many document have associated metadate in the form of classes or labels (categorical variables)

approximate a function that can map documents features onto class information

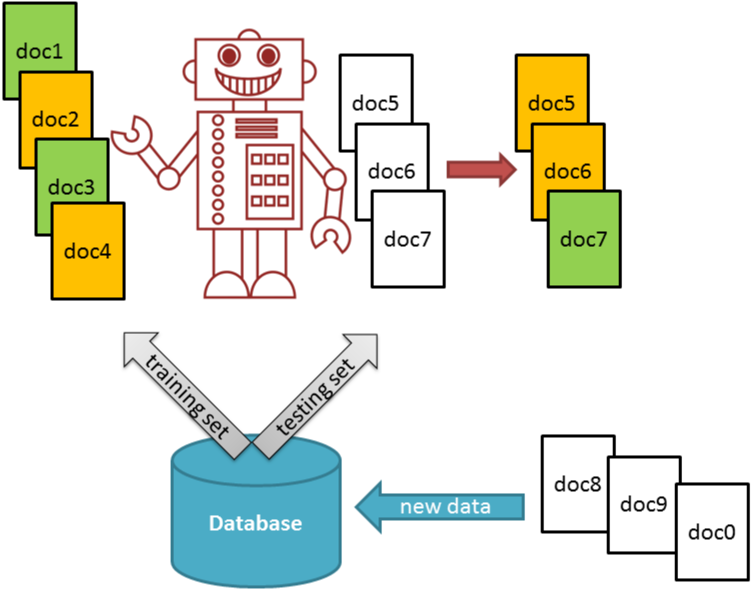
preferably, our model should be the best performing classifier in the set of possible classifiers given labeled data (supervised learning), a classification algorithm will output a solution that categorizes new examples $\rightarrow$ associate labels with subsets of the data

while clustering (unsupervised learning) searches for groups within the corpus, classification learns to map a collection of documents onto a categorical class values or labels $\rightarrow$ find mapping function

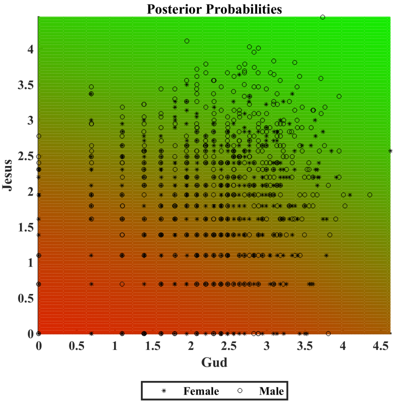
data (features) with class values ($\sim$ labeled data), excellent opportunity to make use of metadata

vast majority of models are **black box models**

workflow: separate data set in training and test subsets (training, test, and validation) $\rightarrow$ train model $\rightarrow$ test model $\rightarrow$ apply model to new data


doc1
doc2
doc3
doc4
doc5
doc6
doc7
doc5
doc6
doc7
training set
testing set
Database
new data
doc8
doc9
doc0

classification in the humanities



binary and multiclass classification problems [1]

naive bayes probabilistic classifier that is fast and popular for in text categorization, but assumes independence between features (naive)

neural network broad framework for machine learning, which is very extremely flexible. Training can be very slow, but classification fast. Prone to overfitting

decision tree versatile and creates sets of rules (binary decisions) that are simple and can be understood (leaves are classes and branches features) $\rightarrow$ white box method

support vector machines works on small datasets (typically binary) with high dimensional data (features $>$ objects) and very memory efficient (only uses the support vectors). Bad performance on noisy data (overlapping classes)

[1] Can be advantageous to reformulate multiclass problems as binary

labeled data the correct class information is available

- metadata is readily available, e.g. author, genre, year of publication
- labels from an external source/databases, e.g. reviews, ratings, reads
- annotate data (expert or raters)

evaluate performance (error rate) of a classifier and compare to other classifiers

most metrics are developed for binary classification problems

confusion matrix: table for describing performance of classifier on training and/or testing data

		true	
		positive	negative
predicted	positive	TP	FP
	negative	FN	TN

True Positive correctly assigns positive class membership
True Negative correctly rejects class membership
False Positive fail to rejects class membership
False Negative reject class membership incorrectly

we train a naive bayes classifier on 1500 verses of the kjv bible labeled with collection data (nt: new testament ot: old testament)

Confusion matrix for binary classification problem:

	nt	ot
nt	644	89
ot	106	661

, verses: $644 + 106 + 89 + 661 = 1500$

accuracy measures in how many cases the predicted class conformed with the correct class: $\dfrac{TP + NP}{TP + TN + FP + FN}$

	nt	ot
nt	644	89
ot	106	661

, accuracy: $\dfrac{(644 + 661)}{1500} = 0.87\ (87\%)$

precision measures the number of selected verses that are relevant, i.e., how certain are we that a classified verse is correctly classified ($\sim$ how many time did the model positively predict a class): $\dfrac{TP}{TP + FP}$

	nt	ot
nt	644	89
ot	106	661

, $precision_{NT}$: $\dfrac{(644)}{644 + 89} = 0.88$

for each class label: How many of the items that got the label should have gotten it? How many should have gotten other labels?

recall measures the number of relevant verses that are selected, i.e., how good is the classifier at detecting verses within a given class: $\frac{TP}{TP+FN}$

	NT	OT
NT	644	89
OT	106	661

, $recall_{NT}$: $\frac{(644)}{644+106} = 0.86$

For each class label: How many items that should have gotten the label did get it? How many were missed?

F-score composite (general) measure of a classifier's accuracy

$$F_1 = 2 \times \frac{percision \times recall}{precision + recall}$$

$$F_1 : 2 \times \frac{.88 \times .86}{.88 + .86} = 0.87$$

F is the harmonic mean of precision and recall.

if a model is sufficiently complex and gets enough data, it can basically memorize the data set (overfitting) $\rightarrow$ need to test the model on held-out data

validation when building a predictive model, we need a way to evaluate the capability of the model on unseen data

- data Split (conventional validation)
- cross validation
- bootstrap

classification with scikit-learn

```
datapath = '/home/kln/corpora/kjv_books'
docs = vanilla_folder(datapath)

import pandas as pd
import numpy as np
metadata = pd.read_csv('/home/kln/corpora/kjv_metadata.csv')
class_id = metadata['class'].tolist()
class_u, class_int = np.unique(class_id, return_inverse = True)

from sklearn.feature_extraction.text import CountVectorizer
countvect = CountVectorizer()
vectspc = countvect.fit_transform(docs)
vectspc.shape

# index value of a word in the vocabulary
countvect.vocabulary_.get(u'god')
countvect.vocabulary_.get(u'woman')

# build vector space model
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
vectspc_tfidf = tfidf_transformer.fit_transform(vectspc)
vectspc_tfidf.shape

# train naive bayes classfier
from sklearn.naive_bayes import MultinomialNB
nb_class = MultinomialNB().fit(vectspc_tfidf, class_id)
# classifier training performance
predicted = nb_class.predict(vectspc_tfidf)
np.mean(predicted == class_id)

# svm for comparison
from sklearn.linear_model import SGDClassifier
svm_class = SGDClassifier(loss='hinge', penalty='l2',alpha=1e-3, n_iter=5,
        random_state=42).fit(vectspc_tfidf, class_id)
predicted = svm_class.predict(vectspc_tfidf)
np.mean(predicted == class_id)
```

classification with RTextTools

```
library(RTextTools)
## separate training and testing set and create a container
# random sample for testing  data from data set
trainidx.v <- 1:nrow(text.dtm)
testidx.v <- sort(sample(trainidx.v, nrow(text.dtm)*.1, replace = FALSE, prob = NULL))
trainidx.v <- sort(trainidx.v[! trainidx.v%in%testidx.v])

# change object type, create_analytics() only handles numeric
classnum.v <- as.numeric(as.factor(class.v))
  # to transform back to original
  factor(classnum.v, labels = unique(class.v))
# create container
container <- create_container(text.dtm, classnum.v, trainSize=trainidx.v,
                              testSize=testidx.v, virgin=FALSE)
# training models
mdl1.l <- train_models(container, algorithms='SVM')
mdl2.l <- train_models(container, algorithms = c('SVM','NNET','TREE') )

# Classifying data
res.df <- classify_models(container, mdl2.l)
head(res.df)
confusion.mat <- as.matrix(table(res.df$SVM_LABEL, container@testing_codes))
rownames(confusion.mat) <- colnames(confusion.mat) <- unique(class.v)
print(confusion.mat)
accuracy <- sum(diag(confusion.mat))/sum(confusion.mat)

# performance metrics
analytics <- create_analytics(container, res.df)
class(analytics)
summary(analytics)
```