Big Data & Automated Content Analysis Week 8 – Wednesday: »Basics of Machine Learning«

Damian Trilling

d.c.trilling@uva.nl
@damian0604
www.damiantrilling.net

31 March 2021

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

Today

Recap: Top-down vs bottom-up

Finding similar variables

- An introduction to dimensionality reduction
- Principal Component Analysis and Singular Value Decomposition

Finding similar cases

- k-means clustering
- Hierarchical clustering

Predicting things

- You have done it before!
- From regression to classification

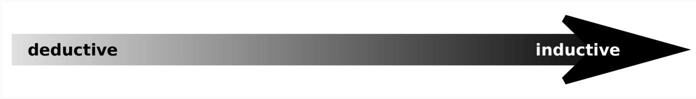
Summing up

Today, the great overview.

Next weeks: Specific applications for content analysis.

Recap

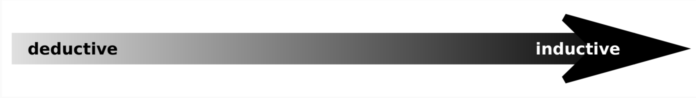
	Methodological approach		
	Counting and Dictionary	*Supervised Machine Learning*	*Unsupervised Machine Learning*
Typical research interests and content features	visibility analysis sentiment analysis subjectivity analysis	frames topics gender bias	frames topics
Common statistical procedures	string comparisons counting	support vector machines naive Bayes	principal component analysis cluster analysis latent dirichlet allocation semantic network analysis



Boumans and Trilling, 2016

The same logic applies to non-textual data!

	Methodological approach		
	Counting and Dictionary	*Supervised Machine Learning*	*Unsupervised Machine Learning*
Typical research interests and content features	visibility analysis sentiment analysis subjectivity analysis	frames topics gender bias	frames topics
Common statistical procedures	string comparisons counting	support vector machines naive Bayes	principal component analysis cluster analysis latent dirichlet allocation semantic network analysis



Boumans and Trilling, 2016

The same logic applies to non-textual data!

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured x1, x2, x3 and you want to predict y, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure y)

Again, you already know some techniques to find out how x1, x2,…x_i co-occur from other courses:

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling (Non-negative matrix factorization and Latent

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure y)

Again, you already know some techniques to find out how x1, x2,... x_i co-occur from other courses:

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling (Non-negative matrix factorization and Latent

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure `y`)
Again, you already know some techniques to find out how `x1`, `x2`,...`x_i` co-occur from other courses:

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling (Non-negative matrix factorization and Latent

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure `y`)

Again, you already know some techniques to find out how `x1`, `x2`,...`x_i` co-occur from other courses:

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling (Non-negative matrix factorization and Latent

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure `y`)
Again, you already know some techniques to find out how `x1`, `x2`,`...x_i` co-occur from other courses:

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling (Non-negative matrix factorization and Latent

Let's distinguish four use cases...

1. Finding similar variables (dimensionality reduction) – unsupervised
2. Finding similar cases (clustering) – unsupervised
3. Predicting a continous variable (regression) – supervised
4. Predicting group membership (classification) – supervised

	x1	x2	x3	x4	x5	y
case1	■	■	■	■	■	■
case2	■	■	■	■	■	■
case3	■	■	■	■	■	■
case4	■	■	■	■	■	■

	x1	x2	x3	x4	x5	(y)
case1	🟧	⬛	🟧	🟦	🟦	(◼)
case2	🟧	⬛	🟧	🟦	🟦	(◼)
case3	🟧	⬛	🟧	🟦	🟦	(◼)
case4	🟧	⬛	🟧	🟦	🟦	(◼)

Dimensionality reduction: finding similar variables (features)

	x1	x2	x3	x4	x5	(y)
case1	🟧	🟧	🟧	🟧	🟧	(⬛)
case2	🟦	🟦	🟦	🟦	🟦	(⬛)
case3	🟧	🟧	🟧	🟧	🟧	(⬛)
case4	🟧	🟧	🟧	🟧	🟧	(⬛)

Clustering: finding similar cases

	x1	x2	x3	x4	x5	→	y
case1	■	■	■	■	■	→	🟧
case2	■	■	■	■	■	→	🟧
case3	■	■	■	■	■	→	🟧
case4	■	■	■	■	■	→	🟧
new case	■	■	■	■	■	→	?

Regression and classification: learn how to predict y.

Note, again, that the ■ signs can be *anything*. For us, often word counts or *tf* · idf scores (x) and, for supervised approaches, a topic, a sentiment, or similar (y).

But it could also be pixel colors or clicks on links or anything else.

	x1	x2	x3	x4	x5	y
case1	■	■	■	■	■	■
case2	■	■	■	■	■	■
case3	■	■	■	■	■	■
case4	■	■	■	■	■	■

Finding similar variables

Finding similar variables

An introduction to dimensionality reduction

Dimensionality reduction

dimensionality = the number of features we have

(1) Explorative data analysis and visualization

- No good way to visualize 10,000 dimensions (or even 4)

(2) The curse of dimensionality

More features means more data (good!), but:

- Too many features can lead to unfeasible computation times
- We need more training cases to increase the likelihood that the possible combinations actually occur

Dimensionality reduction

dimensionality = the number of features we have

(1) Explorative data analysis and visualization

- No good way to visualize 10,000 dimensions (or even 4)

(2) The curse of dimensionality

More features means more data (good!), but:

- Too many features can lead to unfeasible computation times
- We need more training cases to increase the likelihood that the possible combinations actually occur

Dimensionality reduction

First approach: feature selection

- Only choose the features that are really relevant

Example of what we did before: Exclude all terms that occur in more than 50% of the documents, or in less than $n = 5$ documents:

```
vec = CountVectorizer(max_df=0.5, min_df=5)
```

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

Dimensionality reduction

Second approach: feature extraction

- Create a smaller set of features
- E.g.: 1,000 features $\rightarrow$ PCA to reduce to 50 components $\rightarrow$ SML with these 50 component scores as features

Dimensionality reduction

So, we can use unsuvised ML as a dimension reduction step in a supervised ML pipeline.

But it can also be a goal in itself, to understand the data better or to visualize them.

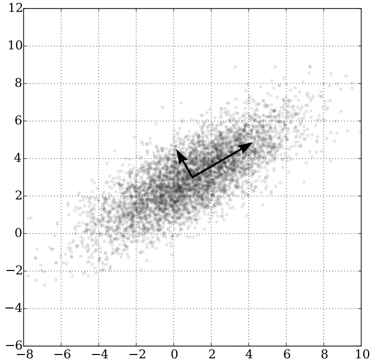
Finding similar variables

Principal Component Analysis and Singular Value Decomposition

PCA

- related to and often confused with Factor Analysis (same menu item in SPSS – many people who believe they run FA actually run PCA!)
- Components are ordered (first explains most variance)
- Components do *not* necessarily carry a meaningful interpretation

PCA



https://upload.wikimedia.org/wikipedia/commons/f/f5/GaussianScatterPCA.svg

Running PCA

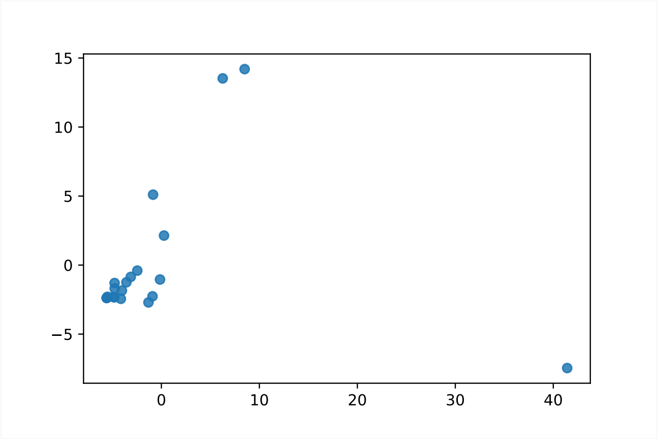
Example of a PCA on a BOW representation of some texts:

```
myvec = CountVectorizer(texts, max_df=.5, min_df=5)
mypca = PCA(n_components=2)

mypipe = make_pipeline(myvec, FunctionTransformer(lambda x: x.todense(),
      accept_sparse=True), mypca)

r = mypipe.fit_transform(texts)
```

PCA does not accept a *sparse matrix* as input (but the CountVectorizer gives one as output), so we need to transform it into a *dense matrix*.

Plotting the result

```
plt.scatter([e[0] for e in r], [e[1] for e in r], alpha=.6)
```



Singular value decomposition

The need to use a dense matrix is *really* a problem for large feature sets (which we have in NLP).

We therefore can better use SVD, which is essentially* the same and very simple to use:

```
mysvd = TruncatedSVD(n_components=2)
mypipe = make_pipeline(myvec, mysvd)
r = mypipe.fit_transform(texts)
```

(In this specific case, we even get exactly the same plot. . .)

* It's mathematically different, but you can SVD is even used "under the hood" by several PCA modules to solve PCA problems.

More info and background: https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8

Singular value decomposition

The need to use a dense matrix is *really* a problem for large feature sets (which we have in NLP).

We therefore can better use SVD, which is essentially* the same and very simple to use:

```
mysvd = TruncatedSVD(n_components=2)
mypipe = make_pipeline(myvec, mysvd)
r = mypipe.fit_transform(texts)
```

(In this specific case, we even get exactly the same plot. . .)

* It's mathematically different, but you can SVD is even used "under the hood" by several PCA modules to solve PCA problems.

More info and background: https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8

Finding similar cases

Finding similar cases

k-means clustering

Grouping features vs grouping cases

Let's consider a corpus of several thousand user comments.

We could use SVD or similar techniques to

- figure out relationships between features
- see which features stand out
- get a first sense what topics are in the corpus.

But:

- We do not learn anything about *which* texts (cases) belong to which topic
- We could use the component scores returned by `.fit_transform()` to then group our cases.

⇒ Alternative: Choose the opposite approach and first find out which cases are most similar, *then* describe what

Grouping features vs grouping cases

Let's consider a corpus of several thousand user comments.

We could use SVD or similar techniques to

- figure out relationships between features
- see which features stand out
- get a first sense what topics are in the corpus.

But:

- We do not learn anything about *which* texts (cases) belong to which topic
- We could use the component scores returned by `.fit_transform()` to then group our cases

⇒ Alternative: Choose the opposite approach and first find out which cases are most similar, *then* describe what

Grouping features vs grouping cases

Let's consider a corpus of several thousand user comments.

We could use SVD or similar techniques to

- figure out relationships between features
- see which features stand out
- get a first sense what topics are in the corpus.

But:

- We do not learn anything about *which* texts (cases) belong to which topic
- We could use the component scores returned by `.fit_transform()` to then group our cases

⇒ Alternative: Choose the opposite approach and first find out which cases are most similar, *then* describe what

Grouping features vs grouping cases

Let's consider a corpus of several thousand user comments.

We could use SVD or similar techniques to

- figure out relationships between features
- see which features stand out
- get a first sense what topics are in the corpus.

But:

- We do not learn anything about *which* texts (cases) belong to which topic
- We could use the component scores returned by `.fit_transform()` to then group our cases

$\Rightarrow$ Alternative: Choose the opposite approach and first find out which cases are most similar, *then* describe what

Grouping features vs grouping cases

Let's consider a corpus of several thousand user comments.

We could use SVD or similar techniques to

- figure out relationships between features
- see which features stand out
- get a first sense what topics are in the corpus.

But:

- We do not learn anything about *which* texts (cases) belong to which topic
- We could use the component scores returned by `.fit_transform()` to then group our cases

$\Rightarrow$ **Alternative: Choose the opposite approach and first find out which cases are most similar, *then* describe what**

k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

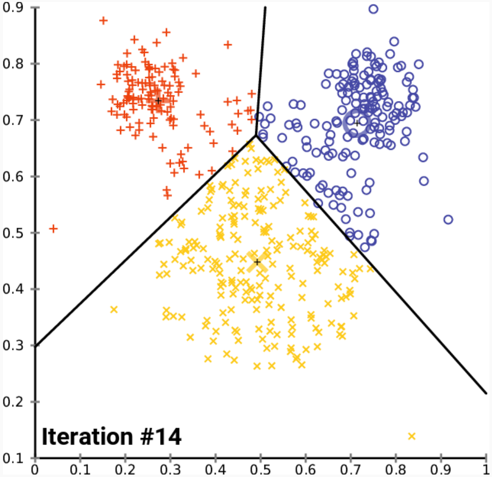
k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

k-means clustering



https://upload.wikimedia.org/wikipedia/commons/e/ea/K-means_convergence.gif

Notice the big symbols indicating the centroids.

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans

k = 5
texts = ['text1 ejkh ek ekh', 'ekyerykel'] # a list of texts

vec = TfidfVectorizer(min_df=5, max_df=.4)
features = vec.fit_transform(texts)
km = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=1)
predictions = km.fit_predict(features)
```

That's it!

- `predictions` is a list of integers indicated the predicted cluster number. We can thus use `zip(predictions, texts)` to put them together.
- We could also use `.fit()` and `.transform()` sperately and use our km to predict clusters for additional cases we have not used to train the model

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans

k = 5
texts = ['text1 ejkh ek ekh', 'ekyerykel'] # a list of texts

vec = TfidfVectorizer(min_df=5, max_df=.4)
features = vec.fit_transform(texts)
km = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=1)
predictions = km.fit_predict(features)
```

That's it!

- `predictions` is a list of integers indicated the predicted cluster number. We can thus use `zip(predictions, texts)` to put them together.
- We could also use `.fit()` and `.transform()` sperately and use our `km` to predict clusters for additional cases we have not used to train the model

Let's get the terms closest to the centroids

```python
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
terms = vec.get_feature_names()

print("Top terms per cluster:")

for i in range(k):
    print("Cluster {}: ".format(i), end='')
    for ind in order_centroids[i, :10]:
        print("{} ".format(terms[ind]), end='')
        print()
```

returns something like:

```
Top terms per cluster:
Cluster 0: heard could if opinions info day how really just around
Cluster 1: systems would ken pc am if as care summary ibm
Cluster 2: year car years was my no one higher single than
Cluster 3: which like seen 1000 few easily based personal work used
Cluster 4: as was he if they my all will get has
```

Let’s get the terms closest to the centroids

```
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
terms = vec.get_feature_names()

print("Top terms per cluster:")

for i in range(k):
    print("Cluster {}: ".format(i), end='')
    for ind in order_centroids[i, :10]:
        print("{} ".format(terms[ind]), end='')
        print()
```

returns something like:

```
Top terms per cluster:
Cluster 0: heard could if opinions info day how really just around
Cluster 1: systems would ken pc am if as care summary ibm
Cluster 2: year car years was my no one higher single than
Cluster 3: which like seen 1000 few easily based personal work used
Cluster 4: as was he if they my all will get has
```

Using k-means clustering...

- we get the cluster membership for each text; and
- we get the terms that are most characteristic for the documents in each cluster.

Finding the optimal k

- The only way to find k is to estimate multiple models with different ks
- No single best solution; finding a balance between error within clusters (distances from centroid) and low number of clusters.
- An elbow plot can be helpful

Finding similar cases

Hierarchical clustering

Downsides of k-means clustering

k-means is fast, but has problems:

- k can only be determined by fitting multiple models and comparing them
- bad results if the wrong k is chosen
- bad results if the (real) clusters are non-spherical
- bad results if the (real) clusters are not evenly sized

Hiearchical clusttering

General idea

- To start, each case has its own cluster
- Merge the two clusters that are most similar
- Repeat until desired number of clusters is reached

Different options

- Stopping criterion: based on numerical statistic (e.g., Duda-Hart) or dendrogram
- Linkage: how to determine which two clusters should be merged?

Hiearchical clusttering

General idea

- To start, each case has its own cluster
- Merge the two clusters that are most similar
- Repeat until desired number of clusters is reached

Different options

- Stopping criterion: based on numerical statistic (e.g., Duda-Hart) or dendrogram
- Linkage: how to determine which two clusters should be merged?

Let's look into some options

https://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering

$\Rightarrow$ Ward's linkage is a good default all-rounder choice, especially if you encounter the problem that other linkages lead to almost all cases ending up in one cluster.

Hierarchical clustering takeaway

- The main reason *not* to use hierarchical methods (but k-means) is their computational cost: when clustering survey data of media users, never use *k*-means!
- But for NLP/ML, costs may be too high (if not used carefully)
- Very much worth considering, though, if you are really into grouping cases!

Important notes for all techniques

Consider the scales of measurement

Clustering is based on distances – if your features are not measured on the same scale, or if it is not meaningful to calculate a numerical distance, it won't produce meaningful results!

Consider standardizing/whitening your features!

Pay attention outliers/extreme cases

Extreme cases or outliers can have a strong influence.

Do proper pre-processing

To reduce the number of features, but also to have *meaningful* features (dimensions on which you expect high distances between the clusters).

Important notes for all techniques

Consider the scales of measurement

Clustering is based on distances – if your features are not measured on the same scale, or if it is not meaningful to calculate a numerical distance, it won't produce meaningful results!

Consider standardizing/whitening your features!

Pay attention outliers/extreme cases

Extreme cases or outliers can have a strong influence.

Do proper pre-processing

To reduce the number of features, but also to have *meaningful* features (dimensions on which you expect high distances between the clusters).

Important notes for all techniques

Consider the scales of measurement

Clustering is based on distances – if your features are not measured on the same scale, or if it is not meaningful to calculate a numerical distance, it won't produce meaningful results!

Consider standardizing/whitening your features!

Pay attention outliers/extreme cases

Extreme cases or outliers can have a strong influence.

Do proper pre-processing

To reduce the number of features, but also to have *meaningful* features (dimensions on which you expect high distances between the clusters).

Predicting things

Predicting things

You have done it before!

You have done it before!

Regression

1. Based on your data, you estimate some regression equation $y_i = \alpha + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i$
2. Even if you have some *new unseen data*, you can estimate your expected outcome $\hat{y}$!
3. Example: You estimated a regression equation where y is newspaper reading in days/week:
   $y = -.8 + .4 \times man + .08 \times age$
4. You could now calculate $\hat{y}$ for a man of 20 years and a woman of 40 years – *even if no such person exists in your dataset*:
   $\hat{y}_{man20} = -.8 + .4 \times 1 + .08 \times 20 = 1.2$
   $\hat{y}_{woman40} = -.8 + .4 \times 0 + .08 \times 40 = 2.4$

You have done it before!

Regression

1. Based on your data, you estimate some regression equation $y_i = \alpha + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i$
2. Even if you have some *new unseen data*, you can estimate your expected outcome $\hat{y}$!
3. Example: You estimated a regression equation where y is newspaper reading in days/week: $y = -.8 + .4 \times man + .08 \times age$
4. You could now calculate $\hat{y}$ for a man of 20 years and a woman of 40 years – *even if no such person exists in your dataset*: $\hat{y}_{man20} = -.8 + .4 \times 1 + .08 \times 20 = 1.2$ $\hat{y}_{woman40} = -.8 + .4 \times 0 + .08 \times 40 = 2.4$

You have done it before!

Regression

1. Based on your data, you estimate some regression equation $y_i = \alpha + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i$
2. Even if you have some *new unseen data*, you can estimate your expected outcome $\hat{y}$!
3. Example: You estimated a regression equation where y is newspaper reading in days/week: $y = -.8 + .4 \times man + .08 \times age$
4. You could now calculate $\hat{y}$ for a man of 20 years and a woman of 40 years – *even if no such person exists in your dataset*: $\hat{y}_{man20} = -.8 + .4 \times 1 + .08 \times 20 = 1.2$ $\hat{y}_{woman40} = -.8 + .4 \times 0 + .08 \times 40 = 2.4$

You have done it before!

Regression

1. Based on your data, you estimate some regression equation $y_i = \alpha + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i$
2. Even if you have some *new unseen data*, you can estimate your expected outcome $\hat{y}$!
3. Example: You estimated a regression equation where y is newspaper reading in days/week: $y = -.8 + .4 \times man + .08 \times age$
4. You could now calculate $\hat{y}$ for a man of 20 years and a woman of 40 years – *even if no such person exists in your dataset*: $\hat{y}_{man20} = -.8 + .4 \times 1 + .08 \times 20 = 1.2$ $\hat{y}_{woman40} = -.8 + .4 \times 0 + .08 \times 40 = 2.4$

You have done it before!

Regression

1. Based on your data, you estimate some regression equation $y_i = \alpha + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i$
2. Even if you have some *new unseen data*, you can estimate your expected outcome $\hat{y}$!
3. Example: You estimated a regression equation where y is newspaper reading in days/week: $y = -.8 + .4 \times man + .08 \times age$
4. You could now calculate $\hat{y}$ for a man of 20 years and a woman of 40 years – *even if no such person exists in your dataset:* $\hat{y}_{man20} = -.8 + .4 \times 1 + .08 \times 20 = 1.2$ $\hat{y}_{woman40} = -.8 + .4 \times 0 + .08 \times 40 = 2.4$

You have done it before!

Regression

1. Based on your data, you estimate some regression equation $y_i = \alpha + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i$
2. Even if you have some *new unseen data*, you can estimate your expected outcome $\hat{y}$!
3. Example: You estimated a regression equation where y is newspaper reading in days/week: $y = -.8 + .4 \times man + .08 \times age$
4. You could now calculate $\hat{y}$ for a man of 20 years and a woman of 40 years – *even if no such person exists in your dataset*: $\hat{y}_{man20} = -.8 + .4 \times 1 + .08 \times 20 = 1.2$ $\hat{y}_{woman40} = -.8 + .4 \times 0 + .08 \times 40 = 2.4$

This is
Supervised Machine Learning!

. . . but. . .

- We will only use *half* (or another fraction) of our data to estimate the model, so that we can use the other half to check if our predictions match the manual coding (“labeled data”,“annotated data” in SML-lingo)
    - e.g., 2000 labeled cases, 1000 for training, 1000 for testing — if successful, run on 100,000 unlabeled cases
- We use many more independent variables (“features”)
- Typically, IVs are word frequencies (often weighted, e.g. tf$\times$idf) ($\Rightarrow$BOW-representation)

. . . but. . .

- We will only use *half* (or another fraction) of our data to estimate the model, so that we can use the other half to check if our predictions match the manual coding (“labeled data”,“annotated data” in SML-lingo)
  - e.g., 2000 labeled cases, 1000 for training, 1000 for testing — if successful, run on 100,000 unlabeled cases
- We use many more independent variables (“features”)
- Typically, IVs are word frequencies (often weighted, e.g. tf$\times$idf) ($\Rightarrow$BOW-representation)

. . . but. . .

- We will only use *half* (or another fraction) of our data to estimate the model, so that we can use the other half to check if our predictions match the manual coding ("labeled data","annotated data" in SML-lingo)
    - e.g., 2000 labeled cases, 1000 for training, 1000 for testing — if successful, run on 100,000 unlabeled cases
- We use many more independent variables ("features")
- Typically, IVs are word frequencies (often weighted, e.g. tf$\times$idf) ($\Rightarrow$BOW-representation)

. . . but. . .

- We will only use *half* (or another fraction) of our data to estimate the model, so that we can use the other half to check if our predictions match the manual coding ("labeled data","annotated data" in SML-lingo)
    - e.g., 2000 labeled cases, 1000 for training, 1000 for testing — if successful, run on 100,000 unlabeled cases
- We use many more independent variables ("features")
- Typically, IVs are word frequencies (often weighted, e.g. tf$\times$idf) ($\Rightarrow$BOW-representation)

Predicting things

From regression to classification

In the machine learning world, predicting some continous value is referred to as a **regression** task. If we want to predict a binary or categorical variable, we call it a **classification** task.

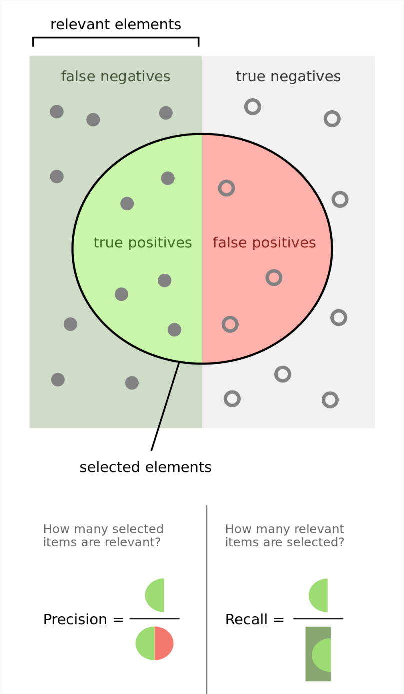
(quite confusingly, even if we use a logistic regression for the latter)

Classification tasks

For many computational approaches, we are actually not that interested in predicting a continous value. Typical questions include:

- Is this article about topix A, B, C, D, or E?
- Is this review positive or negative?
- Does this text contain frame F?
- I this satire?
- Is this misinformation?
- Given past behavior, can I predict the next click?

http://commons.wikimedia.org/wiki/File:Precisionrecall.svg



Some measures

- Accuracy
- Recall
- Precision
- $\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$
- AUC (Area under curve) $[0, 1]$, $0.5 =$ random guessing

Different classification algorithms

- It is an empirical question which one works best
- We typically try several ones and select the best
- (remember: we have a test dataset that we did *not* use to train the model, so that we can assess how well it predicts the test labels based on the test features)
- To avoid *p*-hacking-like scenario's (which we call "overfitting"), there are techniques available (cross-validation, later in this course)

(to make it easier, imagine a binary classfication ("positive"/"negative"), but it doesn't really matter whether there are two or more labels)

Different classification algorithms

- It is an empirical question which one works best
- We typically try several ones and select the best
- (remember: we have a test dataset that we did *not* use to train the model, so that we can assess how well it predicts the test labels based on the test features)
- To avoid *p*-hacking-like scenario's (which we call "overfitting"), there are techniques available (cross-validation, later in this course)

(to make it easier, imagine a binary classfication ("positive"/"negative"), but it doesn't really matter whether there are two or more labels)

Different classification algorithms

- It is an empirical question which one works best
- We typically try several ones and select the best
- (remember: we have a test dataset that we did *not* use to train the model, so that we can assess how well it predicts the test labels based on the test features)
- To avoid p-hacking-like scenario's (which we call "overfitting"), there are techniques available (cross-validation, later in this course)

(to make it easier, imagine a binary classfication ("positive"/"negative"), but it doesn't really matter whether there are two or more labels)

Different classification algorithms

- It is an empirical question which one works best
- We typically try several ones and select the best
- (remember: we have a test dataset that we did *not* use to train the model, so that we can assess how well it predicts the test labels based on the test features)
- To avoid *p*-hacking-like scenario's (which we call “overfitting”), there are techniques available (cross-validation, later in this course)

(to make it easier, imagine a binary classfication ("positive"/"negative"), but it doesn't really matter whether there are two or more labels)

Naïve Bayes

Bayes' theorem

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)}$$

A = Text is about sports

B = Text contains 'very', 'close', 'game'. Furthermore, we simply multiply the propabilities for the features:

$$P(B) = P(very\ close\ game) = P(very) \times P(close) \times P(game)$$

We can fill in all values by counting how many articles are about sports, and how often the words occur in these texts. (Fully elaborated example on https://monkeylearn.com/blog/practical-explanation-naive-bayes-classifier/)

Naïve Bayes

Bayes' theorem

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)}$$

A = Text is about sports

B = Text contains 'very', 'close', 'game'. Furthermore, we simply multiply the propabilities for the features:

$$P(B) = P(very\ close\ game) = P(very) \times P(close) \times P(game)$$

We can fill in all values by counting how many articles are about sports, and how often the words occur in these texts. (Fully elaborated example on https://monkeylearn.com/blog/practical-explanation-naive-bayes-classifier/)

Naïve Bayes

Bayes' theorem

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)}$$

A = Text is about sports

B = Text contains 'very', 'close', 'game'. Furthermore, we simply multiply the propabilities for the features:

$$P(B) = P(\textit{very close game}) = P(\textit{very}) \times P(\textit{close}) \times P(\textit{game})$$

We can fill in all values by counting how many articles are about sports, and how often the words occur in these texts. (Fully elaborated example on https://monkeylearn.com/blog/practical-explanation-naive-bayes-classifier/)

Naïve Bayes

- It's "naïve" because the features are treated as completely independent ($\neq$ "controlling" in regression analysis)
- It's fast and easy
- It's a good *baseline* for binary classification problems

Naïve Bayes

- It's "naïve" because the features are treated as completely independent ($\neq$ "controlling" in regression analysis)
- It's fast and easy
- It's a good *baseline* for binary classification problems

Naïve Bayes

- It's "naïve" because the features are treated as completely independent ($\neq$ "controlling" in regression analysis)
- It's fast and easy
- It's a good *baseline* for binary classification problems

Naïve Bayes

$$P(\text{label} \mid \text{features}) = \frac{P(x_1 \mid label) \cdot P(x_2 \mid \text{label}) \ \cdot \mathrm{P}(\mathrm{x}_3 \mid \text{label}) \cdot \mathrm{P}(\text{label})}{P(x_1) \cdot P(x_2) \cdot P(x_3)}$$

.

- Formulas always look intimidating, but we only need to fill in how many documents containing feature x_n have the label, how often the label occurs, and how often each feature occurs
- Also for computers, this is *really easy and fast*
- Weird assumption: features are independent
- Often used as a baseline

Logistic Regression

Probability of a binary outcome in a regression model

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n)}}$$

Just like in OLS regression, we have an intercept and regression coefficients. We use a threshold (default: 0.5) and above, we assign the positive label (‘good movie’), below, the negative label (‘bad movie’).

Logistic Regression

- The features are *not* independent.
- Computationally more expensive than Naïve Bayes
- We can get probabilities instead of just a label
- That allows us to say how sure we are for a specific case
- . . . or to change the threshold to change our precision/recall-tradeoff

Logistic Regression

- The features are *not* independent.
- Computationally more expensive than Naïve Bayes
- We can get probabilities instead of just a label
- That allows us to say how sure we are for a specific case
- . . . or to change the threshold to change our precision/recall-tradeoff

Logistic Regression

- The features are *not* independent.
- Computationally more expensive than Naïve Bayes
- We can get probabilities instead of just a label
- That allows us to say how sure we are for a specific case
- ...or to change the threshold to change our precision/recall-tradeoff

Logistic Regression

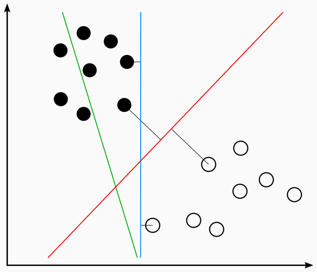
- The features are *not* independent.
- Computationally more expensive than Naïve Bayes
- We can get probabilities instead of just a label
- That allows us to say how sure we are for a specific case
- . . . or to change the threshold to change our precision/recall-tradeoff

Logistic Regression

- The features are *not* independent.
- Computationally more expensive than Naïve Bayes
- We can get probabilities instead of just a label
- That allows us to say how sure we are for a specific case
- . . . or to change the threshold to change our precision/recall-tradeoff

Support Vector Machines

- Idea: Find a hyperplane that best seperates your cases
- Can be linear, but does not have to be (depends on the so-called kernel you choose)
- Very popular



https://upload.wikimedia.org/wikipedia/commons/b/b5/Svm_separating_hyperplanes_%28SVG%29.svg

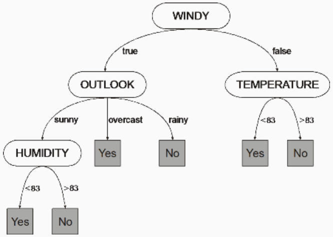
(Further reading: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/)

SVM vs logistic regression

- for *linearly separable* classes not much difference
- with the right hyperparameters, SVM is less sensitive to outliers
- biggest advantage: with the *kernel trick*, data can be transformed that they *become* linearily separable

Decision Trees and Random Forests

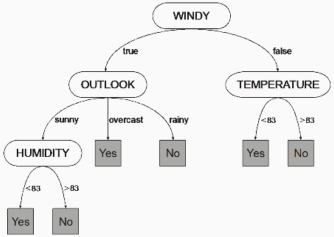
- Model problem as a series of decisions (e.g., if cloudy then . . . if temperature > 30 degrees then . . .)
- Order and cutoff-points are determined by an algorithm
- Big advantage: Model non-linear relationships
- And: They are easy to interpret (!) (“white box”)



https://upload.wikimedia.org/wikipedia/en/4/4f/GEP_decision_tree_with_numeric_and_nominal_attributes.png

Decision Trees and Random Forests

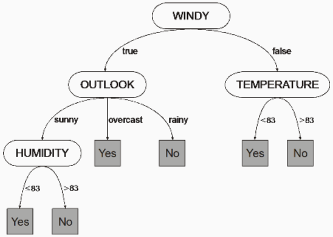
- Model problem as a series of decisions (e.g., if cloudy then . . . if temperature > 30 degrees then . . .)
- Order and cutoff-points are determined by an algorithm
- Big advantage: Model non-linear relationships
- And: They are easy to interpret (!) (“white box”)



https://upload.wikimedia.org/wikipedia/en/4/4f/GEP_decision_tree_with_numeric_and_nominal_attributes.png

Decision Trees and Random Forests

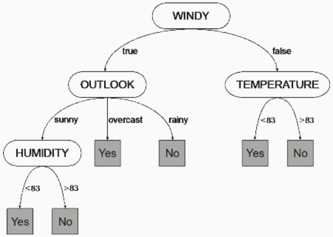
- Model problem as a series of decisions (e.g., if cloudy then . . . if temperature > 30 degrees then . . .)
- Order and cutoff-points are determined by an algorithm
- Big advantage: Model non-linear relationships
- And: They are easy to interpret (!) (“white box”)



https://upload.wikimedia.org/wikipedia/en/4/4f/GEP_decision_tree_with_numeric_and_nominal_attributes.png

Decision Trees and Random Forests

- Model problem as a series of decisions (e.g., if cloudy then . . . if temperature > 30 degrees then . . .)
- Order and cutoff-points are determined by an algorithm
- Big advantage: Model non-linear relationships
- And: They are easy to interpret (!) ("white box")



https://upload.wikimedia.org/wikipedia/en/4/4f/GEP_decision_tree_with_numeric_and_nominal_attributes.png

Decision Trees and Random Forests

Disadvantages of decision trees

- comparatively inaccurate
- once you are in the wrong branch, you cannot go ‘back up’
- prone to overfitting (e.g., outlier in training data may lead to completely different outcome)

Therfore, nowadays people use *random forests*: Random forests *combine* the predictions of *multiple* trees $\Rightarrow$ might be a good choice for your non-linear classification problem

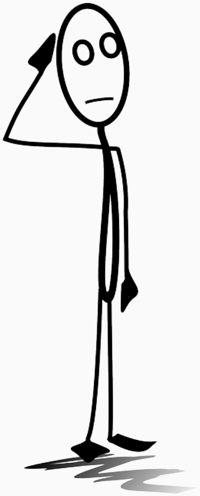
Decision Trees and Random Forests

Disadvantages of decision trees

- comparatively inaccurate
- once you are in the wrong branch, you cannot go ‘back up’
- prone to overfitting (e.g., outlier in training data may lead to completely different outcome)

Therfore, nowadays people use *random forests*: Random forests *combine* the predictions of *multiple* trees $\Rightarrow$ might be a good choice for your non-linear classification problem

Summing up



Any questions?

Things to remember

- unsupervised vs supervised
- rough understanding of different techniques and when to use them
- evaluation metrics (e.g., precision, recall)

$\Rightarrow$ Practical implementations will come in the next weeks.

Enjoy the Easter break!

References

Boumans, J. W., & Trilling, D. (2016). Taking stock of the toolkit: An overview of relevant autmated content analysis approaches and techniques for digital journalism scholars. *Digital Journalism*, *4*(1), 8–23. https://doi.org/10.1080/21670811.2015.1096598