Beyond Counting Words: Working with Word Embeddings

Damian Trilling

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

12–13 April 2021

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

This part: Machine Learning in Python

Machine learning for textual data

From text to feature: count vectorizers and tf-idf vectorizers

Classical Machine Learning

- Zooming in on supervised ML
- You have done it before!
- From regression to classification
- Our first machine learning model in scikit-learn

Machine learning for 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

Some considerations

- Both can have a place in your workflow (e.g., bottom-up as first exploratory step)
- You have a clear theoretical expectation? Bottom-up makes little sense.
- But in any case: you need to transform your text into something "countable".

Some considerations

- Both can have a place in your workflow (e.g., bottom-up as first exploratory step)
- You have a clear theoretical expectation? Bottom-up makes little sense.
- But in any case: you need to transform your text into something "countable".

Some considerations

- Both can have a place in your workflow (e.g., bottom-up as first exploratory step)
- You have a clear theoretical expectation? Bottom-up makes little sense.
- But in any case: you need to transform your text into something "countable".

There are two main libraries: scikit-learn and gensim.

scikit-learn is *the* module for almost all “classic” machine learning tasks.

gensim is a specialized module for topic models and embeddings

[show https://scikit-learn.org]

[show https://radimrehurek.com/gensim/ and https://github.com/RaRe-Technologies/gensim]

From text to feature: count vectorizers and tf-idf vectorizers

What is a vectorizer

- Transforms a list of texts into a sparse (!) matrix (of word frequencies)
- Vectorizer needs to be “fitted” to the training data (learn which words (features) exist in the dataset and assign them to columns in the matrix)
- Vectorizer can then be re-used to transform other datasets

What is a vectorizer

- Transforms a list of texts into a sparse (!) matrix (of word frequencies)
- Vectorizer needs to be "fitted" to the training data (learn which words (features) exist in the dataset and assign them to columns in the matrix)
- Vectorizer can then be re-used to transform other datasets

What is a vectorizer

- Transforms a list of texts into a sparse (!) matrix (of word frequencies)
- Vectorizer needs to be “fitted” to the training data (learn which words (features) exist in the dataset and assign them to columns in the matrix)
- Vectorizer can then be re-used to transform other datasets

Different vectorizers

1. CountVectorizer (=simple word counts)
2. TfidfVectorizer (word counts ("term frequency") weighted by number of documents in which the word occurs at all ("inverse document frequency"))

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t$$

There are different ways to weigh the idf score. A common one is taking the logarithm:

$$idf_t = \log \frac{N}{n_t}$$

where N is the total number of documents and n_t is the number of documents containing term t

Different vectorizers

1. CountVectorizer (=simple word counts)
2. TfidfVectorizer (word counts ("term frequency") weighted by number of documents in which the word occurs at all ("inverse document frequency"))

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t$$

There are different ways to weigh the idf score. A common one is taking the logarithm:

$$idf_t = \log \frac{N}{n_t}$$

where N is the total number of documents and n_t is the number of documents containing term t

Different vectorizers

1. CountVectorizer (=simple word counts)
2. TfidfVectorizer (word counts (“term frequency”) weighted by number of documents in which the word occurs at all (“inverse document frequency”))

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t$$

There are different ways to weigh the idf score. A common one is taking the logarithm:

$$idf_t = \log \frac{N}{n_t}$$

where N is the total number of documents and n_t is the number of documents containing term t

Different vectorizers

1. CountVectorizer (=simple word counts)
2. TfidfVectorizer (word counts ("term frequency") weighted by number of documents in which the word occurs at all ("inverse document frequency"))

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t$$

There are different ways to weigh the idf score. A common one is taking the logarithm:

$$idf_t = \log \frac{N}{n_t}$$

where N is the total number of documents and n_t is the number of documents containing term t

Different vectorizer options

- Preprocessing (e.g., stopword removal)
- Remove words below a specific threshold ("occurring in less than $n = 5$ documents") $\Rightarrow$ spelling mistakes etc.
- Remove words above a specific threshold ("occuring in more than 50% of all documents) $\Rightarrow$ de-facto stopwords
- Not only to improve prediction, but also performance (can reduce number of features by a huge amount)

TheTdidfVecotrizer

A small sidenote

If you calculate tf·idf scores by hand, you will see that they differ from what scikit-learn reports.

First, scikit-learn adds 1 to both N and n_t to avoid divisions by zero and taking the logarithm of zero:

$$idf_t = \log \frac{N+1}{n_t+1}$$

Second, the scores that scikit-learn reports are *normalized* using the he Eucledian norm.

For more info, see
https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

Using a scikit-learn vectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
texts = [’This is the first text text text first’, ’And another text
    yeah yeah’]
vec = CountVectorizer(texts)
vec.fit_transform(texts)

# if we want to see what it looks like
# DON’T DO THIS WITH LARGE MATRICES!
print(vec.get_feature_names())
print(vec.transform(texts).todense())
```

[show in notebook]

Before we can do machine learning, we need to make features

- typically, (weighted) word frequencies (count vs tf·idf)
- normalization steps first (lowercasing, punctuation, (stemming/lemmatizing))
- potentially also other feature (e.g., named entities – or only specific word types)
- unigrams vs ngrams
- pruning (removing extremes)

Before we can do machine learning, we need to make features

- typically, (weighted) word frequencies (count vs tf·idf)
- normalization steps first (lowercasing, punctuation, (stemming/lemmatizing))
- potentially also other feature (e.g., named entities – or only specific word types)
- unigrams vs ngrams
- pruning (removing extremes)

Before we can do machine learning, we need to make features

- typically, (weighted) word frequencies (count vs tf·idf)
- normalization steps first (lowercasing, punctuation, (stemming/lemmatizing))
- potentially also other feature (e.g., named entities – or only specific word types)
- unigrams vs ngrams
- pruning (removing extremes)

Before we can do machine learning, we need to make features

- typically, (weighted) word frequencies (count vs tf·idf)
- normalization steps first (lowercasing, punctuation, (stemming/lemmatizing))
- potentially also other feature (e.g., named entities – or only specific word types)
- unigrams vs ngrams
- pruning (removing extremes)

Before we can do machine learning, we need to make features

- typically, (weighted) word frequencies (count vs tf·idf)
- normalization steps first (lowercasing, punctuation, (stemming/lemmatizing))
- potentially also other feature (e.g., named entities – or only specific word types)
- unigrams vs ngrams
- pruning (removing extremes)

Classical Machine Learning

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)

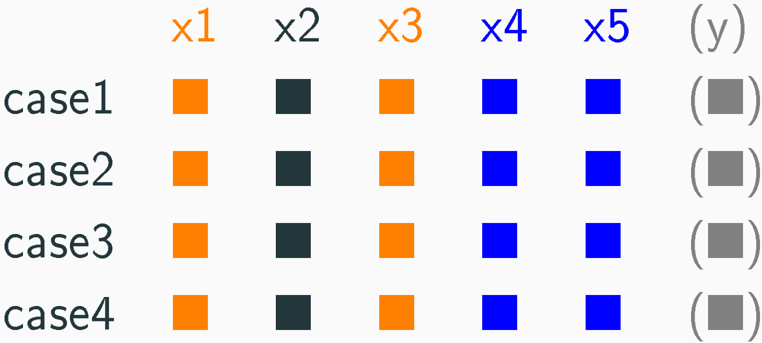
Again, you already know some techniques to find out how x1, x2,...x_i **co-occur from other courses:**

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

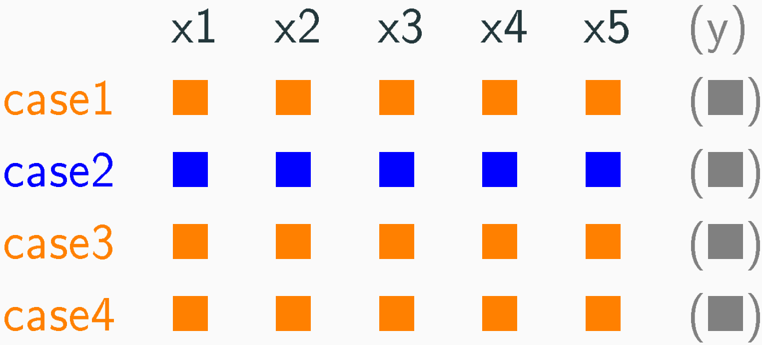
Let's distinguish four use cases...

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	■	■	■	■	■	■



Dimensionality reduction: finding similar variables (features)



Clustering: finding similar cases

	x1	x2	x3	x4	x5		y
case1	■	■	■	■	■	$\rightarrow$	■
case2	■	■	■	■	■	$\rightarrow$	■
case3	■	■	■	■	■	$\rightarrow$	■
case4	■	■	■	■	■	$\rightarrow$	■
new case	■	■	■	■	■	$\rightarrow$	?

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	■	■	■	■	■	■

Classical Machine Learning

Zooming in on supervised ML

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*: $\hat{y}_{man20} = -.8 + .4 \times 1 + .08 \times 20 = 1.2$ $\hat{y}_{woman40} = -.8 + .4 \times 0 + .08 \times 40 = 2.4$

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×idf) (⇒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)

Classical Machine Learning

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.

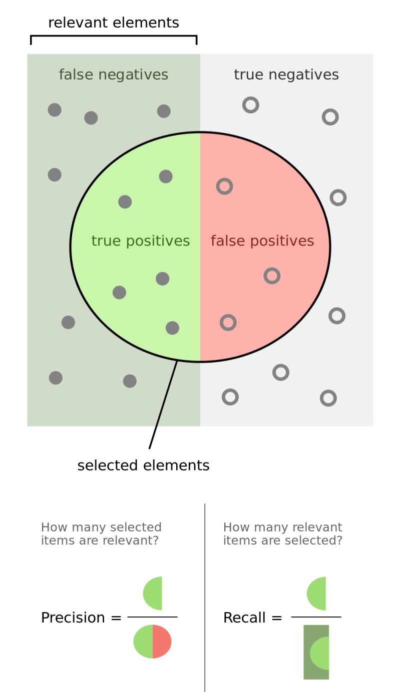
(quite confusingly, even if we use a logistic regression for the latter)

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
- $F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$
- 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:

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

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

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:

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

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:

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

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

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

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

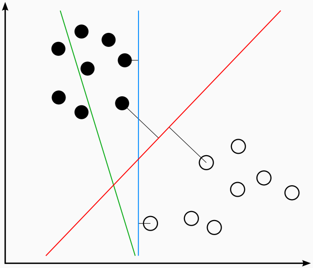
- The features are *not* independent.
- Computationally more expensive than Naïve Bayes
- We can get probabilities instead of just a label
- That allows us to say how sure we are for a specific case
- . . . or to change the threshold to change our precision/recall-tradeoff

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

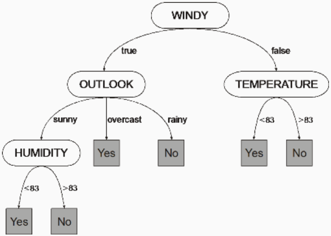
(Further reading: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/)

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

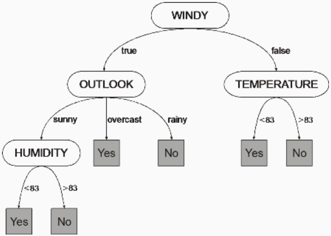
- Model problem as a series of decisions (e.g., if cloudy then . . . if temperature > 30 degrees then . . .)
- Order and cutoff-points are determined by an algorithm
- Big advantage: Model non-linear relationships
- And: They are easy to interpret (!) ("white box")



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

Decision Trees and Random Forests

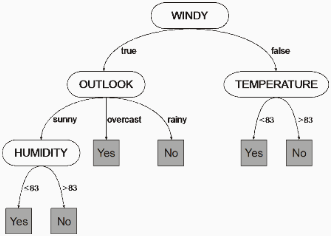
- Model problem as a series of decisions (e.g., if cloudy then . . . if temperature > 30 degrees then . . .)
- Order and cutoff-points are determined by an algorithm
- Big advantage: Model non-linear relationships
- And: They are easy to interpret (!) ("white box")



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

Decision Trees and Random Forests

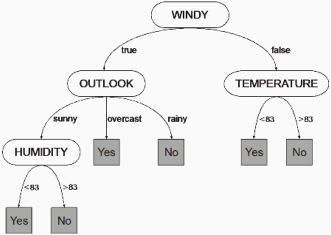
- Model problem as a series of decisions (e.g., if cloudy then . . . if temperature > 30 degrees then . . .)
- Order and cutoff-points are determined by an algorithm
- Big advantage: Model non-linear relationships
- And: They are easy to interpret (!) ("white box")



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 ⇒ might be a good choice for your non-linear classification problem

Classical Machine Learning

Our first machine learning model in scikit-learn

[go to notebook, show scikit-learn]

[in case people are interested, show gensim for LDA (even though a bit off-topic)]

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