Scikit-learn Introduction

Introduction

Supervised Machine Learning

Clustering

Recommendations

Classifications

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Lesson Objectives

- Understand some of the basics of Machine Learning (ML)
- Learn how Python Scikit-learn supports ML
- Become familiar with some of the algorithms in Scikit-learn

Introduction

Introduction

Supervised Machine Learning

Clustering

Recommendations

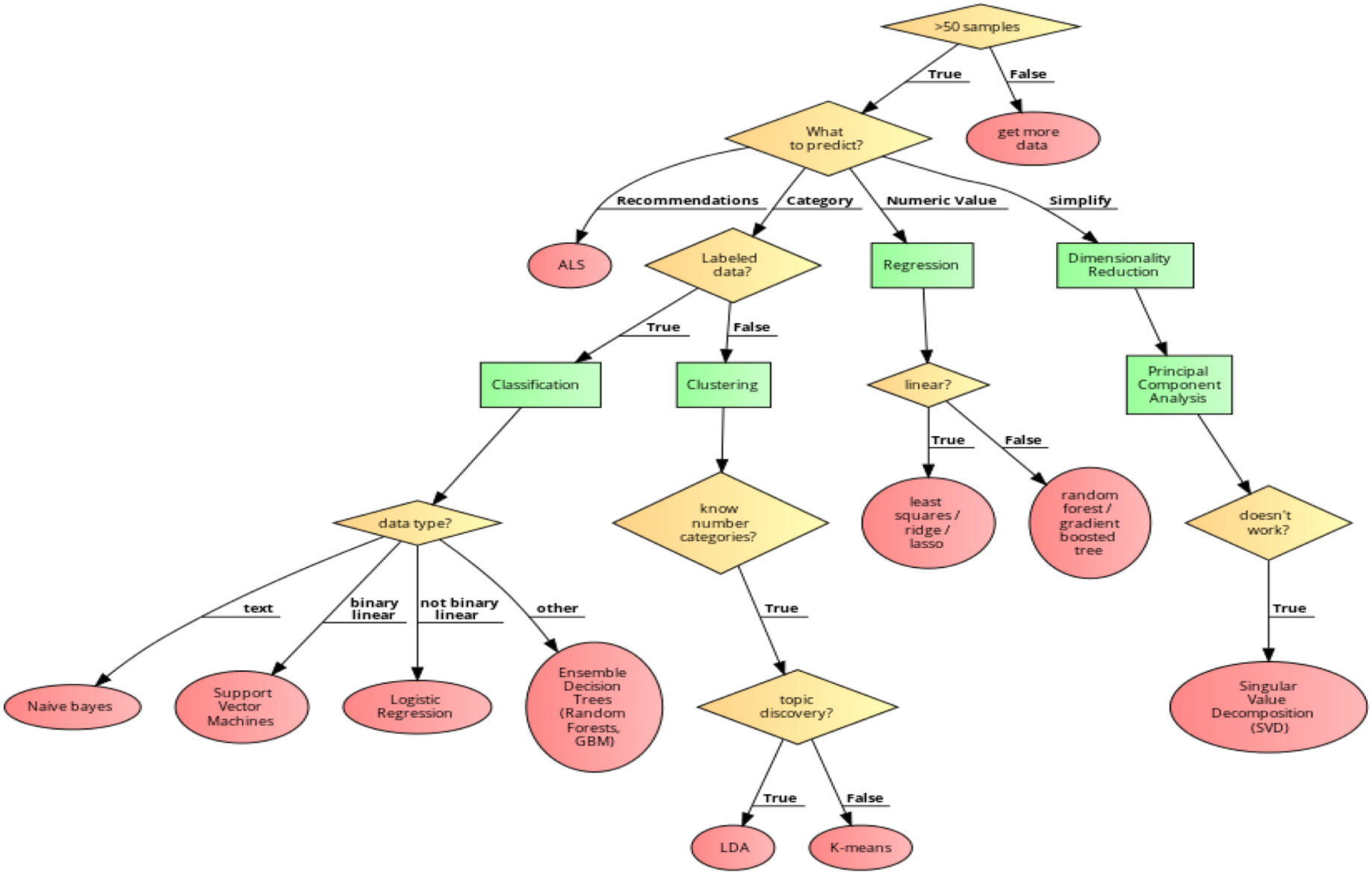
Classifications

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Machine Learning in Python

- Python Scikit-learn library provides lots algorithms
- Lots of common algorithms are supported
  - Classification/Regressions
    - Linear models (linear R, logistic regression, SVM)
    - Decision trees
  - K-Means clustering
  - More

Scikit-learn Algorithm overview



Estimators (base class)

- Estimators are classes that have a .fit() method
- Mostly, these are classes that are used for prediction:
  - Regression
  - Classification

```
>>> estimator.fit(data)
```

- Estimators usually take parameters

```
>>> estimator = Estimator(param1=1,
param2=2)
>>> estimator.param11
```

Calling fit()

- fit can take one or two arguments
- Unsupervised learning:
  - Typically just the data
- Supervised learning: 2 params
  - Data (as ndarray)
  - Labels (the correct answer)

Transformer (base class)

- Transformers are also Estimators
  - They have a .fit() function
  - It may not necessarily be used.
- They also can transform data
  - They also have a .transform() function
- A trivial example (taking the log of the input)

```
>>> transformer =
FunctionTransformer(np.log1p)
>>> transformer.transform(np.array[1,2])
[0.69314718, 1.09861229]
```

- Typically transformers are used to process our data
  - For example, to scale or normalize

Pipelines

- Usually we have many estimators and transformers together
- We can create a pipeline class to hold a pipeline of estimators and transformers.
- Pipeline class
- We can combine transformers together
- Or provide transformers + final estimator

Prediction

- Estimators can also have a predict() method
- Used for predicting results of a model
- An example classification model using a SVM:

```
>>> from sklearn import svm
>>> clf = svm.SVC(gamma=0.001, C=100.)
>>> clf.fit(data, label)
>>> clf.predict(np.array([1,2]))
array([3, 4]) # Prediction output
```

Feature Vectors

- Sklearn runs on Feature Vectors stored as ndArrays
- Pandas series can be converted to NdArrays is they are numeric (usually float64)
- We need to extract columns from Pandas dataframe,
  - Convert them to ndarrays
- But how do we do that?

Categorical Variables

- Algorithms run on ndarrays and matrices of type float64
  - Bool, int64 will be upcasted to float64
- Your data may have string or categorical variables
  - What to do?
- For a label:
  - Use the LabelEncoder
  - Will convert label into integer value and back.
- For a column:
  - Have to convert into a numeric

Strategy 1: Factorization

- Factorization
  - Call pd.factorize() on the dimension
  - Will assign an integer to each unique value
  - E.g. (red = 1, green = 2… etc)
- Problems with Factorization
  - Number may not be meaningful in any way (arbitrary)
  - Model may try to infer something about ordering.
  - Model may not understand that these are discrete values
  - Model may infer some correlation beween values.
  - “Finding signal in the noise” – can lead to bad results

Strategy 2: Dummy Variables

- We can use Dummy Variables
  - Aka One Hot Encoding
  - This will generate One hot encoded features.

```
from sklearn.preprocessing import
OneHotEncoder

fact = pd.factorize(dimensions)
OneHotEncoder().fit(fact).transform(fact)
```

Strategy 3: Quantization

- Turn the category into a real-world number
- Example colors:
  - Red = 450MHz
  - Green …
- Usually join (pd.merge) to a ”lookup table”

Scaling and Normalization

- Usually data needs to be cleaned up and transformed before creating features
- Scaling
  - StandardScaler class will scale data down
  - Remove high magnitude data
- Normalization
  - Will use Z-scoring to norm

Data Whitening (PCA)

Introduction

Supervised Machine Learning

Data Whitening (PCA)

Clustering

Recommendations

Classifications

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Data Whitening

- Data that is heavily correlated may need to be whitened.
- This means de-correlated.
- One way to do this is with PCA: Principal Component Analysis.
- PCA can also be used to reduce dimensionality

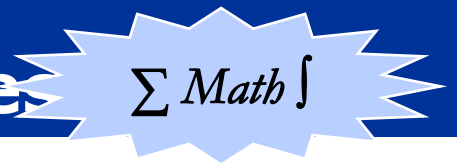
Principal Component Analysis (PCA)

- Not all dimensions are equally interesting.
  Can we reduce the **'dimensionality'** of data, **without loosing too much information**?
- PCA does just that
- It finds low-dimensional representation of data that contains as much as possible of **variation**
- PCA seeks a small number of dimensions that are as **interesting** as possible
- **Interesting** is measured by the amount that the observations vary along each dimension

Using PCA for Visualization

- If we reduce dimensions to 2 or 3, we can visualize highly dimensional data
- Great alternative to Scatter Plot Matrices

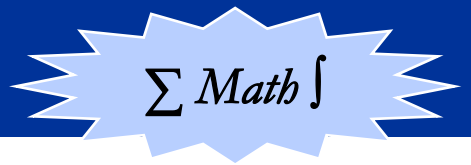
PCA Theory: Eigen Vectors / Values



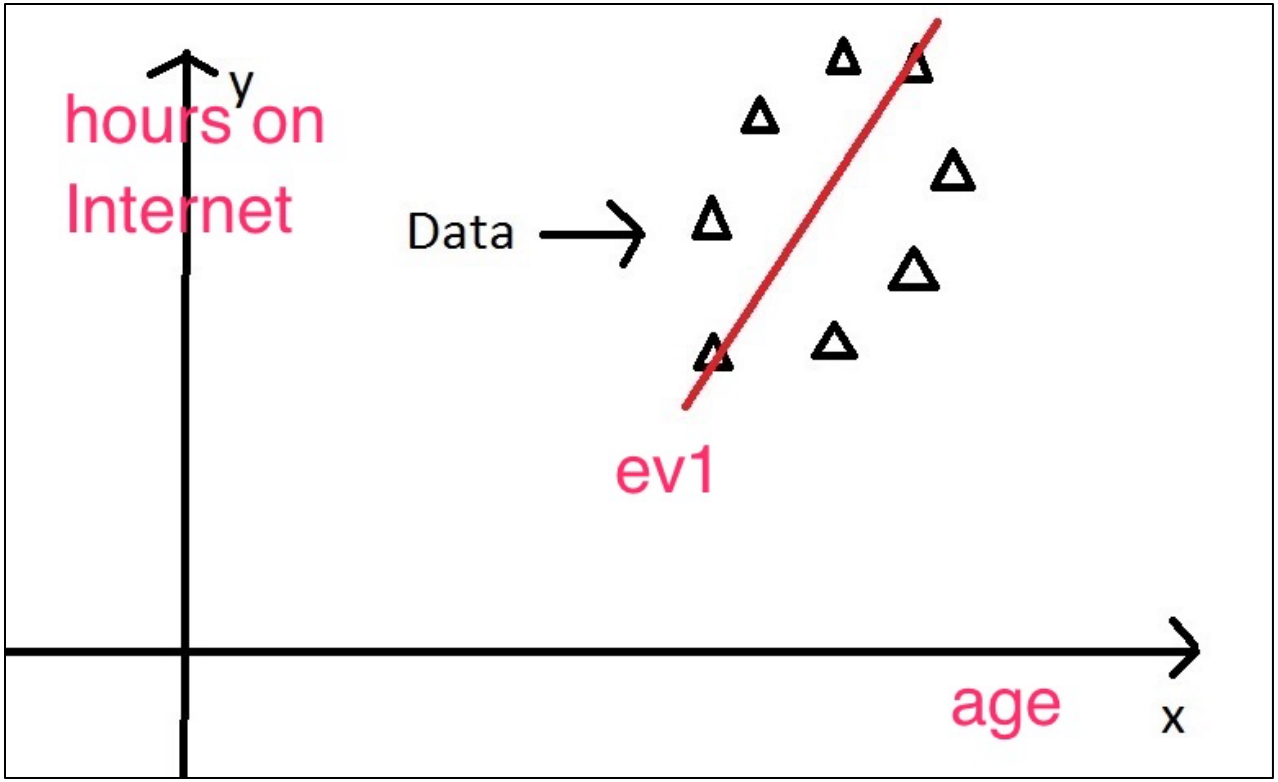
- Say we are measuring the following responses
  - Age, hours on the internet
  - 2 variables → 2 dimensional dataset → 2 Eigen Vectors
- If we measure
  - Age, hours on the internet, hours on the mobile phone
  - 3 variables, 3 dimensional dataset → 3 Eigen vectors
- So number of Eigen vectors = number of dimensions

- EigenVector is a direction – vertical, horizontal, 45' degrees ..etc
- EigenValue is a number – denoting how much 'variance' in the data in that vector's direction
- Eigen Vector and Eigen Value go together (E Vector, E Value)
- EigenVector with highest EigenValue (meaning lot of variance in that direction) becomes a Principal Component

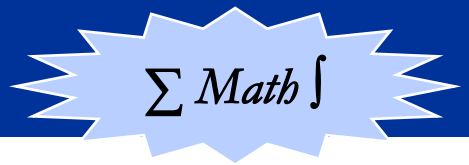
PCA Math: Eigen Vectors



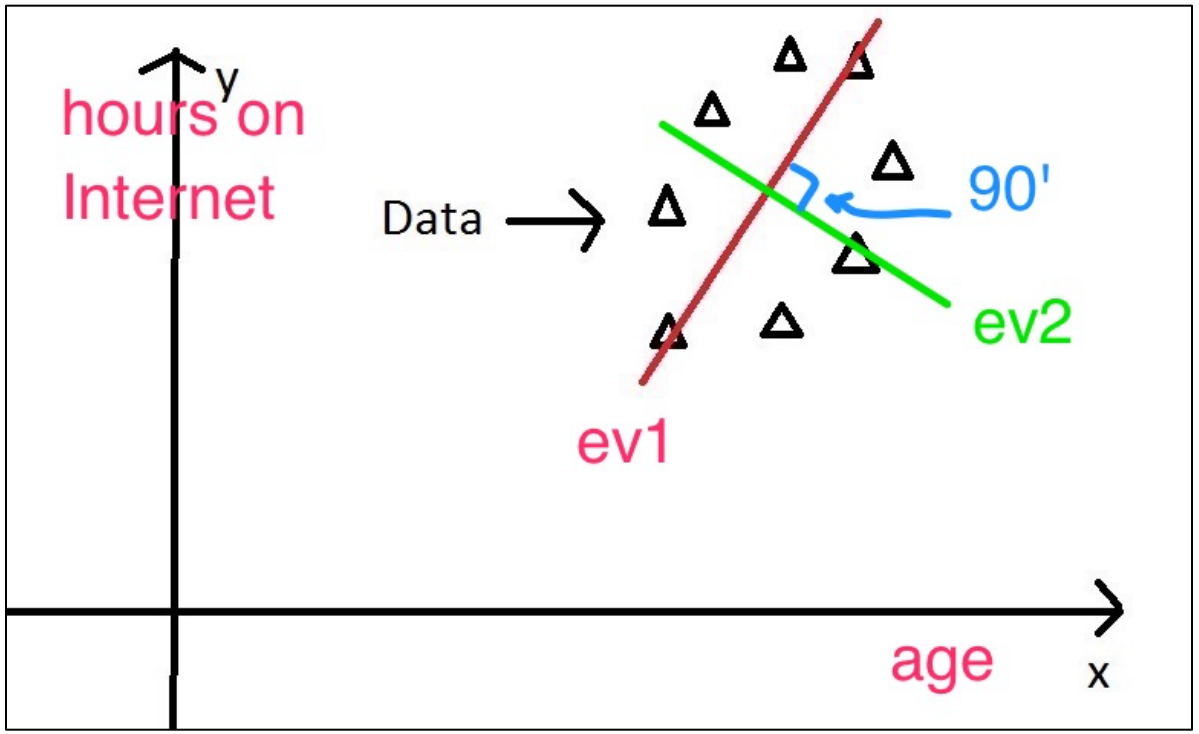
- In the plot below, EigenVector (ev1) is shown that crosses the data with 'highest variance'



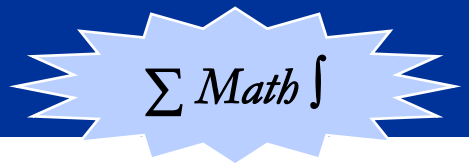
PCA Math: Eigen Vectors



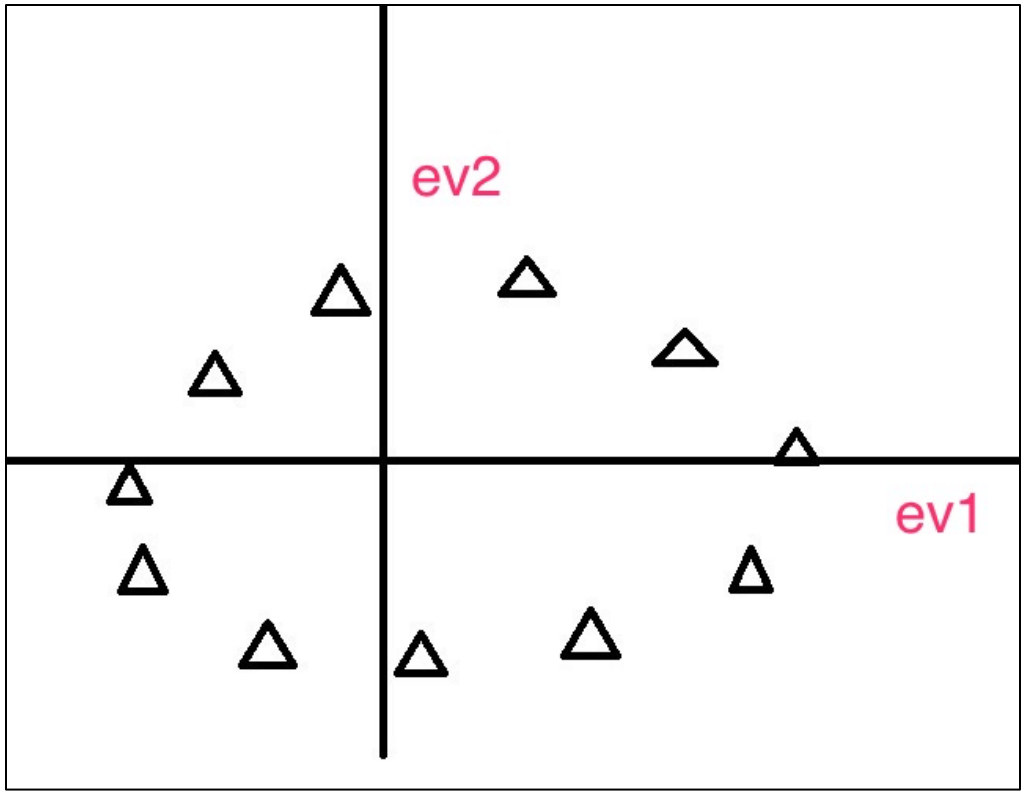
- But we have a 2-dimensional data
  → 2 eigen vectors
- To maximize coverage the second EigenVector will be orthogonal (90 degrees) to the first one (ev1)



PCA Math: Eigen Vectors



- The EigenVectors have given us more useful axis to frame data
- Remember, the data hasn't changed at all, we are just looking at it from a different perspective



PCA on Oxford Internet Study

- Say there were
  - 2500 people interviewed == 2500 observations
  - 15 questions = 15 dimensions = 15 EigenVectors & Values
- Say our 15 Eigen Values are (in descending order)
  [**25, 22, 20, 17**, 10, 7, 6, 5, 4, 3, 1, 0.5, 0.4, 0.3, 0.1]
- We see the first 4 have the biggest values
  - Indicating 4 directions with lots of information
- We have identified our 4 Principal Components (PC)
- We have gone from 15 attributes to 4 (call them PC1, PC2, PC3, PC4)

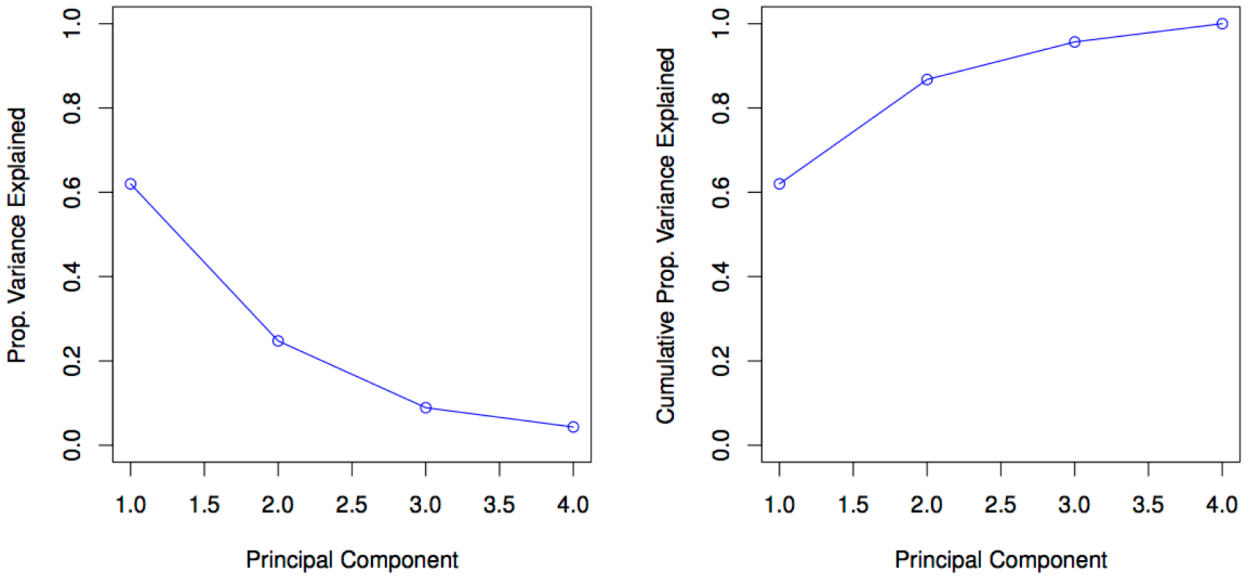
Principal Components

- First Principal Component (PC1) has the largest variance (EigenValue)

- Second Principal Component has second largest variance that is uncorrelated to PC1
  - Orthogonal vector to PC1

Evaluating PCA – Scree Plot

- We use a **scree plot** to understand PCA
- Left chart plots Variance for each PC component.
  - First component (PC1) has the most (62%)
  - Second one PC2 around 25%
  - PC4 hast the least
- Right graph shows 'accumulated' variance when combining PCs. We are moving towards ONE



Clustering

Introduction

Supervised Machine Learning

Clustering

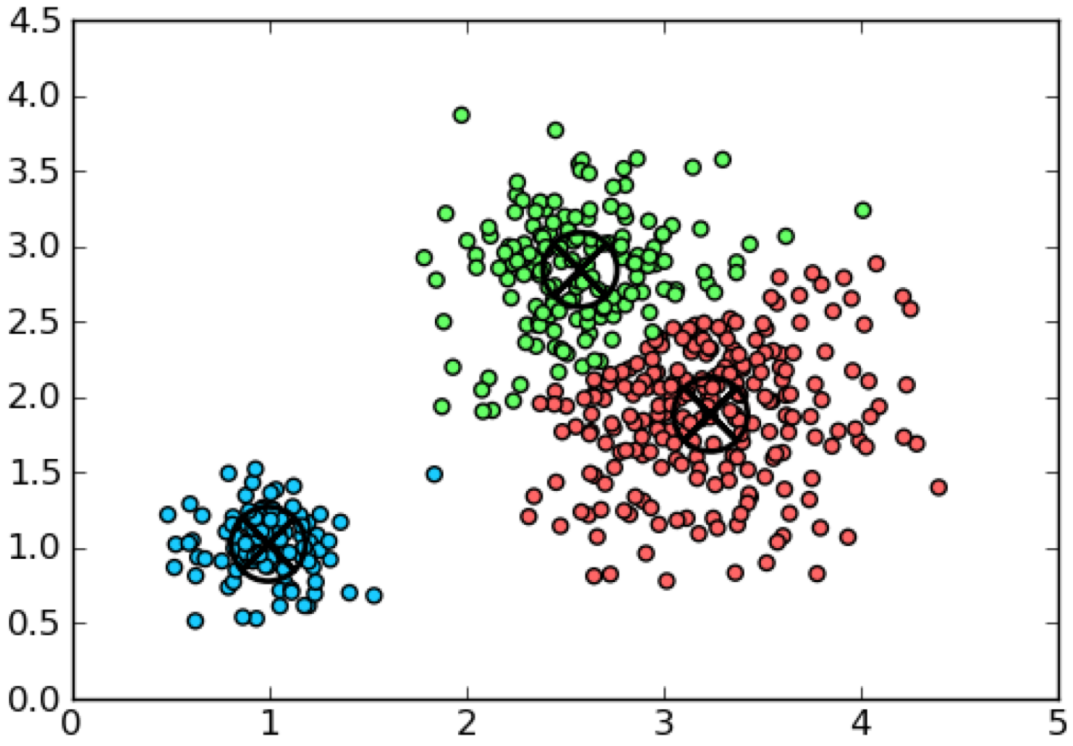
Recommendations

Classifications

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

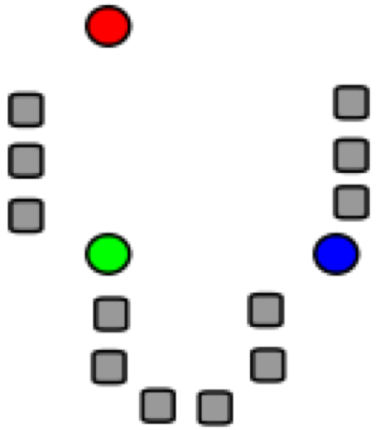
Clustering Vectors

- There are many different clustering algorithms for vectors.
- Simplest is k-means
- K-means requires a known value of k (number of clusters) to start with.

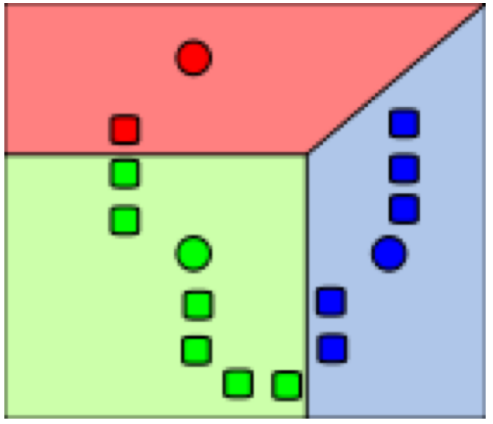


K-Means Clustering

- K-Means: Simplest Clustering Algorithm.
- Step 1: k numbers of points (centroids) are pre-seeded in the data. Example: 3 centroids (red, green, blue)

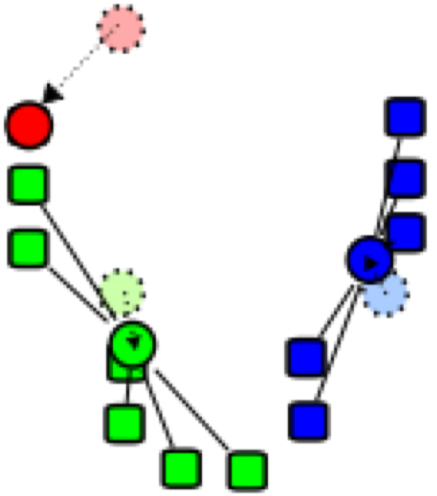


- Step 2: Each point in the dataset is associated with its nearest centroid, as determined by a distance measurement.

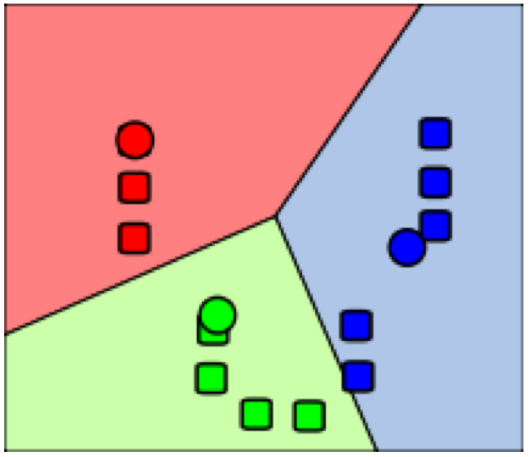


K-Means Clustering

- Step 3: The centroid (geometric center) of the clustered points becomes the new centroid of that cluster. Each centroid is updated.



- Step 4: Repeat steps 2 and 3 until convergence is reached (the points move less than the threshold amount).



K-Means Clustering Summary

- K-Means is simple
  - Well-understood.
  - Easy to parallelize
- Disadvantages:
  - Value of k must be known in advance, which may mean running the exercise many times to get optimum results.
  - Initial centroid positions are important; may cause long convergence.
  - Dense groupings of points are not especially considered
    - Outliers may bias results.
  - Clusters not broadly (hyper)spherical don't work well for k-means.
    - Use hierarchical clustering for these situations.

K-Means in Scikit-learn

- Scikit-learn has good support for k-means
- How to perform k-means clustering in Scikit-learn
  - Put data in ndarray
  - Perform clustering with a specified number of iterations
  - Evaluate the “fit” of the cluster. Is it a good run?
  - If not, change the number of clusters (value of k)
  - Once we have a good clustering run:
    - Map each vector to its nearest cluster
    - Group original data by its corresponding cluster
    - Assign (predict) new vectors to their nearest cluster

K-Means in Scikit-learn

- k-means clustering

```
from sklearn.cluster import kmeans
```

- Default values: just provide KMeans(df, value-of-k)

```
kmeans = KMeans(n_clusters=2,
random_state=0).fit(X)
```

- How many iterations should you have?
  - "It depends"… too low and you might get bad results
  - Too many and you waste time
  - Try 10-20

Evaluating Cluster Performance

- WSSSE: Within Set Sum of Squared Errors

```
WSSSSE = kmeans._inertia
```

  - COST = sum of squared distances of points to cluster center.

- What does this mean?
  - WSSSE will decrease with increasing values of k.
  - "Law of Diminishing Returns"
    - High values of k give marginal gain.

- We can iterate across k until we get good results.

The Elbow Method

- Identify the “elbow” on the curve
- Example: What value of K to select in this case?

Using KMeansModel

- How to apply the model to data?
- Use predict:
  - `kmeans.predict(Vector)`

```
int clusterId =
model.predict(Vector.dense())
```

- Predict returns an integer
  - Cluster number, i.e., 0, 1, 2, 3… (k-1).

Handling New Data

- New Data be assigned on the existing model:
  - Make a Vector out of the new Data
  - Call `KMeansModel.predict(vector)`, to get cluster membership as a number (integer)
- The new data will not affect the existing cluster locations.

Lab : KMeans Clustering on Flights

- **Overview**: Try clustering flights dataset
  - Use NYCFlights
- **Builds on previous labs**: None
- **Approximate time**: 20-30 min.
- **Follow: sklearn/kmeans/README.md**

Classification / Regression

Introduction

Supervised Machine Learning

Clustering

Recommendations

Classification / Regression

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Classification and Regression

- Classificaiton and Regression are both supervised ML.
- Both are to make a prediction
- Classification: predict a categorical variable
  - Could be binary (yes/no)
- Regression: predict a real number
- We ”fit” the model with training data and labels
- We can evaluate the model
- Then we “predict” new data.

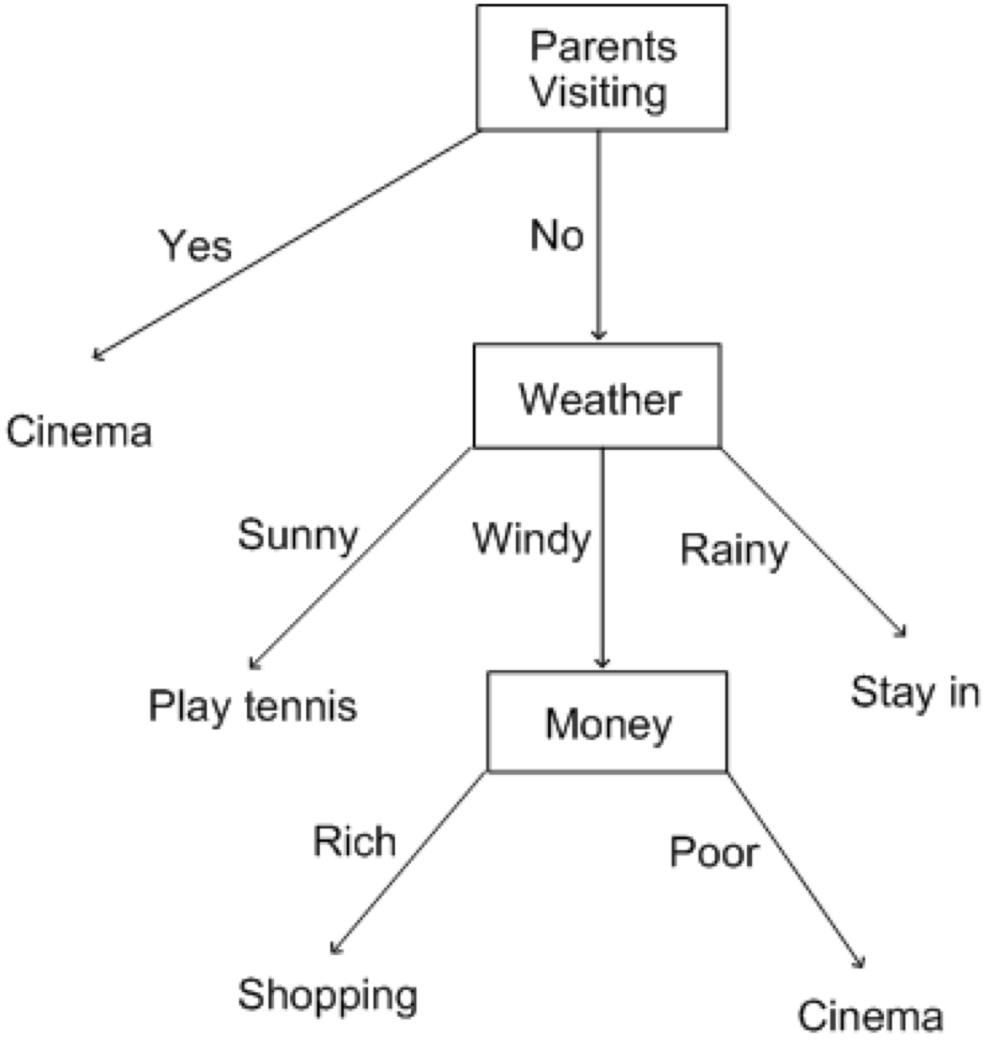
Review of Classification Algorithms

- Naïve Bayes
- Support Vector Machines (using Stochastic Gradient Descent)
- Logistic Regression/MaxEntropy
- Decision Trees
  - A basic decision tree is pretty simple.
  - It is a set of rules collect the rule
  - Decision-tree learning is a means of inferring an appropriate decision tree from the data.

Support Vector Machines (SVMs)

- Support Vector Machine is a classification method that is:
  - Supervised (trained)
  - Linear
  - Binary (splits into two classes) (E.g., Spam or not-spam)
  - Crisp (not fuzzy, not probabilistic)
- How does it work?
  - Draw a line (hyperplane) that separates the two classes
  - Line should perform best possible separation
  - Maximize distance between line and closest points.

Decision Tree



Classification Support in Scikit-learn

- The following algorithms are well-represented in Scikit-learn:
  - Linear methods: SVM, Logistic Regression
  - Decision trees
  - Ensemble decision trees (Random Forests, Gradient Boosted Trees)
  - Naïve Bayes

Evaluating Classification Models

- Let's consider a binary classifier
  - Picks one of two outcomes (spam / not-spam)
- Two approaches
  - Confusion matrix
  - ROC curve

Confusion Matrix / Error Matrix

- Let's consider a binary classifier
  - Picks one of two outcomes (spam / not-spam)
- Say we are classifying 10 emails (6 spam, 4 not-spam)
- See **'confusion matrix'** below

Classification =>	Spam	Not Spam
Actual Spam (6 total)	4 count 66% accuracy Correct **True-Positive Rate (sensitivity)**	2 count 33% Incorrect **False negative rate (miss rate)**
Actual Not Spam (4 total)	1 count 25 % Incorrect **False-positive rate (fall-out)**	3 count 75% Correct **True negative rate (specificity)**

Confusion Matrix: More Than 2 Outcomes

		Predicted		
Actual		**Cat**	**Dog**	**Rabbit**
	Cat (8)	5	3	0
	Dog (6)	2	3	1
	Rabbit (13)	0	2	11

- Algorithm has trouble classifying cats / dogs (too many mis-classification)
- Can classify rabbits pretty well (less mis-classifications)

Confusion Matrix

		Predicted Condition	
		Predicted Positive	**Predicted Negative**
Actual condition	**Positive**	True positive	False negative <u>Miss rate</u> - Guilty prisoner was not convicted
	Negative	False Positive <u>Sensitivity</u> - False alarm - 'Crying wolf' - Hiring someone who is not qualified	True negative

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Confusion Matrix : Calculations

- **Accuracy**

  Overall how accurate is the model?

  = (TP + TN) / total

  = (90 + 70) / 200

  = 0.8 or 80%

- **Mis-classification / Error rate**

  How wrong is the model?

  = (FP + FN) / total

  = (10 + 30) / 200

  = 0.2 or 20%

  = 1 - accuracy

		Predicted Condition	
		Predicted Positive	**Predicted Negative**
Actual condition (n = 200)	**Positive (n = 120)**	True positive (n = 90)	False negative (n = 30)
	Negative (n = 80)	False Positive (n = 10)	True negative (n = 70)

Confusion Matrix : Calculations

- **True Positive Rate (TPR)**
  How often model predicts 'positive' as 'positive' (correctly) ?
  = TP / actual positive
  = 90 / 120
  = 0.75 or 75%

- **False Positive Rate (FPR)**
  How often model predicts 'negative' as 'positive' (incorrectly)
  = FP / actual negative
  = 10 / 80
  = 0.125 or 12.5%

		Predicted Condition	
		Predicted Positive	**Predicted Negative**
Actual condition (n = 200)	**Positive (n = 120)**	True positive (n = 90) TPR = 75%	False negative (n = 30)
	Negative (n = 80)	False Positive (n = 10) FPR = 12.5%	True negative (n = 70)

Confusion Matrix : Calculations

- **Specificity**
  How often model predicts negative' as negative' (correctly) ?
  = TN / actual no
  = 70 / 80
  = 0.875 or 87.5 %
  = 1 - FPR

- **Precision**
  When model predicts 'positive' how often it is right?
  = TP / predicted positive
  = 90 / (90 + 10)
  = 0.9 or 90%

		Predicted Condition	
		Predicted Positive	**Predicted Negative**
Actual condition (n = 200)	**Positive (n = 120)**	True positive (n = 90) TPR = 75%	False negative (n = 30)
	Negative (n = 80)	False Positive (n = 10) FPR = 12.5%	True negative (n = 70) Specificity = 87.5%

- **Prevalence**
  How often does 'positive' occurs in our sample
  = actual positive / total
  = 120 / 200
  = 0.6 or 60%

Confusion Matrix : Calculations

- **Positive Predictive Value (PPV)**

  Similar to precision, takes 'prevalence' into account
  = TP / (TP + FP)
  = 90 / (90 + 10)
  = 0.90 or 90%

		Predicted Condition	
		Predicted Positive	**Predicted Negative**
Actual condition (n = 200)	**Positive (n = 120)**	True positive (n = 90) TPR = 75%	False negative (n = 30)
	Negative (n = 80)	False Positive (n = 10) FPR = 12.5%	True negative (n = 70) Specificity = 87.5%

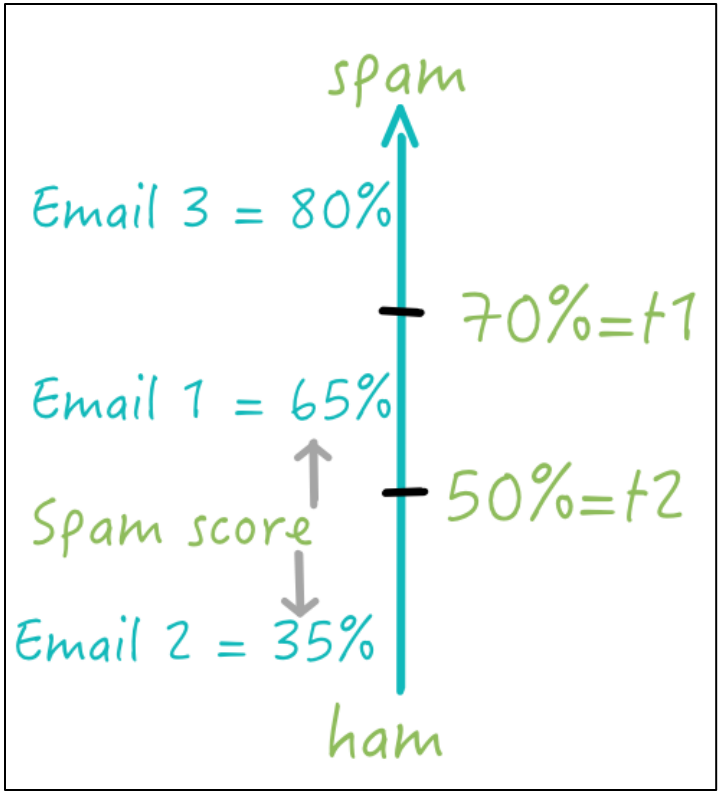
- **Null Error Rate**
  - How often the model would be wrong if it always predicted the majority class?
  - Here our majority = Positive
  - If we always predicted 'positive' we would be wrong 80 times (negative)
  - 80/200 = 40% of time

How is ROC Curve Generated

- Y-axis : True Positive Rate (TPR)
  - Actual=positive, predicted=positive
  - Correct!
- X-axis : False Positive Rate (FPR)
  - Actual=negative, predicted=positive
  - Incorrect!
- 0.0 <= TPR & FPR <= 1.0
- Plot TPR / FPR while varying 'threshold'

Threshold

- Our spam classifier provides a 'spam probability' for each email
  - Probability is between 0.0. and 1.0 (or 0 to 100%)
  - 1.0 definitely spam
  - 0.0 definitely not spam
- When an email's **'spam score'** is above a certain number we mark it as spam
- This is called **'threshold'**
- If spam threshold is lower (say 50%)
  - more emails will be classified as spam (email2, email3)
  - Users will miss emails (as they are in Spam folder)
- If spam threshold is higher (70%)
  - Fewer emails will be classified as spam (email2)
  - Users will see more spam emails be in Inbox
- We need to find the sweet spot !



Threshold

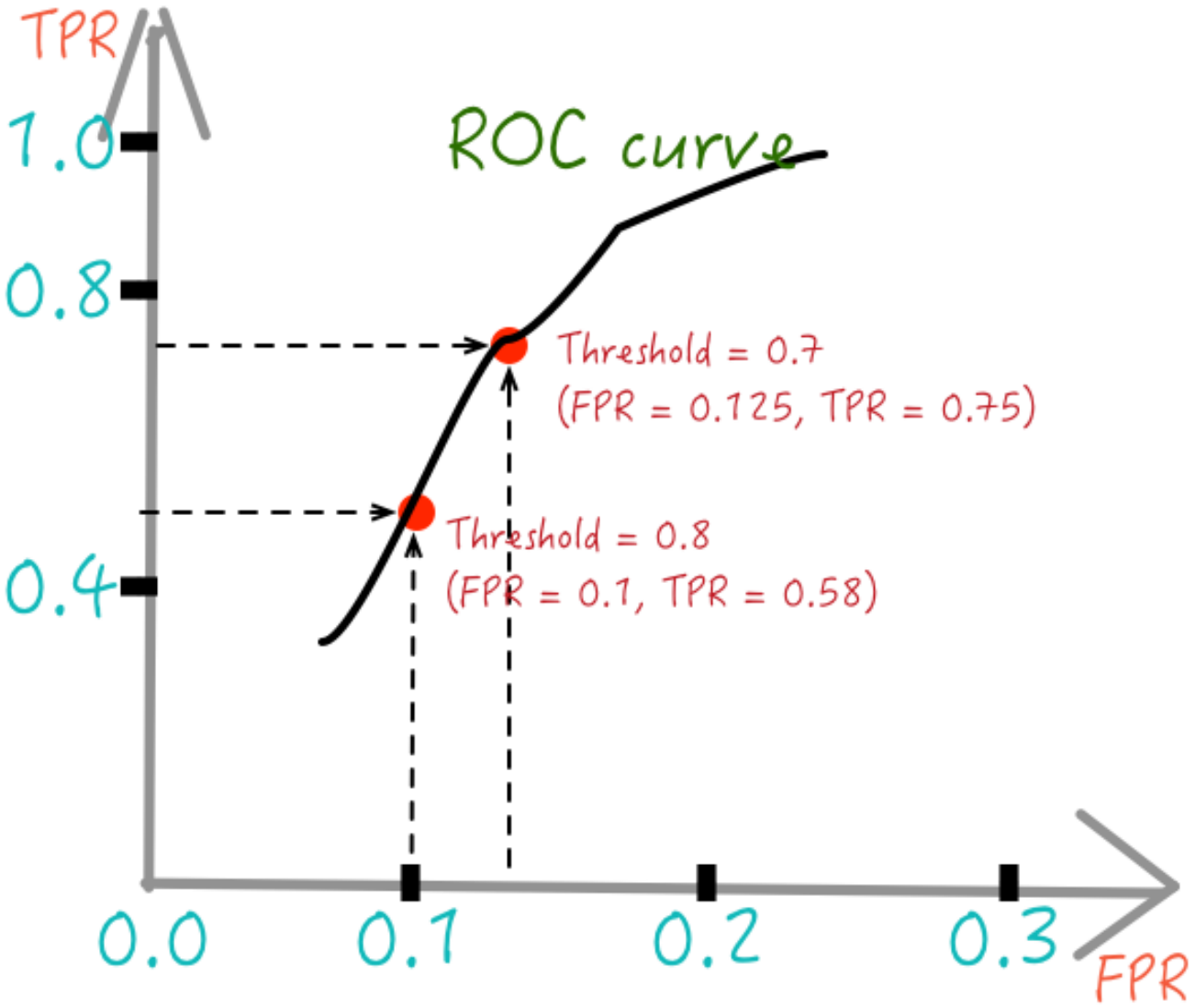
- In first table our threshold is 0.7
  - 90 emails are correctly predicted as spam
- Next table, our threshold is higher 0.8
  - Only 70 emails are classified as spam
  - Lower TPR

		Predicted Condition	
Threshold = 0.7		**Predicted Spam**	**Predicted Not Spam**
Actual condition (total = 200)	**Spam (n = 120)**	True positive (n = 90) TPR = TP / positive = 90/120 = **75%**	False negative (n = 30)
	Not Spam (n = 80)	False Positive (n = 10) FPR = FP / negative = 10/80 = **12.5%**	True negative (n = 70)

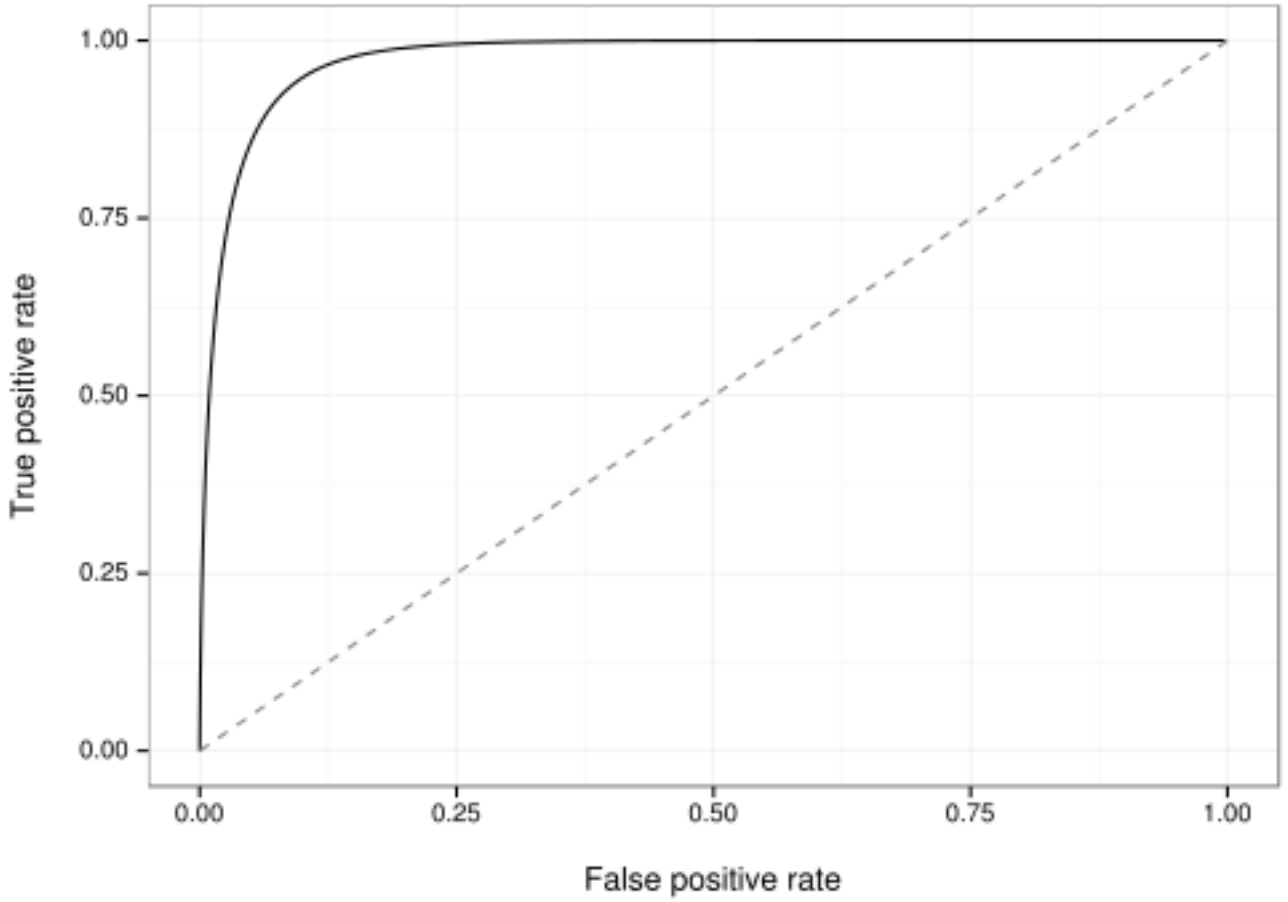
		Predicted Condition	
Threshold = 0.8 (higher)		**Predicted Spam**	**Predicted Not Spam**
Actual condition (total = 200)	**Spam (n = 120)**	True positive (n = 70) TPR = TP / positive = 70 / 120 = 58.33%	False negative (n = 50)
	Not Spam (n = 80)	False Positive (n = 8) FPR = FP / negative = 8 / 80 = 10%	True negative (n = 72)

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

ROC Curve : Receiver Operating Characteristic

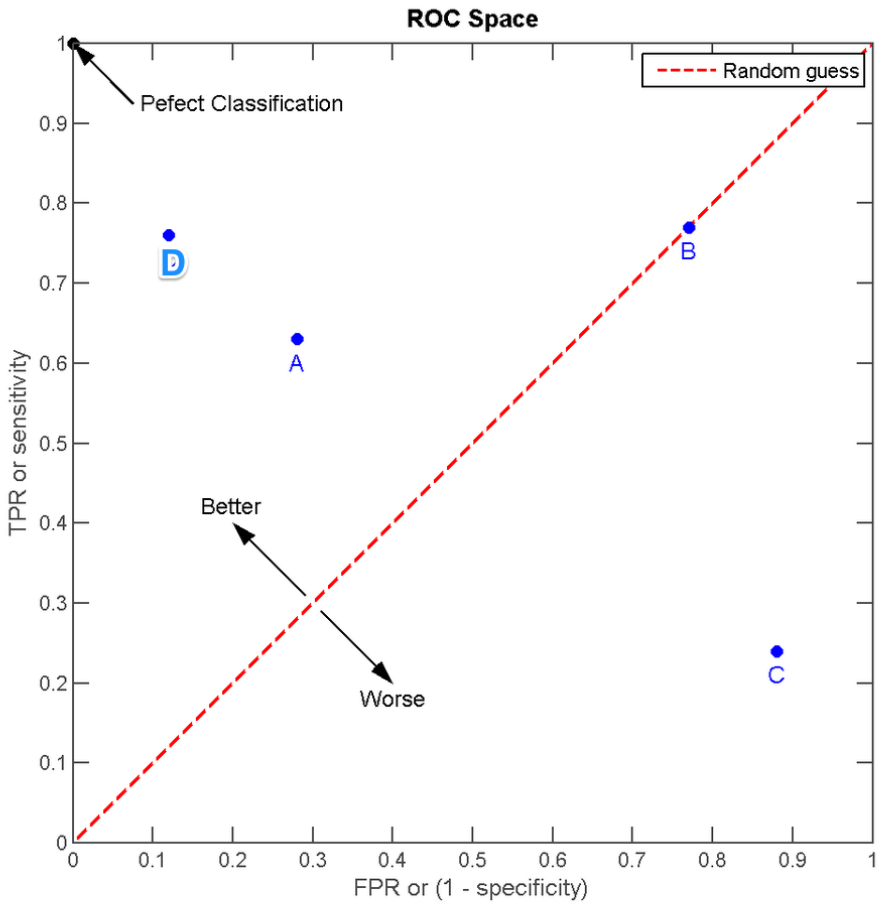


ROC Curve Example



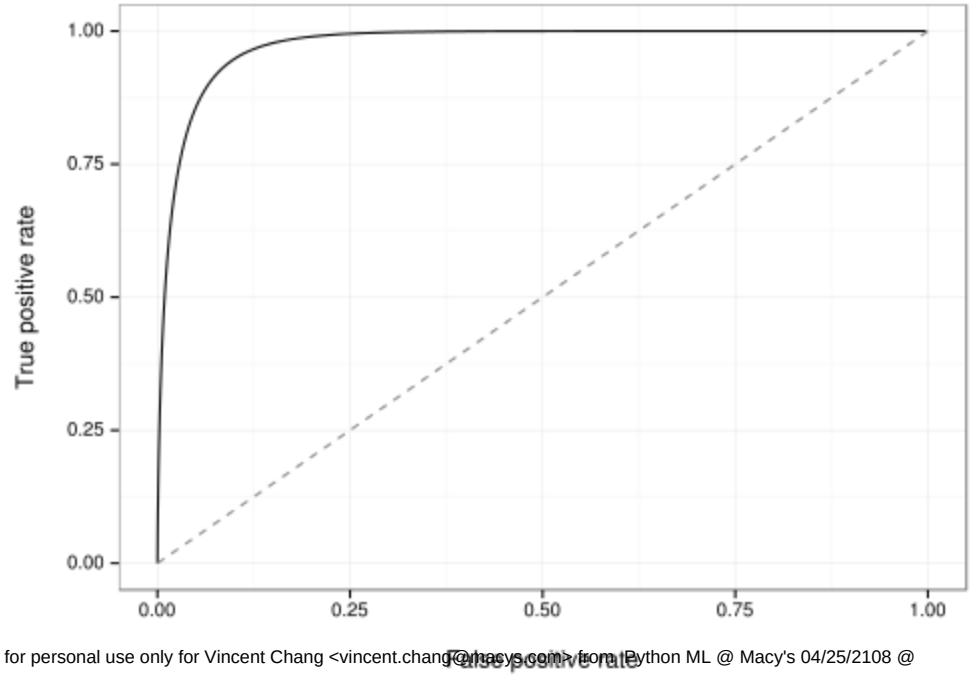
Interpreting ROC Curve

- The red-line plots 'random guess' = B
- Approaching 'top left' corner would be a perfect classifier!
  So D is better A
- C performs worse than random → bad



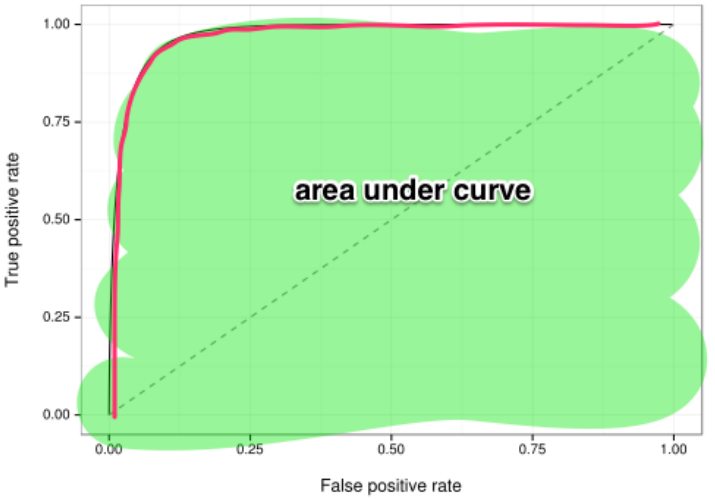
Interpreting ROC Curve

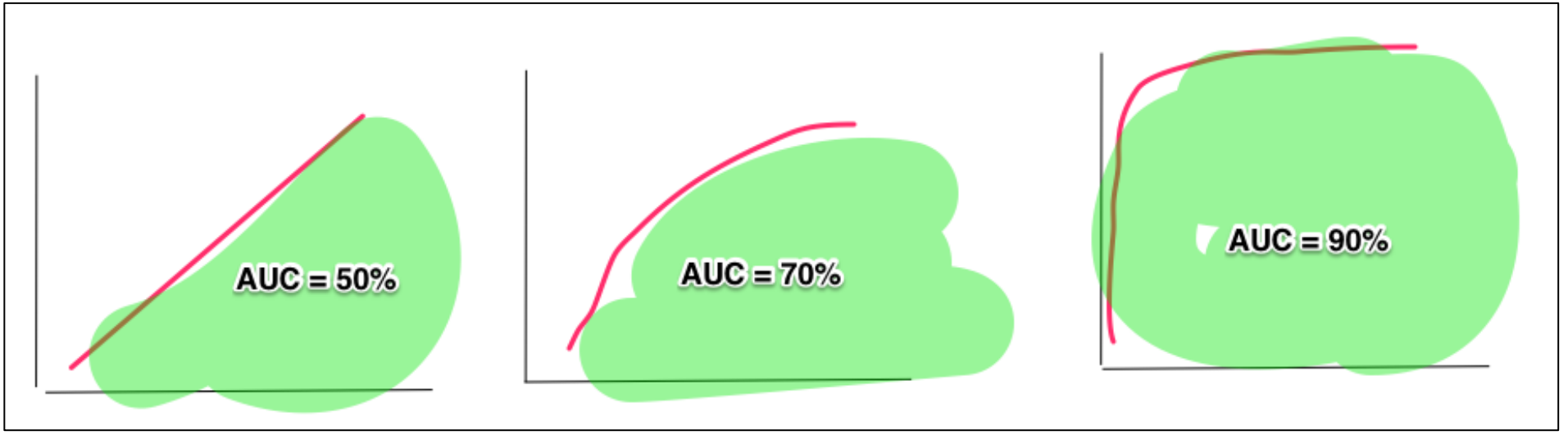
- Shows tradeoff of TPR (sensitivity) vs. FPR (1 – specificity)
- The closer to top-left , the more accurate the model
- Upper left corner (0,1) = perfect classification!
- The closer to middle line (45 degree) the less accurate the test
  - Middle line represents : random classification (50%)



Area Under Curve – AUC (ROC Space)

- Measures the percentage of area 'under the curve'
- AUC is between 0 and 1
- Higher AUC → more accurate the model
- See 3 scenarios below
  - Left most is bad (50%)
  - Middle : OK (70%)
  - Right most : very good (90%)





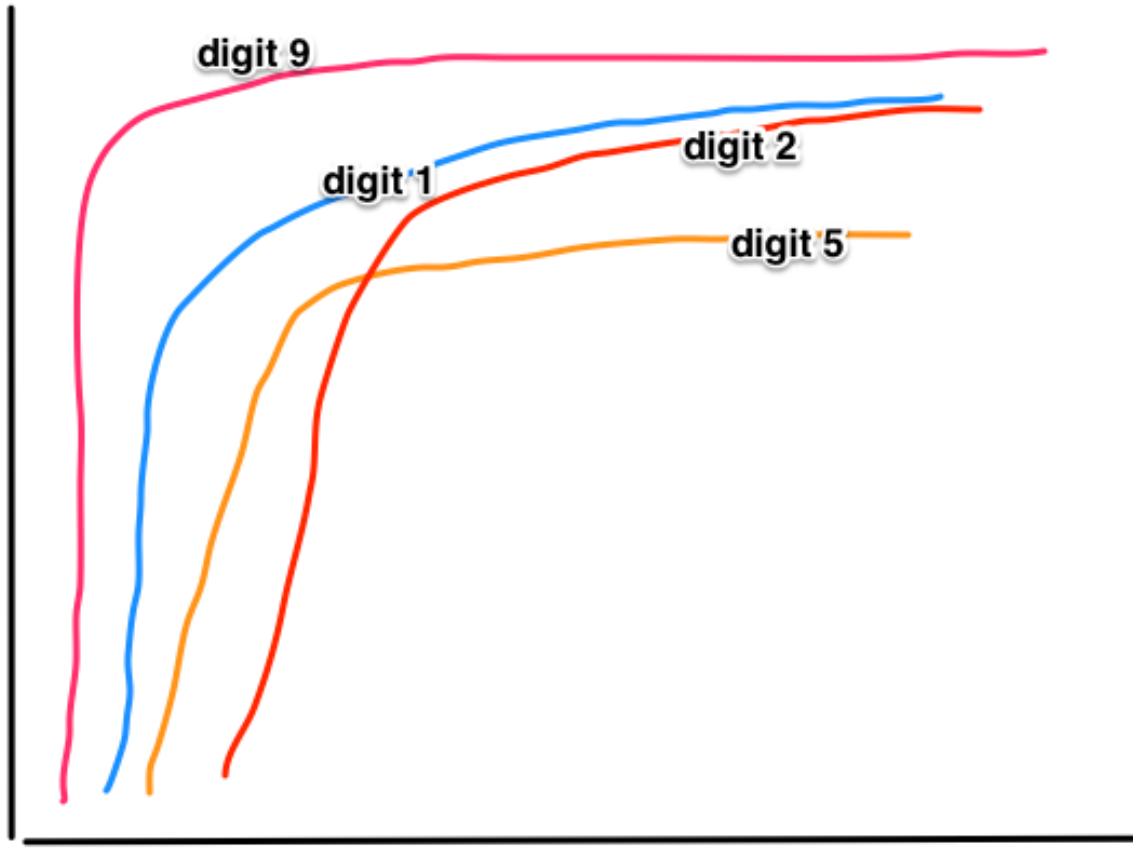
Using AUC to Measure Accuracy

- Accuracy can be specified using a grading system

AUC	Grade
0.9 – 1.00	A - Excellent
0.80 – 0.90	B - good
0.70 - 0.80	C - fair
0.60 – 0.70	D - poor
0.50 – 0.60	F - Fail

ROC / AUC For Multiclass Classifiers

- Say our algorithm recognizes hand-written digits (postal code) into numbers.
- Its ROC can be drawn as follows



Lab: Classification

- **Overview**: In this lab, we will use the "nycflights13" dataset for predicting delayed flights
  - We will use logistic regression as a
- **Builds on previous labs**: None
- **Approximate time**: 25-35 min.

Recommendations

Introduction

Supervised Machine Learning

Clustering

Classification/Regression

Recommendations

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

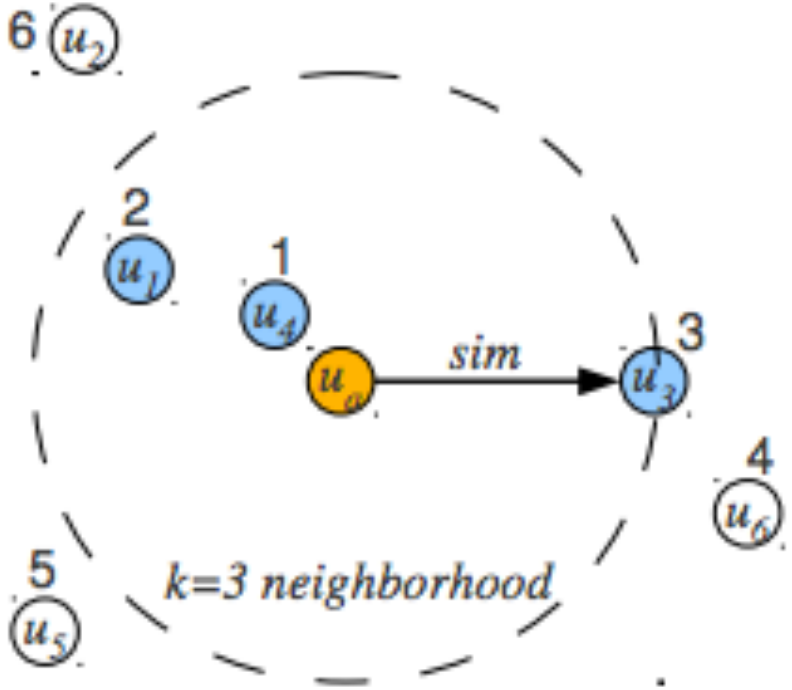
Collaborative Filtering

- Collaborative Filtering (CF) is commonly used in “Recommended For You” or “More Like This” functionality
- Recommendations can be explicit (based on user ratings), or implicit (based on user interest)
- CF is expressed as Users -> Items
  - However, any correlation could be modeled as users to items
- Users and Items could be the same (example: dating site)

Ratings Matrix: Users/Movies

	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8
u_1	?	4.0	4.0	2.0	1.0	2.0	?	?
u_2	3.0	?	?	?	5.0	1.0	?	?
u_3	3.0	?	?	3.0	2.0	2.0	?	3.0
u_4	4.0	?	?	2.0	1.0	1.0	2.0	4.0
u_5	1.0	1.0	?	?	?	?	?	1.0
u_6	?	1.0	?	?	1.0	1.0	?	1.0
u_a	?	?	4.0	3.0	?	1.0	?	5.0
r_a	3.5	4.0			1.3		2.0	

(a)



(b)

Item Ratings Matrix – Normalized

S	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	u_a
i_1	-	0.1	0	**0.3**	**0.2**	**0.4**	0	0.1	2
i_2	0.1	-	**0.8**	**0.9**	0	**0.2**	0.1	0	?
i_3	0	**0.8**	-	0	**0.4**	0.1	0.3	**0.5**	?
i_4	**0.3**	**0.9**	0	-	0	**0.3**	0	0.1	?
i_5	**0.2**	0	**0.7**	0	-	**0.2**	0.1	0	4
i_6	**0.4**	**0.2**	0.1	**0.3**	0.1	-	0	0.1	?
i_7	0	**0.1**	**0.3**	0	0	0	-	0	?
i_8	**0.1**	0	**0.9**	**0.1**	0	0.1	0	-	5
r_a	-	0.0	4.6	2.8	-	2.7	0.0	-	

Recommendations in Python

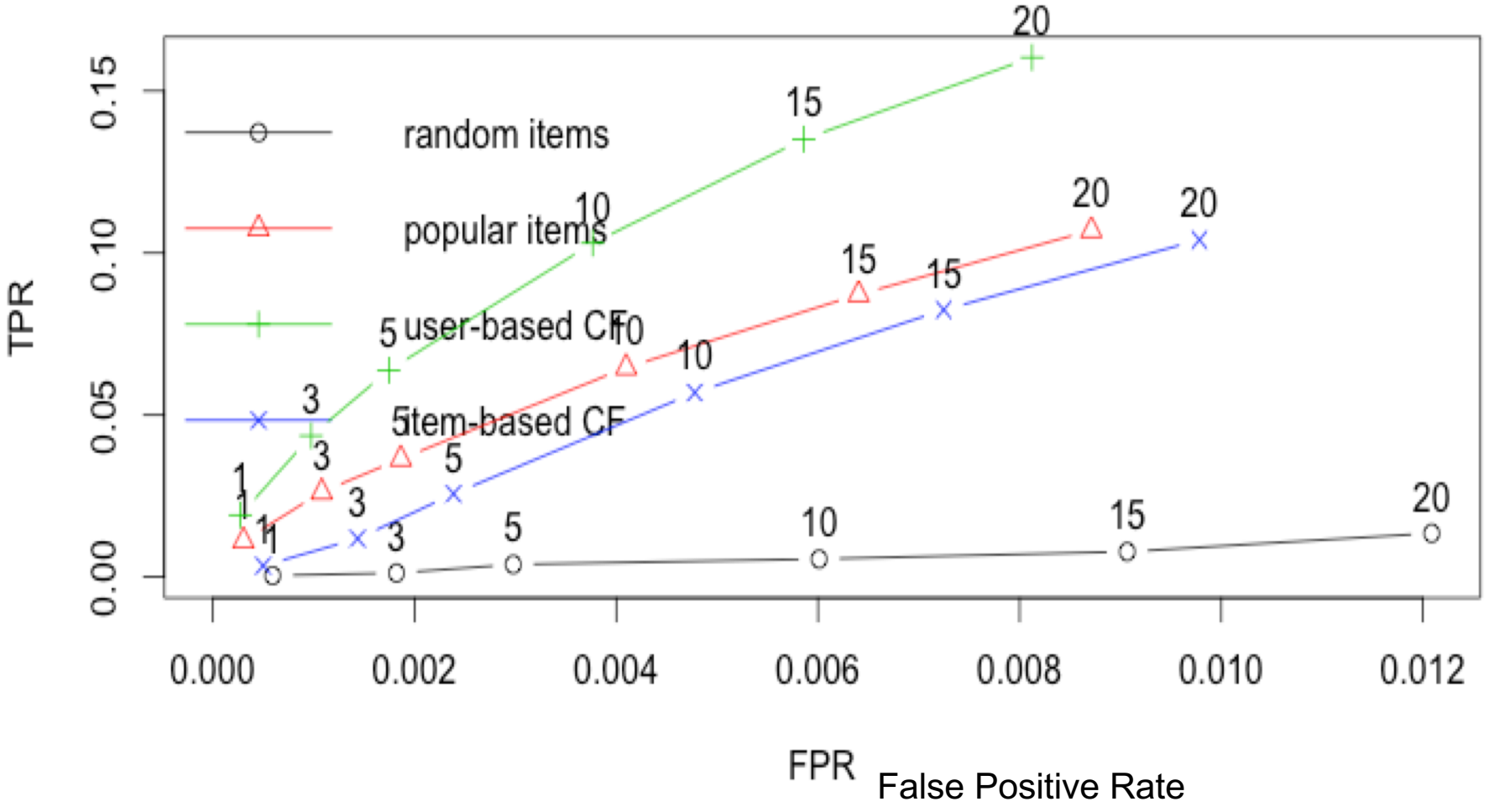
- There is a specialized library called **scikit-surprise** for Recommendations
- Has several ways of performing recommendations
  - SVD (Singular Value Decomposition)
  - ALS (approximation of SVD)
  - K-Nearest Neighbors

Preparing Data for Scikit-surprise

- Scikit-surprise only looks at integer userid and itemid.
  - Assign your data with unique integer userid and itemid.
- Preferences are expressed as a double (higher is better)
  - What if your preferences are binary (yes/no)?
    - Assign a number, say, 5.0 for yes, and 1.0 for no.
  - What if preferences are just “implied”?
    - User viewed item as 5.0, no data for unviewed item.
- Scikit-learn Rating: Integer, Integer, Double

Evaluating CF Recommendations (ROC)

True Positive Rate



Lab : Recommendations

- **Overview**: We will use Movielens example
  - There are a number of TODOs to finish in the code
- **Builds on previous labs**: None

- **Approximate time**: 20-30 min.
- Follow: **6-Scikit-learn/recs/README.md**