Text Classification and Sentiment Analysis

Berit Janssen

Utrecht University
Digital Humanities Lab

Goals

- Classification
  - Preprocessing, training
  - Binary classification
  - Multiclass classification
  - Optimizing classifiers
- Sentiment analysis
  - Lexicon-based sentiment analysis
  - Supervised learning

Classification

Examples for classification in text mining

- Newspaper articles which contain a specific topic
- Author recognition
- Positive or negative opinions in a book review

Machine learning terminology

- *Features*: information which is used to separate data into classes
- In text classification: commonly words / tokens / ngrams
- *Prediction:* features are used to predict labels of the data, which may be compared to known correct labels (“*ground truth*”)

Supervised vs. unsupervised learning

- Supervised learning: learning from labeled data
- Unsupervised learning: find structure in unlabeled data (e.g. clustering texts together based on a similarity metric)

Choices for text classification

- Which features to use?
  - Words (unigrams)
  - Phrases/n-grams
  - Sentences
- How to interpret features?
  - Bag of words
  - Annotated lexicons
  - Syntactic patterns
  - Paragraph structure

Preparing data

- What steps for cleaning and preparing data can you remember?

Preparing data

- Tokenize text, i.e., split it into words
- Remove stop words
- Remove names
- Remove numbers
- Lemmatize text, i.e., “runs” and “run” become one term
- Stem text, i.e., “running” and “runner” become one term
- Document-term matrix: count how often each term occurs in each document

Corpus of book reviews

- Digital Opinions on Translated Literature (DIOPTRA-L)
- Book reviews from Goodreads
  - review text
  - author, title
  - star ratings
  - book edition
  - book genre
  - age category
- Available at https://ianalyzer.hum.uu.nl/

Example data from DIOPTRA-L

text	language	author	author_gender	age_category	book_genre	rating_no	tokenised_text
In a post-Atomic War world three large states ...	English	Joseph Sparrow	male	Adult	Literary fiction	4.0	post atomic war world large state emerge story...
1984 is not a book I would choose myself, beca...	English	Lysanne	female	Adult	Literary fiction	1.0	book choose dystopia theme like kind story lik...
4.5. Woooow, es la primera	Spanish	L. C. Julia	unknown	Adult	Literary fiction	4.0	distopía ganar estrellar y jajaja

Preparing data: document-term matrix

```
[17] from sklearn.feature_extraction.text import CountVectorizer
     vectorizer = CountVectorizer()
     X = vectorizer.fit_transform(data['text'])
     y = data['age_category']
```

```
words = vectorizer.get_feature_names()
print(len(words), words[26665:26942])
print(X)
```

```
30005 ['the', 'thea', 'theaccidentalbookclub', 'theater', 'theaters', 'theatre', 'theatres',
  (0, 303)	1
  (0, 21714)	1
  (0, 21731)	1
  (0, 22451)	1
  (0, 26768)	4
  (0, 10892)	4
  (0, 671)	2
  (0, 26665)	28
  (0, 503)	1
  (0, 26941)	2
  (0, 18965)	1
  (0, 16036)	1
  (0, 29491)	2
```

Preparing data: document-term matrix alternatives

- Alternatively, the document-term matrix can also be weighed with Tf-Idf:
`sklearn.feature_extraction.text.TfIdfVectorizer`

- You can pass the parameter `ngram_count` to the CountVectorizer count combinations of words:

```
CountVectorizer(ngram_range=(1,2))
```

Training a classifier

- Training procedure minimizes prediction error in training data
- Accuracy: percentage of correct labels
- Precision / Recall / F1 in binary classification
- Problem: overfitting

Avoiding overfitting

- Splitting data into training set / test set
- Validation: find the most successful settings for classifiers (e.g., smoothing parameters) on a validation set
- Test classifier on test set (unseen data)
- Accuracy: percentage of correct labels

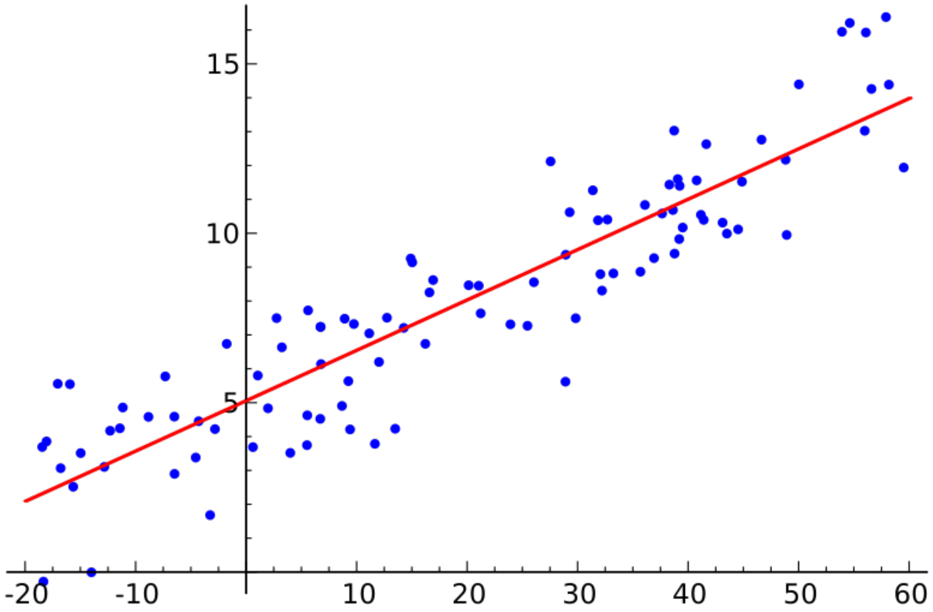
Training a classifier

```
[23] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression(max_iter=300)
model = logistic.fit(X_train, y_train)
model.score(X_test, y_test)
```

How are classifiers trained?

- Datapoints are randomly assigned to classes
- Error term is calculated (classes wrongly assigned)
- Iteratively re-assign classes and calculate error term
- *Convergence* to a minimum error



Source: Wikimedia

Binary classification

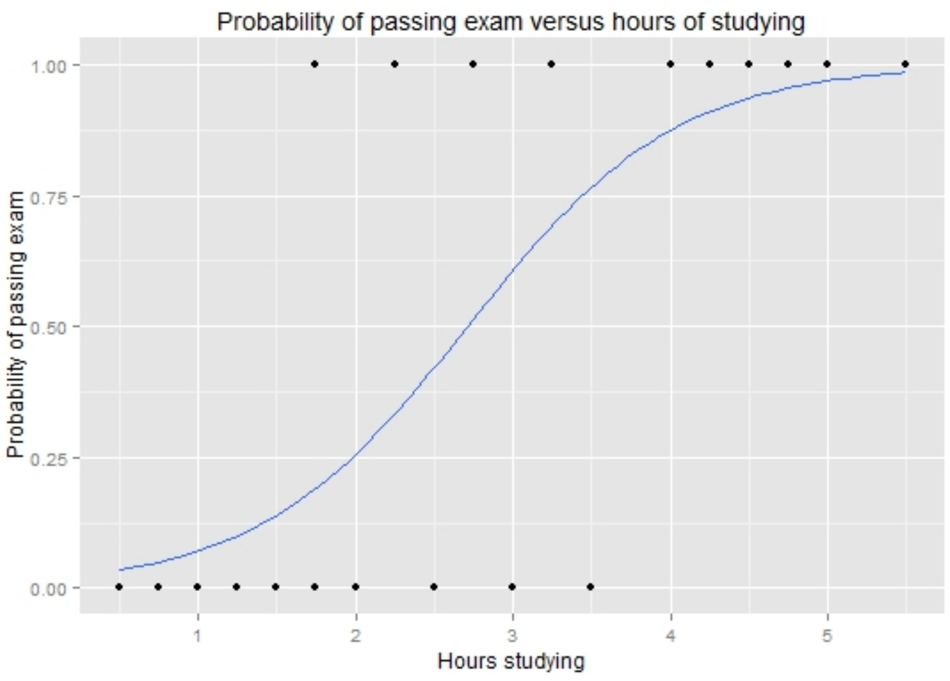
Binary classification

- Can we predict, based on review text, whether the reviewer discusses literature for children or adults?

```
y = data['age_category']
```

Logistic regression

- Find a curve that separates one class from the other
- Words are features whose weights are optimized during training
- Parameter C sets the amount of *regularization*: smaller values of C help to avoid overfitting



Source: Wikimedia

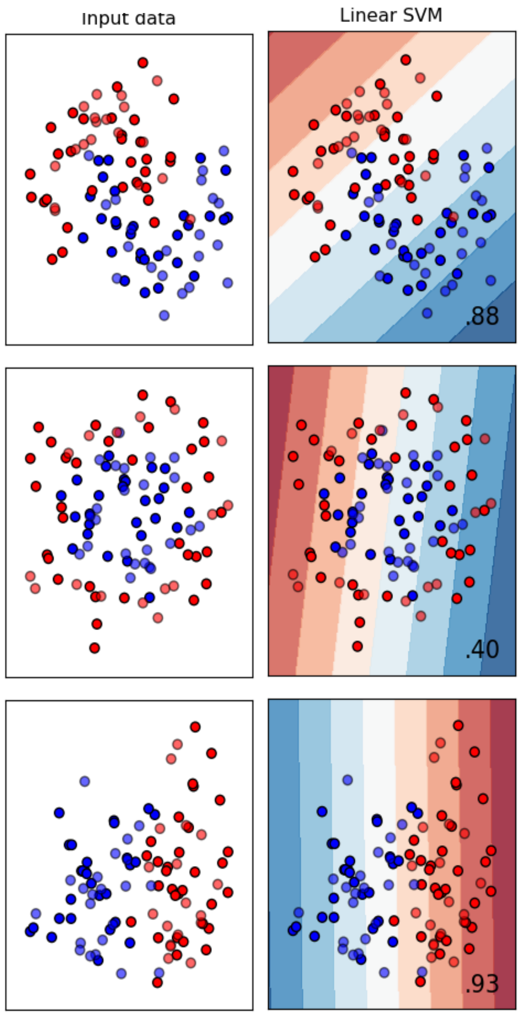
Logistic regression

```
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression(max_iter=300)
model = logistic.fit(X_train, y_train)
model.score(X_test, y_test)
```

```
0.8560606060606061
```

Support vector machine classifier (SVM)

- Relationships between texts are mapped to higher dimensionality (e.g., by considering two words together as another dimension)
- Find a plane in that higher-dimensional space which separate texts of different labels



https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Support vector machine classifier (SVM)

```
from sklearn.svm import LinearSVC
svm = LinearSVC(C=1.0)
model = svm.fit(X_train, y_train)

svm = LinearSVC(C=0.1)
model2 = svm.fit(X_train, y_train)
print('accuracy with default regularization:', model.score(X_test, y_test),
      '\naccuracy with more regularization: ', model2.score(X_test, y_test))
```

```
accuracy with default regularization: 0.8360606060606061
accuracy with more regularization:  0.8554545454545455
```

Multi-class classification

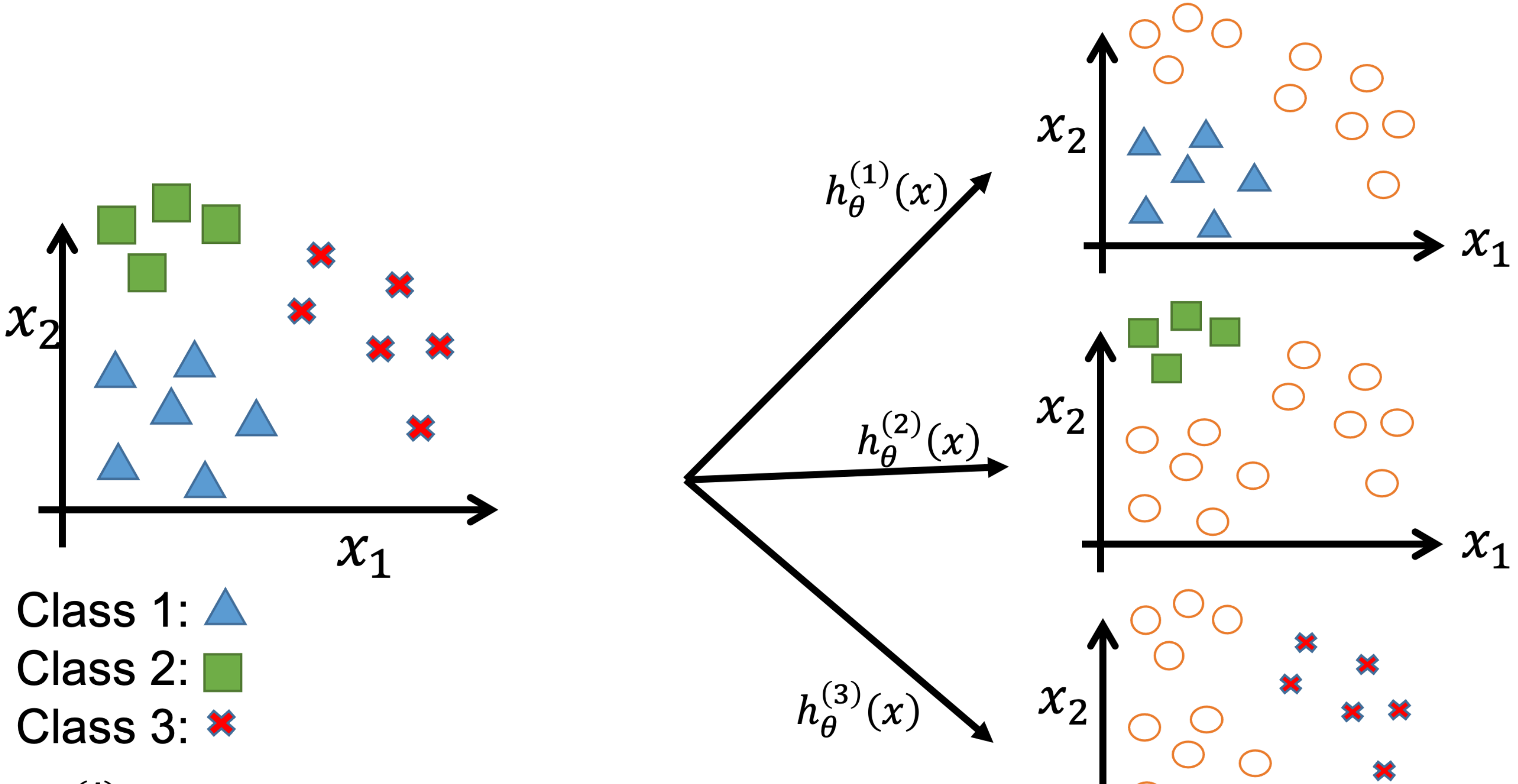
Multi-class classification

- Can we predict, based on review text, which genre the reviewer discusses?

```
[13] y = data['book_genre']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

One-vs-all / one-vs-rest

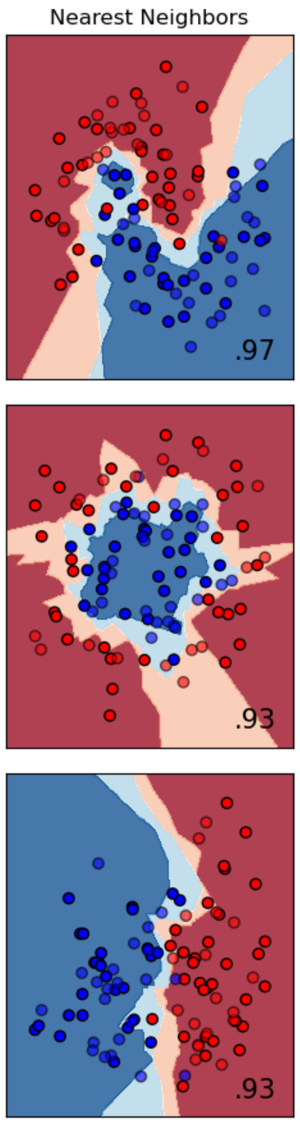


$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \quad (i = 1, 2, 3)$$

Slide credit: Andrew Ng

K-nearest neighbor classifier

- Texts are considered close neighbours if they share many words
- Give a text the same label as the majority of its nearest neighbours
- More considered neighbours (k) lead to higher granularity of the prediction
- Higher k may cause overfitting
- Can be set with `n_neighbors` in sklearn



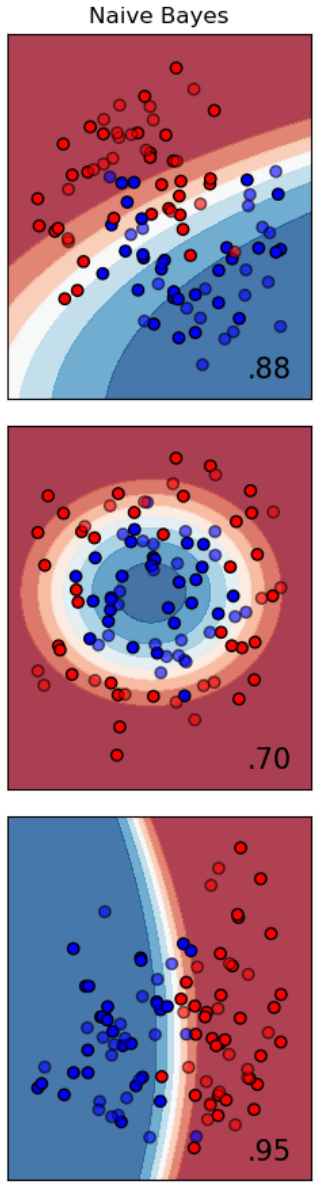
K-nearest neighbour classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
model = knn.fit(X_train, y_train)

knn = KNeighborsClassifier(n_neighbors=10)
model2 = knn.fit(X_train, y_train)

knn = KNeighborsClassifier(n_neighbors=100)
model3 = knn.fit(X_train, y_train)
print('accuracy with 3 neighbours:', model.score(X_test, y_test),
      '\naccuracy with 10 neighbours:', model2.score(X_test, y_test),
      '\naccuracy with 100 neighbours:', model3.score(X_test, y_test))
```

```
accuracy with 3 neighbours: 0.4557575757575757575
accuracy with 10 neighbours: 0.4972727272727272726
accuracy with 100 neighbours: 0.4918181818181818
```

Naive Bayes classifier

- Calculate probabilities of different labels for each text, based on words in the text
- Problem: zero counts (word / label combinations which do not occur in the training data)
- Addressed with Laplace smoothing (add a fixed number to all counts)
- In sklearn can be set with `alpha` (positive number or 0 for no smoothing)



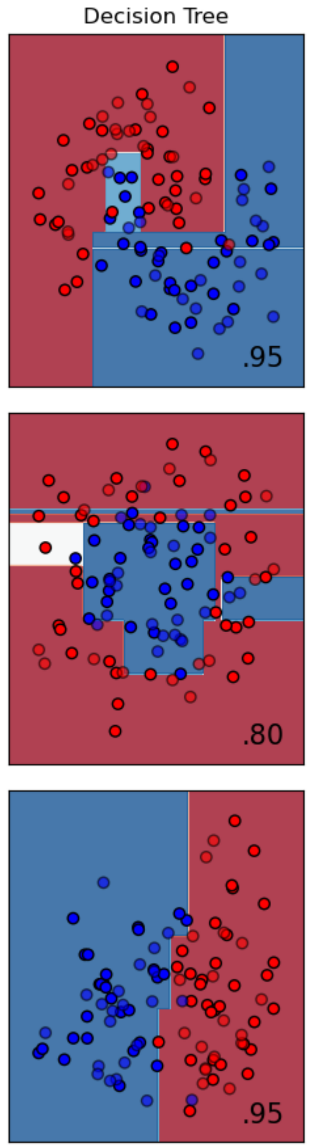
Naive Bayes classifier

```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB(alpha=1)
model = nb.fit(X_train, y_train)

nb = MultinomialNB(alpha=10)
model2 = nb.fit(X_train, y_train)
print('accuracy with alpha=1:', model.score(X_test, y_test),
      '\naccuracy with alpha=10:', model2.score(X_test, y_test))
```

```
accuracy with alpha=1: 0.673030303030303
accuracy with alpha=10: 0.6136363636363636
```

Decision tree classifier

- Word features are used to separate classes (e.g., if this document contains “sorcerer”, it is popular fiction (fantasy))
- Control how many levels the tree has: more levels means higher granularity
- More levels may cause overfitting
- Can be set through `max_depth` in sklearn



Decision tree classifier

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=5)
model = tree.fit(X_train, y_train)

tree = DecisionTreeClassifier(max_depth=None)
model2 = tree.fit(X_train, y_train)
print('accuracy with maximum tree depth 5:', model.score(X_test, y_test),
      '\naccuracy with unlimited tree depth:', model2.score(X_test, y_test))
```

```
accuracy with maximum tree depth 5: 0.5712121212121212
accuracy with unlimited tree depth: 0.5418181818181819
```

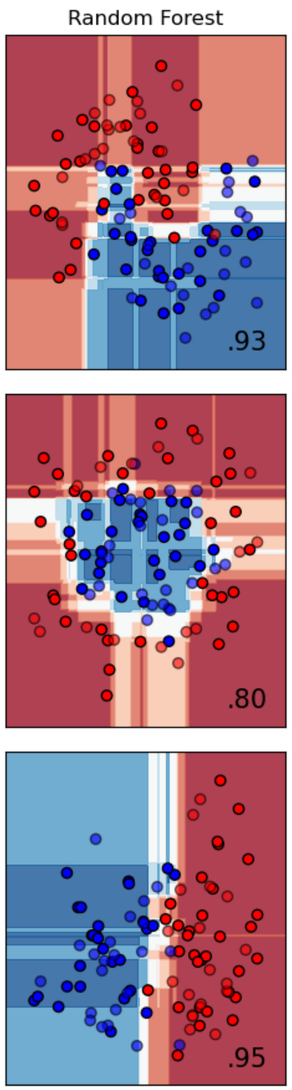
Optimizing classifiers

Ensemble classifiers

- Chain classifiers (use output predictions of one classifier as input features for another)
- Use multiple classifiers and combine their output

Random forest classifier

- Fits multiple decision trees on subsets of the data
- Averages over the individual trees’ predictions
- Can control how many decision trees are used
- More trees means more computation time, avoids overfitting
- Control number of decision trees as `n_estimators` in sklearn
- Can also set parameters (`max_depth`) of the decision trees



Random forest classifier

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=3)
model = rfc.fit(X_train, y_train)

rfc = RandomForestClassifier(n_estimators=20)
model2 = rfc.fit(X_train, y_train)
print('accuracy with 3 trees:', model.score(X_test, y_test),
      '\naccuracy with 20 trees:', model2.score(X_test, y_test))
```

```
accuracy with 3 trees: 0.5227272727272727
accuracy with 20 trees: 0.6203030303030304
```

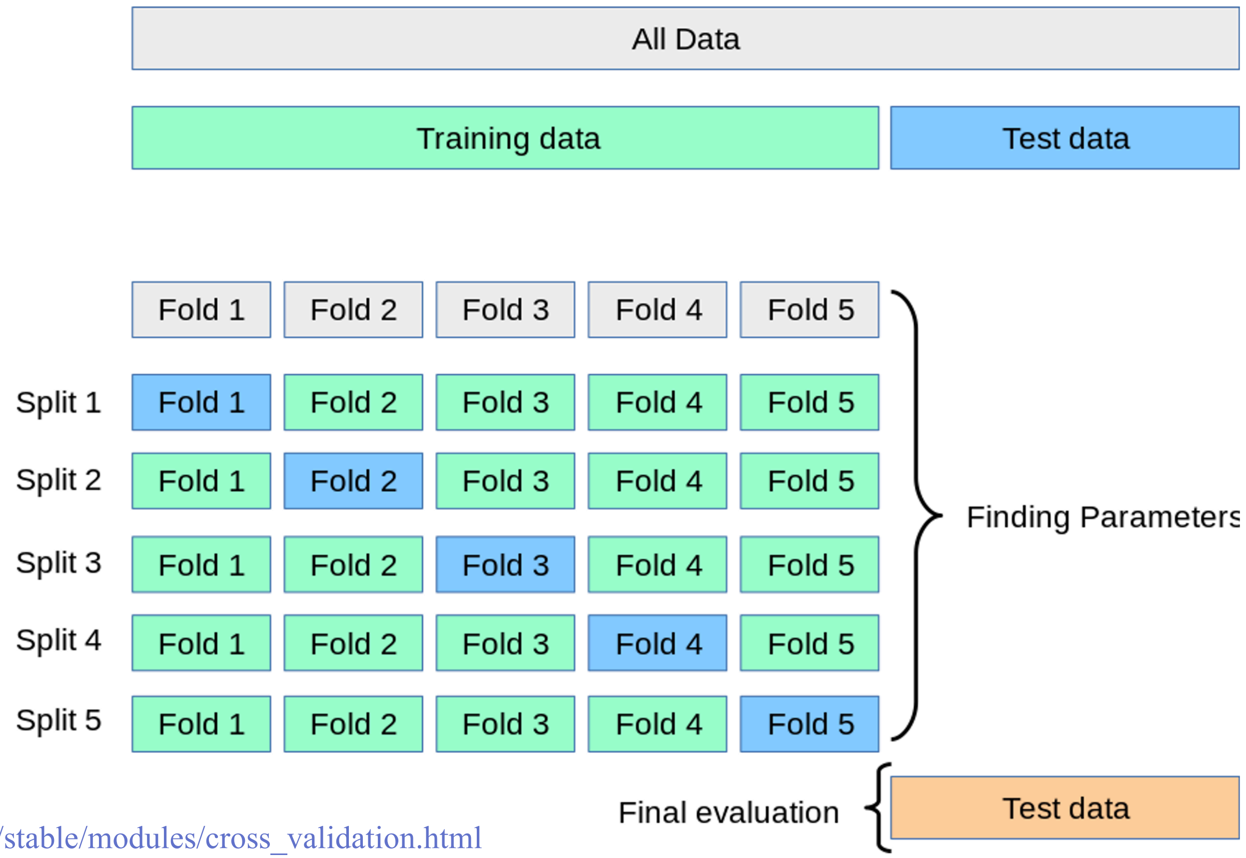
Voting classifier

```
from sklearn.ensemble import VotingClassifier

vc = VotingClassifier(estimators=[('knn', knn), ('nb', nb), ('svm', svm), ('tree', tree)])
vc.fit(X_train, y_train)
vc.score(X_test, y_test)
```

Classifier hyperparameters

- Hyperparameters are classifier parameters
- Example: `n_neighbors` of the K-Nearest Neighbor classifier
- Defaults from sklearn can be used as starting points
- Grid search: optimization procedure to find the values for highest accuracy in a range of values (e.g., from 20 to 100 neighbours)

Avoid overfitting parameters: cross-validation



https://scikit-learn.org/stable/modules/cross_validation.html

Grid search

```
[ ]  from sklearn.model_selection import GridSearchCV
     # set the search space for grid search. In this case, between 2 and 20 nearest neighbors
     parameters = {'n_neighbors': [2,20]}
     knn = KNeighborsClassifier()
     search = GridSearchCV(knn, parameters)
     search.fit(X_train, y_train)
     # the best score achieved
     print(search.score(X_test, y_test))
     # get_params() gives the parameters leading to this best score (in 'estimator')
     search.get_params()
```

```
0.2833333333333333
{'cv': None,
 'error_score': nan,
 'estimator': KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform'),
```

Feature importance

- How much does each word (feature) contribute to classification success?
- Example: decision trees
- model.coefs_ or model.feature_importances_

```
features = vectorizer.get_feature_names()
coefs = model.feature_importances_
zipped = zip(features, coefs)
df = pd.DataFrame(zipped, columns=["feature", "value"])
df = df.sort_values("value", ascending=False)
df.head(10)
```

	feature	value
15482	series	0.481071
2502	caterpillar	0.158612
4552	diary	0.087997
11767	novel	0.075103
6356	fantasy	0.056831
9636	kid	0.024256
9414	italian	0.018527
19366	woman	0.017625
17955	twist	0.014370

Sentiment analysis

Sentiment analysis

- Does this text…
  - recommend a telephone?
  - express a positive opinion of nuclear energy?
  - tear a movie to bits?
  - tell you to read *Le Petit Prince*?

Sentiment analysis – AKA

- Sentiment mining
- Opinion mining
- Subjectivity analysis

Opinion as a quintuple

- *entity, aspect, sentiment, holder, time*:
  - ***entity***: target entity (or object)
  - ***aspect***: aspect (or feature) of the entity
  - ***sentiment***: +, -, or neu, a rating, or an emotion
  - ***holder***: opinion holder
  - ***time***: time when the opinion was expressed

Sentiment analysis objectives

- Simplest task:
  - Is the attitude of this text positive or negative?
- More complex:
  - Rank the attitude of this text from 1 to 5
- Advanced:
  - Detect the target, source, or complex opinion types
  - Implicit opinions or aspects

Challenges in sentiment analysis

- What problems can you think of which might make it difficult to distill opinions from text?

Challenges in sentiment analysis

- Subtlety of sentiment expression
  - irony
  - expression of sentiment using neutral words
- Domain/context dependence
  - words/phrases can mean different things in different contexts and domains
- Effect of syntax on semantics
- Sentiments in other languages than English

Lexicon-based vs. machine learning approaches

- Lexicon-based methods: words which are associated with positive or negative sentiment
- Supervised learning:
  - training classifiers
  - deep learning

Lexicon-based approaches

LWIC (Linguistic Enquiry and Word Count)

- http://liwc.wpengine.com/
- 2300 words, more than 70 classes
- **Affective Processes**
  - negative emotion (bad, weird, hate, problem, tough)
  - positive emotion (love, nice, sweet)
- **Cognitive Processes**
  - tentative (maybe, perhaps, guess),
  - inhibition (block, constraint)
- **Pronouns, Negation** (*no, never*)
- **Quantifiers** (*few, many*)

Bing Liu opinion lexicon

- Bing Liu's Page on Opinion Mining
- http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar
- 6786 words
  - 2006 positive
  - 4783 negative

SentiWordNet

- https://github.com/aesuli/SentiWordNet
- All WordNet synsets automatically annotated for degrees of positivity, negativity, and neutrality/objectiveness
- [estimable(J,3)] “may be computed or estimated”

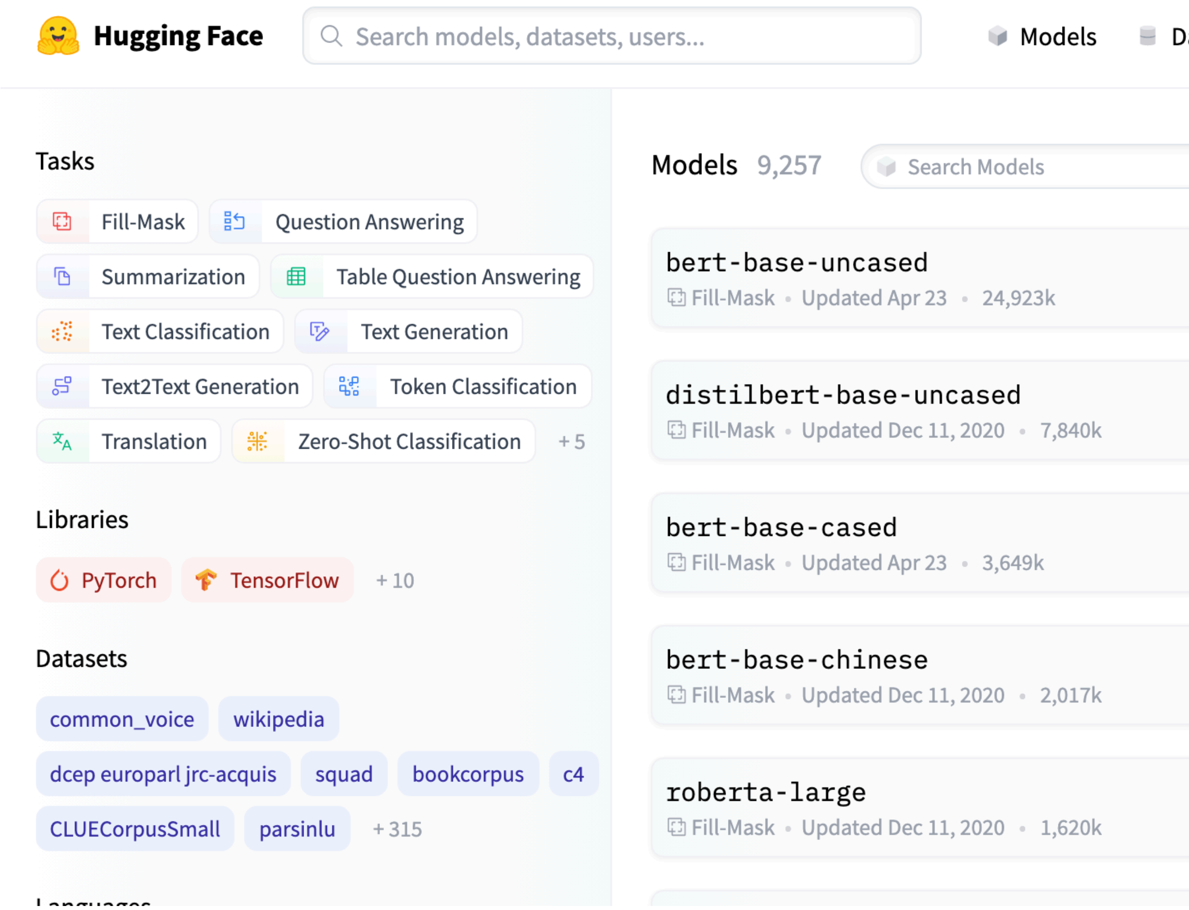
  Pos 0 Neg 0 Obj 1
- [estimable(J,1)] “deserving of respect or high regard”

  Pos .75 Neg 0 Obj .25

Supervised approaches

Sentiment analysis with deep learning

- Available through Python transformer library
- BERT (Bidirectional Encoder Representations from Transformers)
- https://huggingface.co/ collects pre-trained models for various purposes



Sentiment analysis with a model trained on product reviews

```
from transformers import pipeline
classifier = pipeline('sentiment-analysis', model="nlptown/bert-base-multilingual-uncased-sentiment")

out_df = pd.DataFrame()
for i, row in data.head(100).iterrows():
  prediction = classifier(row['tokenised_text'][:512])
  label = int(prediction[0]['label'].split(' ')[0])
  df = pd.DataFrame({'predicted_rating': [label],
                     'star_rating': [int(row['rating_no'])]})
  out_df = out_df.append(df, ignore_index=True)
```

```
[ ] print('percentage correct predictions:', len(out_df[out_df['predicted_rating']==out_df['star_rating']])/100)
```

```
percentage correct predictions: 0.74
```

Conclusion

Summary

- Text classification:
  - Features and prediction
  - Training / test set
  - Binary classification: logistic regression and support vector machines
  - Multiclass classification: K-nearest neighbor, Naive Bayes, Decision trees
  - Optimizing classifiers: ensemble classifiers, hyperparameters, feature importance

Summary

- Sentiment analysis:
  - Challenges
  - Lexicon-based approaches: LWIC, Bing Liu opinion lexicon, SentiWordNet
  - Supervised approaches: training classifiers, deep learning (e.g., with transformers library)

Practical 2

- We will train our own sentiment analysis classifier, using the following steps:
  - Build a document-term matrix (`CountVectorizer` or `TfIdfVectorizer`)
  - Splitting into training and test data (`train_test_split`)
  - Train classifiers
    - Initialize classifier with parameters
    - `model = classifier.fit(X_train, y_train)`
- Measure performance on test set
    - `model.score(X_test, y_test)`