Lecture 1 1/7/202

Lecture 1

Data Prep & Preprocessing

Missing Data - several methods

Smoothing noisy data

smoothing out fluctuations

1. Binning
   partition using eq. width
   a. use the mean
   b. use the boundaries
2. Clustering
   shows the outliers well
3. Create regression model
   that function you create
   is fed data and you replace
   data points w/ results

Data Integration

from multiple data sources

Redundant - look at correlation

between attributes to see if

could be eliminated / combined

Data Transformation: *

Normalization techniques

1. Min-Max

$$v' = [(v - min)/(max - min)] * (newmax - newmin) + newmin$$

ex. \$30,000 b/w [10,000 .. 45,000] to [0..1]

$[30000 - 10000 / 35000] = 0.57$

2. Z-score

$$v' = (v - \bar{x})/SD$$

3. Decimal scaling

$$v' = v/10^i$$

Good when one attribute dominates magnitude of another (like age vs. salary → normalize both)

Descretization

3 types of attributes:

1. nominal - categorical
2. ordinal
3. numeric

Descretization reduces # of values for a continuous attribute
usually done by dividing range into intervals
can be used to turn # into category
Can also be used to do data reduction
Some algorithms require categorical data only

1. Binning - see slides
2. Histogram analysis
3. Cluster analysis - unsupervised
4. Decision tree - supervised
5. Correlation
for example X^2 analysis
Unsupervised

Binning -
Equal Width - N bins of equal size - outliers cause havoc so do data smoothing or noise reduction first
width = W = (High-Low)/N

Equal Depth
each partition contains
same # of samples

Classification
eg decision tree
use entropy to determine
split points - top down, recursive
split

correlation analysis
supervised
Bottom-up merge
eg χ^2-based discretization

If you need to convert categorical
to numerical attributes
standard spreadsheet format
Ex

outlook	becomes	Outlook = overcast	outlook = sunny	outlook = rain
sunny		0	1	0
sunny		0	1	0
overcast		1	0	0
rain		0	0	1

Data Reduction

kinds

Data cube aggregation

Dimensionality reduction

Discretization

Numerosity reduction - reduce # rows

regression

histograms

Clustering

Sampling

Data cube aggregation

reduce data to the concept level needed in the analysis

Dimension reduction

Curse of dimensionality - as data dimensions increase, data starts to become sparse

Dimensionality reduction

avoid the curse
reduces noise
reduces time / space requirements
easier to see patterns

Techniques for Dimensionality reduction

PCA
attribute subset selection
attribute or feature generation

PCA - find projections of data that captures largest amount of variation in data

Original data projected onto smaller space, resulting in dimensionality reduction

This is done by finding eigenvectors of the covariance matrix, and the e vectors define the new space

Steps for PCA

N rows

n dimensions (attributes)

find K ($\leq n$) orthogonal vectors (ie - principal components) that can best be used to represent your data

1. Normalize the input data
2. Compute K orthonormal (unit) vectors - these are the principal components
3. Each input data (vector) is a linear combo of the K PCV's
4. sort the PC's in decreasing order of strength
   give you order of variance they represent
5. Eliminate those w/ weak components

attribute generation
create new attributes
that capture the important
info more effectively than original
1. attribute extraction
2. mapping to new space (like in
date reduction)
3. attribute construction
a. combine features
b. date discretization

Numerosity reduction - reduce rows
Parametric
regression 3. discard original
data
using least squares method
replace data points w/ the
model
Histogram - keep buckets
Cluster - consider items in same
cluster as same item

Sampling – if normally distributed
not appropriate if skewed
Stratified – data is grouped into
strata & take a sample from
each stratum
can use clustering for
strata

Correlation Analyses
for nominal data use χ^2

$$\chi^2 = \sum \frac{(\text{observed} - \text{expected})^2}{\text{EXPECTED}}$$

larger the χ^2, more likely they
are related
cells that contribute most to
χ^2 are where count is
very different from expected

for numeric data use Correlation Coeff
(Pearsons)

r = correlation
+, − or 0 (not correlated)

correlation is a measure of the linear relationship b/w objects

do you standardize the objects and take their dot product

$a'_k = (a_k - \text{mean}(A)) / \text{STD}(A)$
$b'_k = (b_k - \text{mean}(B)) / \text{STD}(B)$
$\text{Corr}(A,B) = A' \bullet B'$

Cross Tab to look at visual pattern of 2 categorical variables
Can plot w/ Bar charts

2: pf → distribution where, for ex, look at frequency of word, and sort in descending order

2- z score normalization
How many std devs away from mean

Discretization: turns numerical data into categorical

inc_bins = pd.cut(vstable.Income, 3)
↑ # of bins

can specify labels, too and specify the quantiles

groupby - against categorical vars to get aggregated stats

Plotting categorical - Bar
use value_counts
vstable["Genre"].value_counts().plot(kind="Bar")

scatter plot
alpha is transparency
s → size of circles
c → color according to
color map (cmap)

Crosstab
can use group by w/ multiple
variable or
gg = pd.crosstab(cat. variable)
then you can plot it

opposite of discretization (going from
numeric attribute to categorical)
can convert categorical to numeric
using pd.get_dummies
need this to compute similarities
or distances, or correlation
analysis

When data is in all numeric form
said to be in "standard spreadsheet"
format

You also need to normalize
so it is in same scale
- correlation analysis
- distance

Any Function
vstable [vstable.isnull(),
any(axis=1)]
gives me rows where there
is a null
vstable [vstable.Gender,isnull()]
values where just Gender
is null

Fill in missing values

vstable.Age, fillna(age_mean
axis=0, inPlace=True)
↑
look across rows

Distances & Similarity Measures

Proximity measure
distance used in context
of categorical variables

Similarity
often normalized to fall
b/w [0,1]

Euclidian - not normalized
so no upper bound

Grouping similar items requires
measuring distance b/w objects
requires representation of objects
as "feature vectors".

Distance Matrix - distance one row
& another D(3,1) distance b/w row
3 and row 1

Proximity Measure

categorical

based on matching

how many attributes match

m = # matches (color match, shape match)

p = total # of variables

Method 1 → $d(i,j) = \frac{p-m}{p}$

i, j ↑ 2 objects

p ↑ # of attributes under they match

method 2 - convert to std spreadsheet then use vector-based similarity (numeric based)

Binary distance Proximity Measure
2 rows, object i & object j

		obj j	
		1	0
obj i	1	q	r
	0	s	t

so q is # of times i & j match

Distance for symetric binary vars

$\frac{r+s}{q+r+s+t}$ → where different

symetric - care about when both match

asymetric $\frac{r+s}{q+r+s}$

Jaccard $\frac{q}{q+r+s}$

Symetric - care about matches
0s and 1s

Asymetric - maybe zeros don't count
Document & Terms
only care when Terms appears
in several docs, don't
care if it doesn't appear
& doesn't include ?

Jaccard - look at similarity
only
if 2 items have maximum
similarity, J=1

Have to standardize or normalize
data to compare

Common Distance Measures for Numeric Data

Consider 2 rows

$$X = (x_1, x_2 \ldots x_n)$$
$$Y = (y_1, y_2 \ldots y_n)$$

Manhattan

$$dist(X,Y) = |x_1 - y_1| + |x_2 - y_2| \ldots$$

Euclidean

$$dist(X,Y) = \sqrt{(x_1 - y_1)^2 + \ldots (x_n - y_n)^2}$$

Dual (similarity measure)

$$sim(X,Y) = \frac{\sum (x_i \times y_i)}{\sqrt{\sum x_i^2 \times \sum y_i^2}}$$

$$dist(X,Y) = 1 - sim(X,Y)$$

Manhattan - location - blocks this way, 2 blocks that way

Euclidean - straight line distance

Minkowski Distance
the generalization

$$d(i,j) = \sqrt[h]{|x_{i1}-x_{j1}|^h + |x_{i2}-x_{j2}|^h + \ldots |x_{ip}-x_{jp}|^h}$$

Manhattan is when $h=1$ (AKA L_1 Norm)
Euclidean is when $h=2$ (AKA L_2 Norm)

$$\sqrt{|x_{i1}-x_{j1}|^2 + \ldots |x_{ip}-x_{jp}|^2}$$

Both are Distance

Vector-Based similarity - take dot product

$$sim(X,Y) = X \cdot Y = \sum x_i \times y_i$$

scalar value represents how similar they are

Cosine similarity is the normalized dot product

$$sim(X,Y) = \frac{X \cdot Y}{\|X\| \times \|Y\|} =$$

$$\frac{\sum_i (x_i \times y_i)}{\sqrt{\sum_i x^2 \times \sum_i y^2}}$$

↖ norm, magnitude of vector

distance $= 1 - Sim(X,Y)$ SEE Slide 14

You are not concerned w/ magnitude of vector, just the distance

cos 90° = 0 orthogonal
cos 0° = 1 on top of one another
Example slide 17

Document similarity example

	T1,	T2	T3...
Doc1	0	4	0 ...
Doc2	3	1	4 ...
Doc3	0	1	0 ...
:	:	:	:

similarity of D1, D3

1) get dot product (Doc1, Doc3)
2) get Norm doc 1
   get Norm doc 3
3) Cosine Doc1, Doc2 = 10 / Norm1 * Norm2

You could also use correlation as a similarity measure. Good when high mean values across objects

Pearson

$$corr(X,Y) = \frac{cov(X,Y)}{stdev(X) \cdot stdev(Y)}$$

used in recommender systems used to compare rows, objects (before we looked at corr between variables (columns)).

How are distances used in classification

KNN Classify new instance based on similarity to or distance from instances we have seen before

Compare object you want to classify to all other objects

KNN - slide 21

given object X, find k most similar objects to X

find class label for these neighbors & vote for what class X should be in

KNN can also be used for numeric prediction

find value of attribute from neighbors

then return a weighted average of neighbors as predicted value

Week 3 1/28/20

Classification Techniques

Cos Similarity is

$sim(X,Y) = X \cdot Y / Norm\ X \cdot norm\ Y$

focus on direction, not magnitude

$norm = \sqrt{\sum X^2 \cdot \sum Y^2}$ → unit vector

Euclidean and Manhattan - Vector Distance

distance also = $d(X,Y) = 1 - sim(X,Y)$

X and Y are 2 vectors

Vector is a record, a row

Cosine similarity ignores magnitude and measures angle between 2 vectors

Distance comes up in KNN

given X, find K most similar objects

If you have high dimensional sparse dataset,

KNN for classification - video store example . 51:00

1. split into test/train
   Evaluate error rate w/test partition
   KNN w/training partition
   · good to randomize rows / it to reduce bias
2. create var w/ column names
   create var for target column
   VS_Target = VS.Incidents15
   Convert categorical data into SSF
   excluding target
   use getdummies
3. Split
   choose training / test 90
   Split data frame
   split target

4. Preprocessing
Normalize data

from sklearn import preprocessing
does normalizing for you!!

min_max_scaler = preprocessing.MinMaxScaler
↑ create normalization object
min_max_scaler.fit(BS_Train)
↑ apply to training data
Create normalized DFs for both
train & test
scaler returns Numpy array

5. Run KNN function 1:07
x = new instance (test)
D = training data, normalized
for K, try different values and
see how it B's error rate
too small - overfit
too large - underfit

the distance return will be normalized distance

6. Classify - voting 1:11

there is a method called Counter

from collections import Counter

print (Counter(neigh_labels))

most_common() gives you majority

Counter(neigh_labels.most_common(1))

↑ both parts of tuple

np.tile - takes input and creates that many copies of it

7. Calculate error rate

error count / test instances

Classification
when prediction is discrete
or nominal values
(e.g. class / category labels)
fixed set of categories

Regression -
predicting continuous variable

Consistency in ML means
hypothesis consistent
w/ training data
What you are trying to get to

Hypothesis?
shape = circle → category = POSITIVE
red → POSITIVE
true? consistent

2:01

slide 5

May have to get more & more specific
to get to consistency
shape = circle & color = red → category = POSITIVE

Classification accuracy
% of instances classified correctly

Lazy - unlike decision tree, KNN doesn't build model beforehand
Happens when you go to classify a new instance

Classification is a 3 step process
2 if data split
1. model construction (learning)
2. Model evaluation (accuracy)
3. Model Use (classification)

Classification Methods
DT
KNN
Bayesian
→ Logistic Regression
Neural Net.
Linear Discriminant Analysis
SVM
Genetic Algorithms
Random Forest
↑ ensemble

Evaluating Models

- Training Set
- Test Set

Evaluation Set → used to test model performance

Cross validation (slide 13)

divide data (randomly) into n folds each w/ same # of records

Each model is trained w/ n-1 folds & tested w/ remaining fold

Used to find best algorithms & optimal training parameters

Steps - slide 13

assess final model using evaluation set

For example - 5 folds
train on 4/5 thr, test on 1/5 th
do it again w/ another 4/5 thr
until all data used
average results

Then D parameters & do it again

Build model on full training data
not just folds

See slide 14 for graphic

another approach - Bootstrap
Validation
sampling w/ replacement
n instances in data set
sampled n times

Measuring Effectiveness

in Classification, most common measure is accuracy using confusion matrix (Slide 16 2.22)

		Predicted T	Predicted F	Total
actual	T	18	2	20
	F	3	15	18
Total		21	17	38

error rate is inverse of accuracy

		Predicted T	Predicted F
actual	T	TP	FN
	F	FP	TN

accuracy = TP + TN / Total

error rate = 1 - accuracy

Sensitivity = TP / all P (class label of interest rarely - cancer)

Specificity = TN / all N

other measures

Precision

% instances classifier predicted as + that were + $TP/TP+FP$

perfect = 1

Recall

% + classifier predicted correctly as + $TP/TP+FN$

perfect = 1

F score

mean of precision & recall

$F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$

Numeric Prediction / Estimation
typically uses some form
of regression

linear → response variable Y
predicted by single variable X

multiple linear → >1 X

use least squares

Non-linear regression

$y = w + w_1 X_1 + w_2 X^2 + w_3 X^3$

can convert to linear w/
new variable

$X_2 = X^2$, $X_3 = X^3$

Regression Tree

CART - used both for class
and regression

Model Tree

generalization of Regression tree

There do better than linear regression

Evaluation Metrics of Prediction (Numeric Prediction)

how close your result to actual

MAE $|P_1 - a_1| + |P_n - a_n| / n$

RMSE $\sqrt{\frac{(P_1 - a_1)^2 + \ldots + (P_n - a_n)^2}{n}}$

↑

std deviation of error

High Bias - model underfits low Var data

Low Bias - over fit

High Var

High Bias (underfitting)?
add features
use more complex model

Low Bias - overfitting
more training instances
↓ features
simpler model

regularization coefficient - used in regression

Text Categorization Learning

Bayesian (naive)
kNN
relevance Feedback
Nearest Neighbor
SVM

Similarity - Cosine most effective
w/ composite weights for terms
based on how frequently
term appears across docs
TF-IDF

TF = term frequency
idF = inverse doc frequency
Goal: assign a TF x idF to each
term in each document
idF - penalizes words that appear
in every doc, like stop words

Relevance Feedback (Rocchio)
get initial results of query
get feedback
modify original query
re-issue search query

Read TF-IDF notebook
Bayes class video
Decision Tree Video

General ML Procedure

1. Get Data
2. Clean data
3. Separate Class into its own dataFrame → target
4. Get Dummies for cat data
5. train-test-split w/ sklearn which
   - gets train, test, target train, target test
     giving it training data and target and test size (0.2)
     (random state value will give you same random data vs. new random each time)
6. Normalize Train & test data
   a. min-max-scaler = preprocessing.MinMaxScaler().fit(Xs-train)
   b. apply to train & test datasets
7. Import & Fit Classifiers
8. Call Predict Function on a test Instance

and Print the Test

Example

```
# Create classifier
knnclf = neighbors.KNeighborsClassifier(
    5, weights='distance')
# Fit it
knnclf.fit(VS_Train_Norm, VS_Target_Norm)
# Predict
knnpreds_Test = knnclf.predict(VS_Test_Norm)
```

10. Run Classification Report, etc. to Evaluate Accuracy

```
from sklearn.metrics
Print(classification_report(VS_Target_Test, knnpreds_Test))
                                  ↑ labels
```

11. Get Avg Accuracy across Test Instances

```
Print(knnclf.score(VS_Test_norm, VS_Target_Test))
```

12. Compare to Performance on training data to W for over/under fitting

```
Print(Knnclf.score(VS_Train_Norm, VS_Target_Train))
```

Don't Need to normalize for Decision Tree

Overfitting – Train Data Score high
Test Data Scor Low

① regression - get X train y train
X test y test

set aside Test set
model selection (looking at parameters, diff model(s) done on Training
→ Cross-Validation done on training
Once you get params & model
Then train model on full training
Then test on test.

Typical model use (trains model)
create model object
Fit using that mode on data
Evaluate (test)
predict (y_predict = clf.predict(X))
show Accuracy
Show Classification Report
Show Confusion matrix

Feature Selection

From sklearn import feature_selection

most informative attributes

percentile = 30 → top 30%

measure correlation between response & variable

classification uses cat vars so uses χ^2

→ regression uses f value - independently

does not look at interactions among variables or combos of variables

Xtrain_fs = fs.fit_transform(Xtrain, ytrain)

creates subset of attributes

2:24 2/11/20

fs.get_support tells you how features did (T/F)

fs.scores → χ^2 values → higher better

Now need to evaluate performance of new smaller feature set

- do training again

1. create classifier (like dt = tree....)
2. fit it (X_train^FS, y_train^B)
3. transform on TEST TS transforms(X_test^FS)
4. Evaluate performance

Can use crossvalidation to do feature selection to determine percentile (line 24)
finds optimal percentile
Then translate % to column #
there are my optimal # of features

Evaluate the best # of features on test set
feature_selection.selectBEST(...optimal_num_features)

X_train_FS = FS.Fit_Transform(X_train, y_train)
dT = classifier
dT.Fit
X_test_FS = FS.transform(X_test)
measure performance

Model Selection

determine best params for model

you can see model params w

```
print(dt.get_params())
```

one way

1. build classifier object w/ param
2. `scores = cross_val_scores(dt, X_train, y_train, cv=5)`
3. `scores.mean()`
4. try another value for param 3 run again

like `criteria='entropy'`
`criteria='gini'`

for example

5. try best on full training data vs. cross validation
6. measure performance

You fit to train (X_train, y_train)
Measure performance on test

For more systematic way to test params, look at function Calc_Params line 32
uses X validation to test parameters

Classification → score returns classification accuracy

Regression → score needs to return MSE or RMSE

* Grid Search CV gives more systematic way to look at different numbers of multiple params

from sklearn.model_selection import GridSearchCV
so much easier

Regression steps in general

See Ridge Regression in Regression-SciKit-Learn.ipynb

1. Create regression object

```
ridge = Ridge(alpha=0.8)
```

2. Train model using training data set

```
ridge.fit(X,y)
```

3. Compute RMSE on Training data

```
P = ridge.predict(X)
err = P-y
Total_Error = np.dot(err,err)
rmse_train = np.sqrt(Total_Error
                     / len(P))
```

4. Compute RMSE using 10-fold cross validation

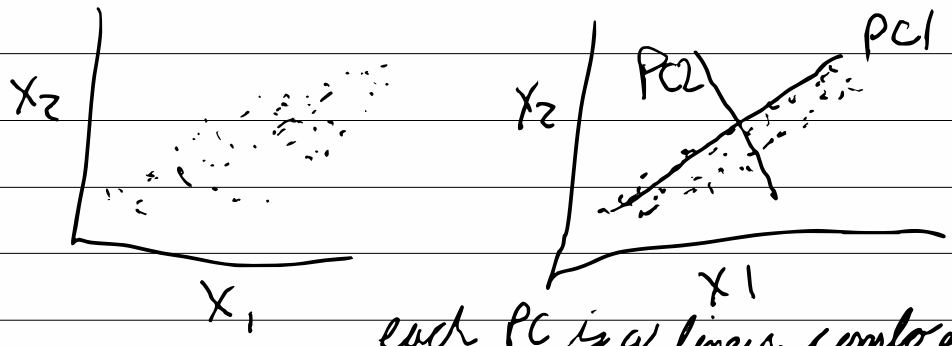
```
rmse_10cv = cross_validate(ridge,
    X,y,10)
```

↑ See Notebook for this function

compare RMSE and RMSE_CV

PCA

orthogonal means they are NOT correlated - reduces noise



each PC is a linear combo of original 2 variables

PCA

data matrix A

n objects

p variables (which may be correlated)

Summarizes it by creating uncorrelated axes (PCs) that are linear combinations of the original p values

Computed using e-vectors & e-values of the cov-variance matrix

Principal
so each component is a
linear combo of original variables

Co-variance of 2 variables is their
tendency to vary together

1st PC captures the most variance
in the data
2nd next most variance
first few give you most of the
info in the data

you will ultimately have P PCs

Computed using COV matrix and
e-values and e-vectors
PCs are the e-vectors

the key!

You have P original variables

$X_1, X_2, \ldots X_p$

Produce new variables

$Y_1, Y_2, Y_3, \ldots Y_p$

$Y_1 = a_{11} X_1 + a_{12} X_2 + a_{1p} X_p$

$Y_2 = a_{21} X_1 + a_{22} X_2 + \ldots a_{2p} X_p$

$\vdots$

Y_i's are the Principle Components,

The a's are the values in the eigenvectors

Y_i's are uncorrelated w/ each other (orthogonal)

$Y_1 \Rightarrow$ explains the most variance in data set

So you pick the top k Y's, drop the rest

1st component is Y_1 - 2nd will be orthogonal to that Y_2

Eigenvalues represent the variance of the PC

Eigen vector → the coefficients in the linear combo

$Z_1 = [a_{11}, a_{12} \ldots a_{1p}] \rightarrow \lambda_1$

eigenvector of the covariance matrix
= coefficients of 1st PC

$Z_2 = [a_{21}, a_{22}, a_{23} \ldots a_{2p}] \rightarrow \lambda_2$

2nd eigen vector of cov matrix and coefficient

dimensionality reduction - you don't need all P variables. You try to figure out how much variance is captured in the top PCs starting from 1st and going down (the lambdas)

- For independent variables, orthogonal $cov(x,y) = 0$
  $cov(x,x) = var(x)$
  cov matrix is square & symmetric
  diag for independent variables are just the variances themselves
  and cov are the non-diag elements

- To calc cov matrix from dataset
  1- center data by subtracting the mean of each variable
  2- compute $\frac{1}{n-1}(A^T A)$
  example slide 10
  use this table to find PCs

Eigenvalues are the variances of the coordinates on each PC axis

So you recompute the matrix w/ the subset of eigenvalues & vectors & compare to original matrix

Higher λ → contains more info

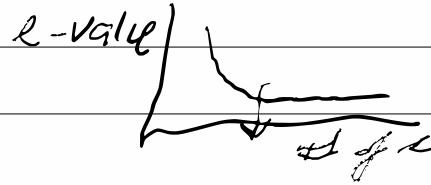
sum of λ's called the trace

score for a PC -

$$y_{ki} = z_{1k}x_{1i} + z_{2k}x_{2i} \dots$$

y is $n \times k$ matrix of PC scores

Screeplot - the knee is the # of components



Single Value Decomposition

Matrix $D = m \times n$

m customers n items

Use when

- D is sparse (less than 1% of entries have ratings, for example)
- n is large

so finding matches to less popular items difficult

Main idea:

compress the columns (items) into lower dimensional representation

decompose

$$D = U \Sigma V^T$$

$m \times n \quad m \times n \quad n \times n \quad n \times n$

$V^T \rightarrow$ eigenvectors of $D^T D$

Σ is diagonal - singular values - like PCs

U rows are coefficients ↓ show variance

Numpy SVD - pass to it matrix you want to factorize

spits out U, S, VT matrices

U - m x m - LEFT singular values of original matrix

S - matrix of singular values m x n

VT - V transpose n x n - right singular values

original matrix = np.dot(U, np.dot(np.diag(S), VT))

SVD gives you top k largest singular values in sigma

select the columns and the rows from V^T

goal is smaller # of dimensions

pick small # of singular values (S)

corresponding # of columns in U matrix and rows in V^T matrix

low_rank = np.dot(U_low, np.dot(
Sigma, VT_low))

V^T is the lower dimension version
of original data

TD in example
3 dimension (vs 10 dimension)
version of each of the documents
3 "terms" vs. 10

Good for info retrieval

Line 65 - gives you lower dimension
version of query
You also have lower dimension
version of documents
67 - measure similarity between
query and documents

Combined terms are like themes

We clustering terms.

Recommender Systems:

You have users ratings (r)
items
measure utility of items i for
user u
recommendation to each user made
by offering items w/ highest
predicted ratings

some function R: users x items →
ratings

Collaborative Recommender system
Neighborhood based Models
User based
K most similar users
Knn
Predict rating on item as
weighted avg of ratings by neighbors

model-based

item-based (items vectors of users w/ ratings as values)
- item-item similarities measured by if rated similarly by users
- predictions for target item based on user's own rating on similar items

Content-based
based on content of items
movies - based on ratings of director

→ pairwise similarity of items

find $P \geq Q$ that give you lowest sum of square error
using wise gradient descent converges

SVD for Matrix Factorization

$$M_K = U_K \times \Sigma_K \times V_K^T$$

U_K = user-feature matrix

V_K^T = feature item matrix

Ratings prediction computed using dot products

where K is lower dimension

Evaluating Recommender System

normally - separate train/test of ratings for each user

Use KNN or whatever on train data

Predict set aside ratings

Look at difference between predict and test (MAE or RMSE or regression)

P1&2 - Difference is you set aside a portion of each users ratings

standard

data matrix is user x item matrix

user - index for one of the users

item - index

goal - generate prediction rating for this user, this item

look at all items similar to item

measure similarity of all items

then looks at user's ratings of these items

weighted avg of all ratings is the rating

user = user

item = j

if rating is 0, do nothing

if rating not 0 look at overlap of item j and item d can look at in terms of overlap

if overlap is not 0, found similarity
predicted rating will be weighted avg of all ratings
user rating × similarity / all similarity
return rating

recommend
N is # of recommendations you want
for each user, predict rating of all items & return top N

SVDEst - only difference:
estimation is SVD based
1st does singular value decomp on the data
creates transformed representations of items based on transformed feature space of items
Varkoded 4 as dimension space

do not use this for assignment 4
(listing 14.3)

Test of item based recommender system
all similarity measures

[12] reduced dimension space for
items reduced to 4 dimensions
nowrate items
recommend function
pass data matrix
a user (#4)
similarity
estimate method (std item-
based)
returns index of item 3 predicted
rating

Jobe Ratings
use gradient descent optimization
approach for learning the P and Q

models the rating data
rows = users
columns = jokes
values = rating

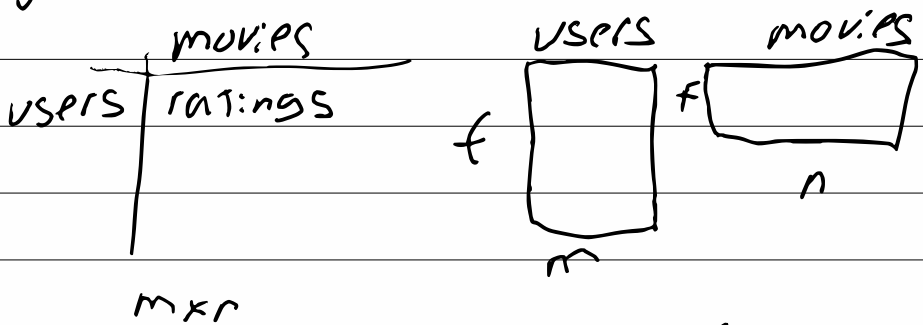
matrix-factorization functions
uses gradient descent to
learn matrices P & Q
slow - full GD, not stochastic
3000 steps (epochs)
Predictions for user 979, joke 9
dot product of P[979], Q[9].T

if you want all predictions, dot(P, Q.T)
preds = np.dot(P, Q.T)
includes ratings that were
already known
Fast - goes thru rating data,
generate predictions
compare, find error, avg for MAE

Matrix factorization is good for sparse data like user ratings in Netflix

use factorization of a sparse matrix to create two dense matrices

e.g.



also for reducing large # of features to fewer

Support Vector Machines

another supervised method

primarily used for classification
particularly for image classification
and text classification

Many class. models use distribution
to determine labels for new
points (like Naive Bayes)

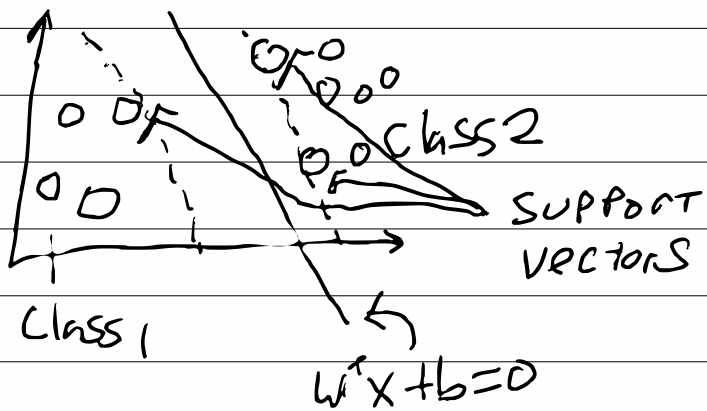
SVM: discriminative classification:
instead of modeling each class,
find a line or curve in 2d
that divides the classes from
each other

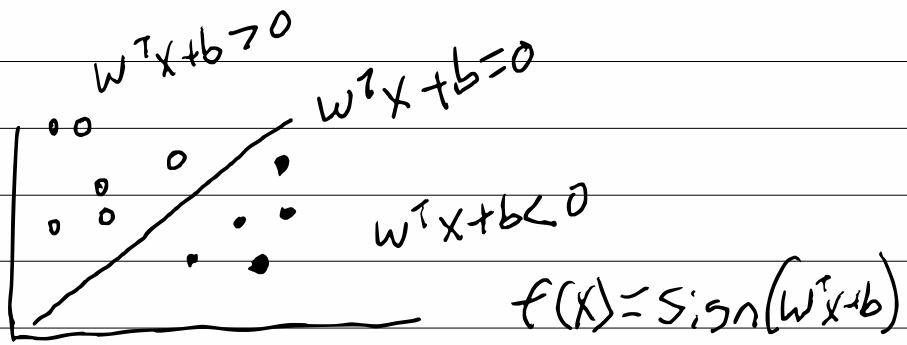
LDA is another discriminative
classification

Want a decision boundary that is max
distance from both classes

SVM tries to find that boundary - called maximizing the margin

The points that push up against the margin are the support vectors





norm = $\sqrt{\text{sum of squares}}$

Nice because only points I care about are the support vectors

Size of dataset doesn't matter

Soft Margin Classification
when training set not clearly linearly separable

added slack variable to manage noise resulting in margin called "soft"

Conjugacy and priors

Subjective Prior
expert opinion used to determine probability

Objective Prior
I don't know much about problem
so I put in a non informative prior
no prior info available
use a wide distribution
like Uniform - all probabilities equally likely
or diffuse distributions

Conjugacy

when posterior distribution is the same family as prior, but w/ new parameter values

Ex, for example

Beta prior - Beta posterior

↓ Binomial

Advantage - do not need to calculate integral

Week 8

Use MAP if you need a point estimate for the posterior (maximum value)

Use MLE if you need the likelihood

EAP - expected A Posteriori - expected value of posterior distribution

Trace plot - estimation from sampling

pm.find_MAP - gives you a point estimate

pm.summary(trace)

mean
interval
Markov chain
for each one of your unknown parameters

Bayesian sampling methods
used to find posterior density distributions
You sample from posterior distribution to approximate the posterior
Become more exact as more samples are used
PYMC3 - need hundreds of thousands of samples
Markov Chain Monte Carlo sampling (MCMC)
main method to sample many variations

Monte Carlo is just a sampling method

Markov Chain - all info from past captured in my present and can be used to predict future
used to create prob distributions over time

Metropolis Hastings - type of MCMC
a variation
weighted random walk
accepts or rejects proposal from MC