ typically not the case that we can (or want to) classify perfectly into the leaves!

At each node, algorithmic tricks allow fast searching over all the variables to find the one that should be used.

Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add 'much' to our classification ability.

$\rightarrow$ typically not the case that we can (or want to) classify perfectly into the leaves!

At each node, algorithmic tricks allow fast searching over all the variables to find the one that should be used.

and clearly need a metric for 'best' split in given x:

Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add 'much' to our classification ability.

→ typically not the case that we can (or want to) classify perfectly into the leaves!

At each node, algorithmic tricks allow fast searching over all the variables to find the one that should be used.

and clearly need a metric for 'best' split in given x: typically based on how homogenous the resulting subset of the data is

Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add 'much' to our classification ability.

$\rightarrow$ typically not the case that we can (or want to) classify perfectly into the leaves!

At each node, algorithmic tricks allow fast searching over all the variables to find the one that should be used.

and clearly need a metric for 'best' split in given x: typically based on how homogenous the resulting subset of the data is

e.g. 'Gini impurity' and 'Variance Reduction'

Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add 'much' to our classification ability.

→ typically not the case that we can (or want to) classify perfectly into the leaves!

At each node, algorithmic tricks allow fast searching over all the variables to find the one that should be used.

and clearly need a metric for 'best' split in given x: typically based on how homogenous the resulting subset of the data is

e.g. 'Gini impurity' and 'Variance Reduction'

+ Trees are easy to interpret,

Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add 'much' to our classification ability.

→ typically not the case that we can (or want to) classify perfectly into the leaves!

At each node, algorithmic tricks allow fast searching over all the variables to find the one that should be used.

and clearly need a metric for 'best' split in given x: typically based on how homogenous the resulting subset of the data is

e.g. 'Gini impurity' and 'Variance Reduction'

\+ Trees are easy to interpret, and we can report relative variable importance statistics.

Notes

This classifier is known as a tree, and the recursive partitioning on each derived subset continues until further splits doesn't add 'much' to our classification ability.

$\rightarrow$ typically not the case that we can (or want to) classify perfectly into the leaves!

At each node, algorithmic tricks allow fast searching over all the variables to find the one that should be used.

and clearly need a metric for 'best' split in given x: typically based on how homogenous the resulting subset of the data is

e.g. 'Gini impurity' and 'Variance Reduction'

$+$ Trees are easy to interpret, and we can report relative variable importance statistics.

But. . .

But. . .

These basic CART (Classification and Regression Trees) approaches show instability in practice:

But...

These basic CART (Classification and Regression Trees) approaches show instability in practice:

i.e. minor changes to training data can have large consequences for classification decisions,

But. . .

These basic CART (Classification and Regression Trees) approaches show instability in practice:

i.e. minor changes to training data can have large consequences for classification decisions, because any error is propagated down the tree.

But...

These basic CART (Classification and Regression Trees) approaches show instability in practice:

i.e. minor changes to training data can have large consequences for classification decisions, because any error is propagated down the tree.

$\rightarrow$ related to problems of overfitting in the training data.

But. . .

These basic CART (Classification and Regression Trees) approaches show instability in practice:

i.e. minor changes to training data can have large consequences for classification decisions, because any error is propagated down the tree.

$\rightarrow$ related to problems of overfitting in the training data.

So effort is made to prune the trees back

But. . .

These basic CART (Classification and Regression Trees) approaches show instability in practice:

i.e. minor changes to training data can have large consequences for classification decisions, because any error is propagated down the tree.

→ related to problems of overfitting in the training data.

So effort is made to prune the trees back—remove less helpful branches.

But. . .

These basic CART (Classification and Regression Trees) approaches show instability in practice:

i.e. minor changes to training data can have large consequences for classification decisions, because any error is propagated down the tree.

$\rightarrow$ related to problems of overfitting in the training data.

So effort is made to prune the trees back—remove less helpful branches.

Or can construct many trees (from slightly different samples of the data) and average over them:

But...

These basic CART (Classification and Regression Trees) approaches show instability in practice:

i.e. minor changes to training data can have large consequences for classification decisions, because any error is propagated down the tree.

$\rightarrow$ related to problems of overfitting in the training data.

So effort is made to prune the trees back—remove less helpful branches.

Or can construct many trees (from slightly different samples of the data) and average over them: known as bagging ('bootstrap aggregating').

But. . .

These basic CART (Classification and Regression Trees) approaches show instability in practice:

i.e. minor changes to training data can have large consequences for classification decisions, because any error is propagated down the tree.

$\rightarrow$ related to problems of overfitting in the training data.

So effort is made to prune the trees back—remove less helpful branches.

Or can construct many trees (from slightly different samples of the data) and average over them: known as bagging ('bootstrap aggregating').

$\rightarrow$ random forests combines *many* trees (forest) and

But. . .

These basic CART (Classification and Regression Trees) approaches show instability in practice:

i.e. minor changes to training data can have large consequences for classification decisions, because any error is propagated down the tree.

$\rightarrow$ related to problems of overfitting in the training data.

So effort is made to prune the trees back—remove less helpful branches.

Or can construct many trees (from slightly different samples of the data) and average over them: known as bagging ('bootstrap aggregating').

$\rightarrow$ random forests combines *many* trees (forest) and at each split a *random sample* of features is considered (rather than all features).

Tree Ensemble Approaches

Tree Ensemble Approaches

- We can reduce overfitting by fitting multiple trees. In particular, suppose there are M trees and Θ_m is the set of parameters for each tree, T_m:

Tree Ensemble Approaches

- We can reduce overfitting by fitting multiple trees. In particular, suppose there are M trees and Θ_m is the set of parameters for each tree, T_m:

$$f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$$

1. Bagging and Random Forests: fit trees *independently* to different subsets of data.

Tree Ensemble Approaches

- We can reduce overfitting by fitting multiple trees. In particular, suppose there are M trees and Θ_m is the set of parameters for each tree, T_m:

$$f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$$

1. Bagging and Random Forests: fit trees *independently* to different subsets of data.

2. Boosting and Multiple Additive Regression Trees: fit trees *sequentially* to *transformations* of the data.

Tree Ensemble Approaches

- We can reduce overfitting by fitting multiple trees. In particular, suppose there are M trees and Θ_m is the set of parameters for each tree, T_m:

$$f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$$

1 Bagging and Random Forests: fit trees *independently* to different subsets of data.

2 Boosting and Multiple Additive Regression Trees: fit trees *sequentially* to *transformations* of the data.

3 Bayesian Trees: similar to boosting, except trees constructed in hierarchical Bayesian framework.

Tree Bagging and Random Forests

Tree Bagging and Random Forests

- Bagging—bootstrap aggregating—rests on observation that trees fit to different subsets of data (observations) will give different predictions.

Tree Bagging and Random Forests

- Bagging—bootstrap aggregating—rests on observation that trees fit to different subsets of data (observations) will give different predictions.

- If those trees are independent, then we get a low bias, low variance estimate of the true response.

Tree Bagging and Random Forests

- Bagging—bootstrap aggregating—rests on observation that trees fit to different subsets of data (observations) will give different predictions.

- If those trees are independent, then we get a low bias, low variance estimate of the true response. So, grow the trees (fully) and just take an average over the M samples:

$$\widehat{f_{\text{bag}}}(X_i) = \frac{1}{M} \sum_{m=1}^{M} T_m(X_i; \hat{\Theta}_m)$$

Tree Bagging and Random Forests

- Bagging—bootstrap aggregating—rests on observation that trees fit to different subsets of data (observations) will give different predictions.

- If those trees are independent, then we get a low bias, low variance estimate of the true response. So, grow the trees (fully) and just take an average over the M samples:

$$\widehat{f_{\text{bag}}}(X_i) = \frac{1}{M} \sum_{m=1}^{M} T_m(X_i; \hat{\Theta}_m)$$

- In practice, this can result in quite correlated trees,

Tree Bagging and Random Forests

- Bagging—bootstrap aggregating—rests on observation that trees fit to different subsets of data (observations) will give different predictions.

- If those trees are independent, then we get a low bias, low variance estimate of the true response. So, grow the trees (fully) and just take an average over the M samples:

$$\widehat{f_{\text{bag}}}(X_i) = \frac{1}{M} \sum_{m=1}^{M} T_m(X_i; \hat{\Theta}_m)$$

- In practice, this can result in quite correlated trees, so random forests does splits of random subset of variables at each node in a tree.

Boosting

Boosting

- Build trees sequentially,

Boosting

- Build trees sequentially, with focus on observations that current ensemble (so far) struggles with.

Boosting

- Build trees sequentially, with focus on observations that current ensemble (so far) struggles with.

- so, solve

$$\widehat{\Theta_m} = \arg\min_{\Theta_m} \sum_{i=1}^{M} L(y_i,\ f_{m-1}(X_i) + T_m(X_i; \Theta_m))$$

where $f_{m-1}(X_i)$ is the value of the sum of all the trees we've fit up to now, and $T_m(X_i; \Theta_m)$ is the tree we are fitting in the mth stage.

Boosting

- Build trees sequentially, with focus on observations that current ensemble (so far) struggles with.
- so, solve

$$\widehat{\Theta_m} = \arg\min_{\Theta_m} \sum_{i=1}^{M} L(y_i,\ f_{m-1}(X_i) + T_m(X_i; \Theta_m))$$

where $f_{m-1}(X_i)$ is the value of the sum of all the trees we've fit up to now, and $T_m(X_i; \Theta_m)$ is the tree we are fitting in the mth stage.

So this is fitting a tree to the observations (the y_i) where the current value of the loss is greatest

Boosting

- Build trees sequentially, with focus on observations that current ensemble (so far) struggles with.

- so, solve

$$\widehat{\Theta_m} = \arg\min_{\Theta_m} \sum_{i=1}^{M} L(y_i,\ f_{m-1}(X_i) + T_m(X_i; \Theta_m))$$

where $f_{m-1}(X_i)$ is the value of the sum of all the trees we've fit up to now, and $T_m(X_i; \Theta_m)$ is the tree we are fitting in the mth stage.

So this is fitting a tree to the observations (the y_i) where the current value of the loss is greatest—i.e. where the model currently fits worse.

Boosting

- Build trees sequentially, with focus on observations that current ensemble (so far) struggles with.

- so, solve

$$\widehat{\Theta_m} = \arg\min_{\Theta_m} \sum_{i=1}^{M} L(y_i,\ f_{m-1}(X_i) + T_m(X_i; \Theta_m))$$

  where $f_{m-1}(X_i)$ is the value of the sum of all the trees we've fit up to now, and $T_m(X_i; \Theta_m)$ is the tree we are fitting in the mth stage.

So this is fitting a tree to the observations (the y_i) where the current value of the loss is greatest—i.e. where the model currently fits worse.

Boosting, in practice

Boosting, in practice

- logic is simple. . . but difficult in practice.

Boosting, in practice

- logic is simple. . . but difficult in practice. So we approximate the minimization with gradient boosting—known also as multiple additive regression trees (MART),

Boosting, in practice

- logic is simple. . . but difficult in practice. So we approximate the minimization with gradient boosting—known also as multiple additive regression trees (MART), or gradient boosted machines (GBM).

Boosting, in practice

- logic is simple. . . but difficult in practice. So we approximate the minimization with gradient boosting—known also as multiple additive regression trees (MART), or gradient boosted machines (GBM).

- Essence is:

Boosting, in practice

- logic is simple... but difficult in practice. So we approximate the minimization with gradient boosting—known also as multiple additive regression trees (MART), or gradient boosted machines (GBM).
- Essence is: fit new tree to negative gradient of the loss function.

Boosting, in practice

- logic is simple... but difficult in practice. So we approximate the minimization with gradient boosting—known also as multiple additive regression trees (MART), or gradient boosted machines (GBM).

- Essence is: fit new tree to negative gradient of the loss function. Here gradient is derivative of loss wrt $f_{m-1}(X_i)$, $\frac{\partial L(\cdot)}{\partial f_{m-1}(\cdot)}$

Boosting, in practice

- logic is simple... but difficult in practice. So we approximate the minimization with gradient boosting—known also as multiple additive regression trees (MART), or gradient boosted machines (GBM).
- Essence is: fit new tree to negative gradient of the loss function. Here gradient is derivative of loss wrt $f_{m-1}(X_i)$, $\frac{\partial L(\cdot)}{\partial f_{m-1}(\cdot)}$
  Where that is largest, is the place (observations) where loss is increasing most.

Boosting, in practice

- logic is simple. . . but difficult in practice. So we approximate the minimization with gradient boosting—known also as multiple additive regression trees (MART), or gradient boosted machines (GBM).

- Essence is: fit new tree to negative gradient of the loss function. Here gradient is derivative of loss wrt $f_{m-1}(X_i)$, $\frac{\partial L(\cdot)}{\partial f_{m-1}(\cdot)}$

  Where that is largest, is the place (observations) where loss is increasing most. So, negative of that is what we want.

Boosting, in practice

- logic is simple. . . but difficult in practice. So we approximate the minimization with gradient boosting—known also as multiple additive regression trees (MART), or gradient boosted machines (GBM).

- Essence is: fit new tree to negative gradient of the loss function. Here gradient is derivative of loss wrt $f_{m-1}(X_i)$, $\frac{\partial L(\cdot)}{\partial f_{m-1}(\cdot)}$
  Where that is largest, is the place (observations) where loss is increasing most. So, negative of that is what we want.

- In practice, there are parameters to control depth of trees, and overall learning rate.

Bayesian (Additive Regression) Trees (BART)

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as $f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as $f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$
- Suppose we wrote this, instead:

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as $f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$
- Suppose we wrote this, instead:

$$y_i = \sum_{m=1}^{M} T_m(X_i; \Theta_m) + \epsilon_i,$$

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as $f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$
- Suppose we wrote this, instead:

$$y_i = \sum_{m=1}^{M} T_m(X_i; \Theta_m) + \epsilon_i,$$

where $\epsilon_i \sim N(0, \sigma^2)$ and y_i is a continuous outcome.

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as $f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$
- Suppose we wrote this, instead:

$$y_i = \sum_{m=1}^{M} T_m(X_i; \Theta_m) + \epsilon_i,$$

  where $\epsilon_i \sim N(0, \sigma^2)$ and y_i is a continuous outcome.

- Approach is "Bayesian" insofar as we put (independent) priors on the parameters on Θ: depth of tree,

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as $f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$
- Suppose we wrote this, instead:

$$y_i = \sum_{m=1}^{M} T_m(X_i; \Theta_m) + \epsilon_i,$$

where $\epsilon_i \sim N(0, \sigma^2)$ and y_i is a continuous outcome.

- Approach is "Bayesian" insofar as we put (independent) priors on the parameters on Θ: depth of tree, variables used at the splits,

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as $f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$
- Suppose we wrote this, instead:

$$y_i = \sum_{m=1}^{M} T_m(X_i; \Theta_m) + \epsilon_i,$$

where $\epsilon_i \sim N(0, \sigma^2)$ and y_i is a continuous outcome.

- Approach is "Bayesian" insofar as we put (independent) priors on the parameters on Θ: depth of tree, variables used at the splits, terminal-node values,

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as $f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$
- Suppose we wrote this, instead:

$$y_i = \sum_{m=1}^{M} T_m(X_i; \Theta_m) + \epsilon_i,$$

where $\epsilon_i \sim N(0, \sigma^2)$ and y_i is a continuous outcome.

- Approach is "Bayesian" insofar as we put (independent) priors on the parameters on Θ: depth of tree, variables used at the splits, terminal-node values, error variance.

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as $f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$
- Suppose we wrote this, instead:

$$y_i = \sum_{m=1}^{M} T_m(X_i; \Theta_m) + \epsilon_i,$$

where $\epsilon_i \sim N(0, \sigma^2)$ and y_i is a continuous outcome.

- Approach is "Bayesian" insofar as we put (independent) priors on the parameters on Θ: depth of tree, variables used at the splits, terminal-node values, error variance.
- priors control how much each tree affects the outcome surface (basically our $\hat{y}$ prediction, conditioned on X), in way that ensures no tree dominates

Bayesian (Additive Regression) Trees (BART)

- Recall ensembles can be written as $f(X_i) = \sum_{m=1}^{M} T_m(X_i; \Theta_m)$
- Suppose we wrote this, instead:

$$y_i = \sum_{m=1}^{M} T_m(X_i; \Theta_m) + \epsilon_i,$$

where $\epsilon_i \sim N(0, \sigma^2)$ and y_i is a continuous outcome.

- Approach is "Bayesian" insofar as we put (independent) priors on the parameters on Θ: depth of tree, variables used at the splits, terminal-node values, error variance.
- priors control how much each tree affects the outcome surface (basically our $\hat{y}$ prediction, conditioned on X), in way that ensures no tree dominates
- by MCMC we obtain posterior sample of M trees, and thus sample of $Y_i|X_i$: draws give measure of uncertainty over X relationship with Y.

Using Random Forests (in regression context). . .

Using Random Forests (in regression context)...

what stems are most important predictors of treaty harshness? (note: $p >> n$)

what stems are most important predictors of treaty harshness? (note: $p >> n$)

use reduction in mean square error when that term is used in a tree

Using Random Forests (in regression context)...

what stems are most important predictors of treaty harshness? (note: $p >> n$)

use reduction in mean square error when that term is used in a tree (increase in MSE when it is not used)

Using Random Forests (in regression context)...

what stems are most important predictors of treaty harshness? (note: $p >> n$)

use reduction in mean square error when that term is used in a tree (increase in MSE when it is not used)

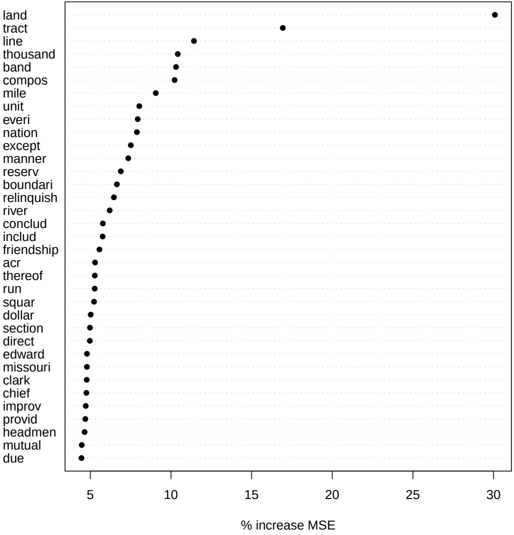
fairly robust to TDM construction choices

Using Random Forests (in regression context)...

what stems are most important predictors of treaty harshness? (note: $p >> n$)

use reduction in mean square error when that term is used in a tree (increase in MSE when it is not used)

fairly robust to TDM construction choices



Associations

Associations

stem	appearance	common phrasing (frequency)	ρ
friendship	friendship	"A treaty of peace and friendship" (15)	0.504
mutual	mutually	"shall be mutually forgiven and forgot" (19)	0.255
peac	peace	"A treaty of peace and friendship" (15)	0.179

Associations

stem	appearance	common phrasing (frequency)	ρ
`friendship`	friendship	"A treaty of peace and friendship" (15)	0.504
`mutual`	mutually	"shall be mutually forgiven and forgot" (19)	0.255
`peac`	peace	"A treaty of peace and friendship" (15)	0.179
`cession`	cession	"In consideration of the foregoing cession" (15)	-0.205
`relinquish`	relinquish	"cede and relinquish to the United States" (4)	-0.208
`boundari`	boundary	"land included within the following boundaries" (4)	-0.214
`tract`	tract	"One tract," (14)	-0.442
`dollar`	dollars	"forty dollars" (11)	-0.457
`land`	lands	"one section of land" (29)	-0.567
`reserv`	reservation	"one other reservation" (5)	-0.622

Ensembles

Better yet. . . Ensembles of Algorithms

Better yet. . . Ensembles of Algorithms

Plus can up-weight the most problematic observations

Better yet. . . Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers

Better yet. . . Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers when combining lots of trees:

Better yet. . . Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers when combining lots of trees: *boosting*.

Better yet. . . Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers when combining lots of trees: *boosting*.

Actually,

Better yet. . . Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers when combining lots of trees: *boosting*.

Actually, these ideas—bagging, forests, boosting—can be made more general:

Better yet...Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers when combining lots of trees: *boosting*.

Actually, these ideas—bagging, forests, boosting—can be made more general: ensemble learning involves bringing together different algorithms to produce better results (better prediction) than any one technique.

Better yet... Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers when combining lots of trees: *boosting*.

Actually, these ideas—bagging, forests, boosting—can be made more general: ensemble learning involves bringing together different algorithms to produce better results (better prediction) than any one technique.

e.g. Hillard, Purpura and Wilkerson are interested in predicting topic (20) and sub-topic (226) of $379,000$ Congressional bills.

Better yet... Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers when combining lots of trees: *boosting*.

Actually, these ideas—bagging, forests, boosting—can be made more general: ensemble learning involves bringing together different algorithms to produce better results (better prediction) than any one technique.

e.g. Hillard, Purpura and Wilkerson are interested in predicting topic (20) and sub-topic (226) of $379,000$ Congressional bills.

Better yet... Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers when combining lots of trees: *boosting*.

Actually, these ideas—bagging, forests, boosting—can be made more general: ensemble learning involves bringing together different algorithms to produce better results (better prediction) than any one technique.

e.g. Hillard, Purpura and Wilkerson are interested in predicting topic (20) and sub-topic (226) of $379,000$ Congressional bills.

Why? State of the art requires *months* to train human coders,

Better yet. . . Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers when combining lots of trees: *boosting*.

Actually, these ideas—bagging, forests, boosting—can be made more general: ensemble learning involves bringing together different algorithms to produce better results (better prediction) than any one technique.

e.g. Hillard, Purpura and Wilkerson are interested in predicting topic (20) and sub-topic (226) of $379,000$ Congressional bills.

Why? State of the art requires *months* to train human coders, although intercoder agreement is high.

Better yet. . . Ensembles of Algorithms

Plus can up-weight the most problematic observations and upweight the most accurate classifiers when combining lots of trees: *boosting*.

Actually, these ideas—bagging, forests, boosting—can be made more general: ensemble learning involves bringing together different algorithms to produce better results (better prediction) than any one technique.

e.g. Hillard, Purpura and Wilkerson are interested in predicting topic (20) and sub-topic (226) of $379,000$ Congressional bills.

Why? State of the art requires *months* to train human coders, although intercoder agreement is high.

Q Can they use supervised learning to do better? (better in terms of time: assume humans are accurate)

“Computer Assisted Topic Classification for Mixed-Methods Social Science Research”

“Computer Assisted Topic Classification for Mixed-Methods Social Science Research”

Split human coded data in two:

“Computer Assisted Topic Classification for Mixed-Methods Social Science Research”

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

“Computer Assisted Topic Classification for Mixed-Methods Social Science Research”

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing,

“Computer Assisted Topic Classification for Mixed-Methods Social Science Research”

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing, and then use four methods:

"Computer Assisted Topic Classification for Mixed-Methods Social Science Research"

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing, and then use four methods:

1. Naive Bayes

“Computer Assisted Topic Classification for Mixed-Methods Social Science Research”

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing, and then use four methods:

1. Naive Bayes
2. SVM

“Computer Assisted Topic Classification for Mixed-Methods Social Science Research”

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing, and then use four methods:

1. Naive Bayes
2. SVM
3. MaxEnt: like NB but features are weighted in way that maximizes likelihood of training set

"Computer Assisted Topic Classification for Mixed-Methods Social Science Research"

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing, and then use four methods:

1. Naive Bayes
2. SVM
3. MaxEnt: like NB but features are weighted in way that maximizes likelihood of training set
4. Boostexter: boosting algorithm that combines weak(er) classifiers

“Computer Assisted Topic Classification for Mixed-Methods Social Science Research”

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing, and then use four methods:

1. Naive Bayes
2. SVM
3. MaxEnt: like NB but features are weighted in way that maximizes likelihood of training set
4. Boostexter: boosting algorithm that combines weak(er) classifiers

In topic stage,

"Computer Assisted Topic Classification for Mixed-Methods Social Science Research"

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing, and then use four methods:

1. Naive Bayes
2. SVM
3. MaxEnt: like NB but features are weighted in way that maximizes likelihood of training set
4. Boostexter: boosting algorithm that combines weak(er) classifiers

In topic stage, compare predictions $\hat{y}_i \in \{1, \ldots, 20\}$ from methods:

"Computer Assisted Topic Classification for Mixed-Methods Social Science Research"

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing, and then use four methods:

1. Naive Bayes
2. SVM
3. MaxEnt: like NB but features are weighted in way that maximizes likelihood of training set
4. Boostexter: boosting algorithm that combines weak(er) classifiers

In topic stage, compare predictions $\hat{y}_i \in \{1, \ldots, 20\}$ from methods: allocate document to winner via some 'vote' aggregation system.

"Computer Assisted Topic Classification for Mixed-Methods Social Science Research"

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing, and then use four methods:

1. Naive Bayes
2. SVM
3. MaxEnt: like NB but features are weighted in way that maximizes likelihood of training set
4. Boostexter: boosting algorithm that combines weak(er) classifiers

In topic stage, compare predictions $\hat{y}_i \in \{1, \ldots, 20\}$ from methods: allocate document to winner via some 'vote' aggregation system.

In subtopic stage,

“Computer Assisted Topic Classification for Mixed-Methods Social Science Research”

Split human coded data in two: $187,000$ in training set, $187,000$ in test set.

Some preprocessing, and then use four methods:

1. Naive Bayes
2. SVM
3. MaxEnt: like NB but features are weighted in way that maximizes likelihood of training set
4. Boostexter: boosting algorithm that combines weak(er) classifiers

In topic stage, compare predictions $\hat{y}_i \in \{1, \ldots, 20\}$ from methods: allocate document to winner via some ‘vote’ aggregation system.

In subtopic stage, use SVM (as best performer).

Results

Results

TABLE 3. Bill Title Interannotator Agreement for Five Model Types

	SVM	MaxEnt	Boostexter	Naïve Bayes	Ensemble
Major topic $N = 20$	88.7% (.881)	86.5% (.859)	85.6% (.849)	81.4% (.805)	89.0% (.884)
Subtopic $N = 226$	81.0% (.800)	78.3% (.771)	73.6% (.722)	71.9% (.705)	81.0% (.800)

Note. Results are based on using approximately 187,000 human-labeled cases to train the classifier to predict approximately 187,000 other cases (that were also labeled by humans but not used for training). Agreement is computed by comparing the machine's prediction to the human assigned labels. (AC1 measure presented in parentheses).

Results

TABLE 3. Bill Title Interannotator Agreement for Five Model Types

	SVM	MaxEnt	Boostexter	Naïve Bayes	Ensemble
Major topic $N = 20$	88.7% (.881)	86.5% (.859)	85.6% (.849)	81.4% (.805)	89.0% (.884)
Subtopic $N = 226$	81.0% (.800)	78.3% (.771)	73.6% (.722)	71.9% (.705)	81.0% (.800)

Note. Results are based on using approximately 187,000 human-labeled cases to train the classifier to predict approximately 187,000 other cases (that were also labeled by humans but not used for training). Agreement is computed by comparing the machine's prediction to the human assigned labels. (AC1 measure presented in parentheses).

AC1 is intercoder reliability corrected for chance agreement.

Results

TABLE 3. Bill Title Interannotator Agreement for Five Model Types

	SVM	MaxEnt	Boostexter	Naïve Bayes	Ensemble
Major topic $N = 20$	88.7% (.881)	86.5% (.859)	85.6% (.849)	81.4% (.805)	89.0% (.884)
Subtopic $N = 226$	81.0% (.800)	78.3% (.771)	73.6% (.722)	71.9% (.705)	81.0% (.800)

Note. Results are based on using approximately 187,000 human-labeled cases to train the classifier to predict approximately 187,000 other cases (that were also labeled by humans but not used for training). Agreement is computed by comparing the machine's prediction to the human assigned labels. (AC1 measure presented in parentheses).

AC1 is intercoder reliability corrected for chance agreement.

Improvement over SVM alone is real,

Results

TABLE 3. Bill Title Interannotator Agreement for Five Model Types

	SVM	MaxEnt	Boostexter	Naïve Bayes	Ensemble
Major topic $N = 20$	88.7% (.881)	86.5% (.859)	85.6% (.849)	81.4% (.805)	89.0% (.884)
Subtopic $N = 226$	81.0% (.800)	78.3% (.771)	73.6% (.722)	71.9% (.705)	81.0% (.800)

Note. Results are based on using approximately 187,000 human-labeled cases to train the classifier to predict approximately 187,000 other cases (that were also labeled by humans but not used for training). Agreement is computed by comparing the machine's prediction to the human assigned labels. (AC1 measure presented in parentheses).

AC1 is intercoder reliability corrected for chance agreement.

Improvement over SVM alone is real, though small.

Results

TABLE 3. Bill Title Interannotator Agreement for Five Model Types

	SVM	MaxEnt	Boostexter	Naïve Bayes	Ensemble
Major topic $N = 20$	88.7% (.881)	86.5% (.859)	85.6% (.849)	81.4% (.805)	89.0% (.884)
Subtopic $N = 226$	81.0% (.800)	78.3% (.771)	73.6% (.722)	71.9% (.705)	81.0% (.800)

Note. Results are based on using approximately 187,000 human-labeled cases to train the classifier to predict approximately 187,000 other cases (that were also labeled by humans but not used for training). Agreement is computed by comparing the machine's prediction to the human assigned labels. (AC1 measure presented in parentheses).

AC1 is intercoder reliability corrected for chance agreement.

Improvement over SVM alone is real, though small.

Classification especially good in cases where methods agree on topic.

Results

TABLE 3. Bill Title Interannotator Agreement for Five Model Types

	SVM	MaxEnt	Boostexter	Naïve Bayes	Ensemble
Major topic $N = 20$	88.7% (.881)	86.5% (.859)	85.6% (.849)	81.4% (.805)	89.0% (.884)
Subtopic $N = 226$	81.0% (.800)	78.3% (.771)	73.6% (.722)	71.9% (.705)	81.0% (.800)

Note. Results are based on using approximately 187,000 human-labeled cases to train the classifier to predict approximately 187,000 other cases (that were also labeled by humans but not used for training). Agreement is computed by comparing the machine's prediction to the human assigned labels. (AC1 measure presented in parentheses).

AC1 is intercoder reliability corrected for chance agreement.

Improvement over SVM alone is real, though small.

Classification especially good in cases where methods agree on topic.