Data Science
Feature Engineering: Feature Creation and Feature Selection

Won Kim

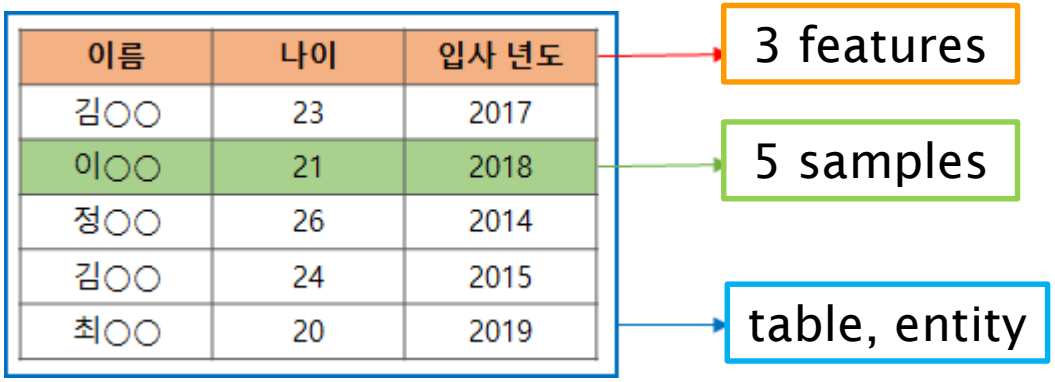
2022

Roadmap: Data Preprocessing

- Data Restructuring
- Data Value Changes
- Feature Engineering
- Data Reduction

Reminder: Key Terms

- feature – variable, column in a table, dimension
- observations, sample – rows, records



Feature Engineering

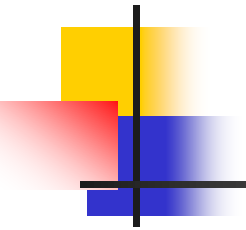
- Feature creation
  - Process of acquiring and creating all features relevant to achieving the objective of a data science project.
- Feature selection
  - Process of selecting only the most relevant features from a given dataset.
- Feature reduction
  - Process of reducing the number of features by transforming data so that much of the information is concentrated in a small number of features.

Critically Important, But Very Difficult

- Feature engineering requires domain experts and expert data scientists.
- Data scientists must examine potentially very many features (including combinations of features), and determine all features relevant to a data science project.
- They need to use statistics techniques and software tools, including Python Featuretools library.
- However, human experts must make the final feature selection, by trial and error.

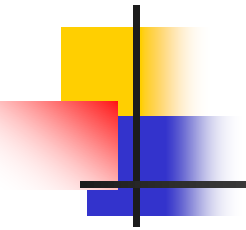
Roadmap: Feature Engineering

- Feature Creation
- Feature Selection
- Feature Reduction



Feature Creation

- Deriving Features from Existing Features
- Converting Values to New Features



Deriving Features

- Create a new feature from an existing feature(s) in the same entity (i.e., RDB table, Excel worksheet)
- Create a new feature by merging or aggregating over two related, but different entities.

Derive New Features from Existing Ones (within the same entity) (1/4)

- https://openclassrooms.com/en/courses/6389626-train-a-supervised-machine-learning-model/6398776-create-new-features-from-existing-features
- Create a new feature by binning (categorizing) an existing feature.
  - (e.g., from the "happiness" numerical feature, the new "happiness-band" categorical feature is created.)

country	happiness	happiness_band
Belarus	5.718908	M
Brazil	6.546897	H
Malawi	3.867638	L
Mozambique	4.549767	L
Peru	5.577263	M
Australia	7.309061	H

Derive New Features from Existing Ones (within the same entity) (2/4)

- Create a new feature by binning (categorizing) an existing feature.
  - (e.g., from the "country" categorical feature, the new "region" categorical feature is created.)

country	happiness	region
Belarus	5.718908	Europe & Central Asia
Brazil	6.546897	Latin America & Caribbean
Malawi	3.867638	Sub-Saharan Africa
Mozambique	4.549767	Sub-Saharan Africa
Peru	5.577263	Latin America & Caribbean
Australia	7.309061	East Asia & Pacific

Derive New Features from Existing Ones (within the same entity) (3/4)

- Given a time and date feature,

borough	timestamp_of_call
Richmond Upon Thames	1/14/17 23:11
Waltham Forest	24/01/1017 07:56
Hackney	2/12/17 11:35
Brent	1/30/17 23:55
Redbridge	2/21/17 1:25

- Split it into new features (year, month, day, date, hour and minute) for a more meaningful analysis.

borough	day	month	year	weekday	hour	minute
Richmond Upon Thames	14	1	2017	7	23	11
Waltham Forest	24	1	2017	3	7	56
Hackney	12	2	2017	1	11	35
Brent	30	1	2017	2	23	55
Redbridge	21	2	2017	3	1	25

Derive New Features from Existing Ones (within the same entity) (4/4)

- From a time-date feature and a numerical feature ("joined" and "income")

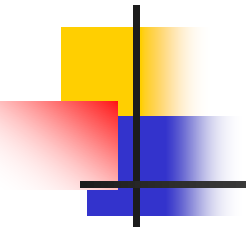
client_id	joined	income	credit_score
46109	2002-04-16	172677	527
49545	2007-11-14	104564	770
41480	2013-03-11	122607	585
46180	2001-11-06	43851	562
25707	2006-10-06	211422	621

- Create two new features ("joined_month" and "log_income")

client_id	joined	income	credit_score	join_month	log_income
46109	2002-04-16	172677	527	4	12.059178
49545	2007-11-14	104564	770	11	11.557555
41480	2013-03-11	122607	585	3	11.716739
46180	2001-11-06	43851	562	11	10.688553
25707	2006-10-06	211422	621	10	12.261611

Merging or Aggregating Over Different Entities

- Machine learning algorithms accept only a single entity (table).
- This means logically connected multiple entities in an RDB cannot be used as input to machine learning algorithms. They must first be merged into a single entity.
- The objective of merging or aggregating over different entities is to consolidate the results into a single entity.



Terminology

- entity – Pandas Data Frame, RDB table
- entity set – set of entities, RDB tables
- shared variable – RDB foreign key
- transformation – deriving a new feature from existing features within an entity
- aggregation – merging different entities, or calculating statistics on a group of rows in an entity and merging the result as a feature of another entity

Example Entities (1/2): Clients

- 'client_id' is the primary key.

client_id	joined	income	credit_score
46109	2002-04-16	172677	527
49545	2007-11-14	104564	770
41480	2013-03-11	122607	585
46180	2001-11-06	43851	562
25707	2006-10-06	211422	621

Example Entities (2/2): Loans

- Note: Same client may have multiple loans
  'client_id' is shared with the Clients entity.

client_id	loan_type	loan_amount	repaid	loan_id	loan_start	loan_end	rate
25707	other	9942	1	10438	2009-03-26	2010-10-22	2.39
39384	other	13131	1	10579	2012-08-12	2014-06-17	2.95
49624	other	2572	1	10578	2004-05-04	2005-12-16	2.28
29841	credit	10537	1	10157	2010-08-04	2013-03-11	3.43
39505	other	6484	1	10407	2011-02-14	2012-12-07	1.14
44601	home	4475	1	10362	2005-07-29	2007-07-06	6.58
39384	credit	1770	1	10868	2013-08-03	2016-04-28	2.64
48177	other	1383	0	11264	2009-08-08	2012-01-03	5.69
32885	home	11783	0	10301	2000-08-10	2003-03-12	2.64
49068	cash	6473	1	11546	2002-09-01	2004-10-23	5.18

Derive New Features by Aggregating Over Two Different Entities

- Aggregate over one entity (e.g., loans in the loan entity for the same client), compute statistics (e.g., mean, max, min of the loans), and add the result as new features of a different entity (e.g., the client in the client entity).

client_id	joined	income	credit_score	join_month	log_income	mean_loan_amount	max_loan_amount	min_loan_amount
46109	2002-04-16	172677	527	4	12.059178	8951.600000	14049	559
49545	2007-11-14	104564	770	11	11.557555	10289.300000	14971	3851
41480	2013-03-11	122607	585	3	11.716739	7894.850000	14399	811
46180	2001-11-06	43851	562	11	10.688553	7700.850000	14081	1607
25707	2006-10-06	211422	621	10	12.261611	7963.950000	13913	1212
39505	2011-10-14	153873	610	10	11.943883	7424.050000	14575	904
32726	2006-05-01	235705	730	5	12.370336	6633.263158	14802	851
35089	2010-03-01	131176	771	3	11.784295	6939.200000	13194	773
35214	2003-08-08	95849	696	8	11.470529	7173.555556	14767	667
48177	2008-06-09	190632	769	6	12.158100	7424.368421	14740	659

Example Entities: (Loan)Payments

- Note: Same loan may have multiple payments.
  'loan_id' is shared with the Loans entity.

loan_id	payment_amount	payment_date	missed
10302	489	2006-06-17	1
11652	1896	2014-08-17	0
11827	2755	2005-02-26	1
10078	624	2005-05-28	1
10177	1474	2002-05-03	0
10660	701	2005-09-22	1
11251	568	2000-08-27	0
10826	2538	2005-03-20	0
11896	1055	2004-06-02	0
10742	437	2005-06-04	1

Featuretools Python Library

- A tool that helps feature creation through feature transformation and feature aggregation
- https://towardsdatascience.com/automated-feature-engineering-in-python-99baf11cc219

Create an (empty) Entityset

```
import pandas as pd
import numpy as np
import featuretools as ft

# Create new entityset
es = ft.EntitySet(id = 'clients')
```

Add an Entity to an Entity Set (1/3)

- Each entity must have an index.
- The index in the clients Data Frame is the client_id.
- Add an entity with an existing index to an entityset:

```
# Create an entity from the client dataframe
# This dataframe already has an index and a time index
# index=RDB primary key
# time index= date_time feature

es = es.entity_from_dataframe(entity_id = 'clients',
dataframe = clients, index = 'client_id', time_index =
'joined')
```

Add an Entity to an Entity Set (2/3)

- The Loans Data Frame also has an index on loan_id and is added to the entityset similarly.

Add an Entity to an Entity Set (3/3)

- There is no index for the Payments Data Frame.
- When we add this to the entityset, we need to pass the parameter make_index = True and specify the name of the index.
- featuretools can infer the data type of each column in an entity. But we can override this by passing column types to the parameter variable_types .

```
# Create an entity from the payments dataframe
# This does not have an index
es = es.entity_from_dataframe(entity_id = 'payments',
                              dataframe = payments,
                              variable_types = {'missed':
                                    ft.variable_types.Categorical},
                                make_index = True,
                                index = 'payment_id',
                                time_index = 'payment_date')
```

Inspecting the Entity

```
In [16]: es['payments']
```

```
Out[16]: Entity: payments
           Variables:
             loan_id (dtype: numeric)
             payment_amount (dtype: numeric)
             payment_date (dtype: datetime_time_index)
             missed (dtype: categorical)
             payment_id (dtype: index)
           Shape:
             (Rows: 3456, Columns: 5)
```

Aggregation (1/2)

- # import pandas as pd

- # Group loans by client id and calculate mean, max, min of loans
- stats = loans.groupby('client_id')['loan_amount'].agg(['mean', 'max', 'min'])
- stats.columns = ['mean_loan_amount', 'max_loan_amount', 'min_loan_amount']

- # Merge with the clients dataframe
- stats = clients.merge(stats, left_on = 'client_id', right_index=True, how = 'left')

- stats.head(10)

Create Relationships Between Entities

- Relationship between two entities
  - parent to child, one-to-many relationship
  - Each row in the parent entity can have multiple rows in the child entity as children.
- For example, Clients is the parent of Loans.
  - Each client may have multiple loans.
- Likewise, for Loans and Payments.
- The parents are linked to their children by a shared variable. We group the rows in a child entity by the parent (shared) variable and calculate statistics on them.

Example

```
# Create a relationship between clients and loans
r_client_previous = ft.Relationship(es['clients']['client_id'],
                                    es['loans']['client_id'])
# Add the relationship to the entity set
es = es.add_relationship(r_client_previous)

# Relationship between previous loans and previous payments
r_payments = ft.Relationship(es['loans']['loan_id'],
                             es['payments']['loan_id'])
# Add the relationship to the entity set
es = es.add_relationship(r_payments)
es
```

Result

```
Entityset: clients
  Entities:
    clients [Rows: 25, Columns: 6]
    loans [Rows: 443, Columns: 8]
    payments [Rows: 3456, Columns: 5]
  Relationships:
    loans.client_id -> clients.client_id
    payments.loan_id -> loans.loan_id
```

dfs function

- New features are created in featuretools using the feature primitives or stacking multiple primitives.
- The ft.dfs function is used to create features using specific primitives.
- dfs stands for ‘deep feature synthesis’. (It has nothing to do with deep learning. ‘deep’ here means ‘stacked’)

Some of the Feature Primitives

name	type	description
num_true	aggregation	Finds the number of 'True' values in a boolean.
percent_true	aggregation	Finds the percent of 'True' values in a boolean feature.
time_since_last	aggregation	Time since last related instance.
num_unique	aggregation	Returns the number of unique categorical variables.
avg_time_between	aggregation	Computes the average time between consecutive events.
all	aggregation	Test if all values are 'True'.
min	aggregation	Finds the minimum non-null value of a numeric feature.
mean	aggregation	Computes the average value of a numeric feature.
seconds	transform	Transform a Timedelta feature into the number of seconds.
second	transform	Transform a Datetime feature into the second.
and	transform	For two boolean values, determine if both values are 'True'.
month	transform	Transform a Datetime feature into the month.
cum_sum	transform	Calculates the sum of previous values of an instance for each value in a time-dependent entity.
percentile	transform	For each value of the base feature, determines the percentile in relation
time_since_previous	transform	Compute the time since the previous instance.
cum_min	transform	Calculates the min of previous values of an instance for each value in a time-dependent entity.

Example

- We pass the entityset, the target_entity, which is the entity where we want to add the features, the selected trans_primitives (transformations), and agg_primitives (aggregations).

```
# Create new features using specified primitives

features, feature_names = ft.dfs(entityset = es,
target_entity = 'clients',
                      agg_primitives = ['mean', 'max',
'percent_true', 'last'],
                      trans_primitives = ['years', 'month',
'subtract', 'divide'])
```

Result (1/3)

- The result is a Data Frame of new features for each client.
- For example, we have the month each client joined which is a transformation feature primitive.

client_id	MONTH(joined)
25707	10
26326	5
26695	8
26945	11
29841	8

Result (2/3)

- We also get a number of aggregation primitives, such as the average payment amounts for each client.

client_id	MEAN(payments.payment_amount)
25707	1178.552795
26326	1166.736842
26695	1207.433824
26945	1109.473214
29841	1439.433333

Result (3/3)

- The resulting Data Frame has 793 new features!
- Even though we specified only a few feature primitives, featuretools created many new features by combining and stacking these primitives.

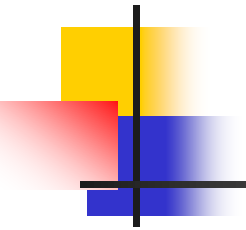
client_id	MEAN(loans.loan_amount)	MEAN(loans.rate)	MAX(loans.loan_amount)	MAX(loans.rate)	LAST(loans.loan_type)	LAST(loans.loan_amount)
25707	7963.950000	3.477000	13913	9.44	home	2203
26326	7270.062500	2.517500	13464	6.73	credit	5275
26695	7824.722222	2.466111	14865	6.51	other	13918
26945	7125.933333	2.855333	14593	5.65	cash	9249
29841	9813.000000	3.445000	14837	6.76	home	7223

Homework

- Using the same Pandas Data Frames, repeat the Featuretools Python library walkthrough example, with just one difference.
  - Instead of creating three new features for the Clients entity (min_loan_amount,max_loan_amount, and mean_loan_amount), create just one new feature (total_loan_amount).
- Submit the result to CyberCampus in a single WORD file.

Roadmap: Feature Creation

- Deriving Features from Existing Features
- Converting Values to New Features



Converting Values to Features

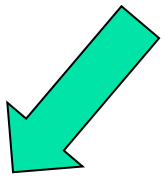
- Convert a categorical feature with N values to N new features (with numerical values)
- Convert a corpus of N words into an N-element vector
- Convert each pixel in an image of N pixels to a new feature

Convert Categorical Data Values to Features (1/2)

- Categorical feature values need to be converted to numbers. Machine Learning algorithms can only accept numbers as input.

date	temperature	weather
Jan 1	-3	sunny
Jan 2	2	cloudy
Jan 3	-5	snow

Weather:
{sunny, cloudy, snow, rain)



Date	temp	w-sunny	w-cloudy	w-snow	w-rain
Jan 1	-3	1	0	0	0
Jan 2	2	0	1	0	0
Jan 3	-5	0	0	1	0

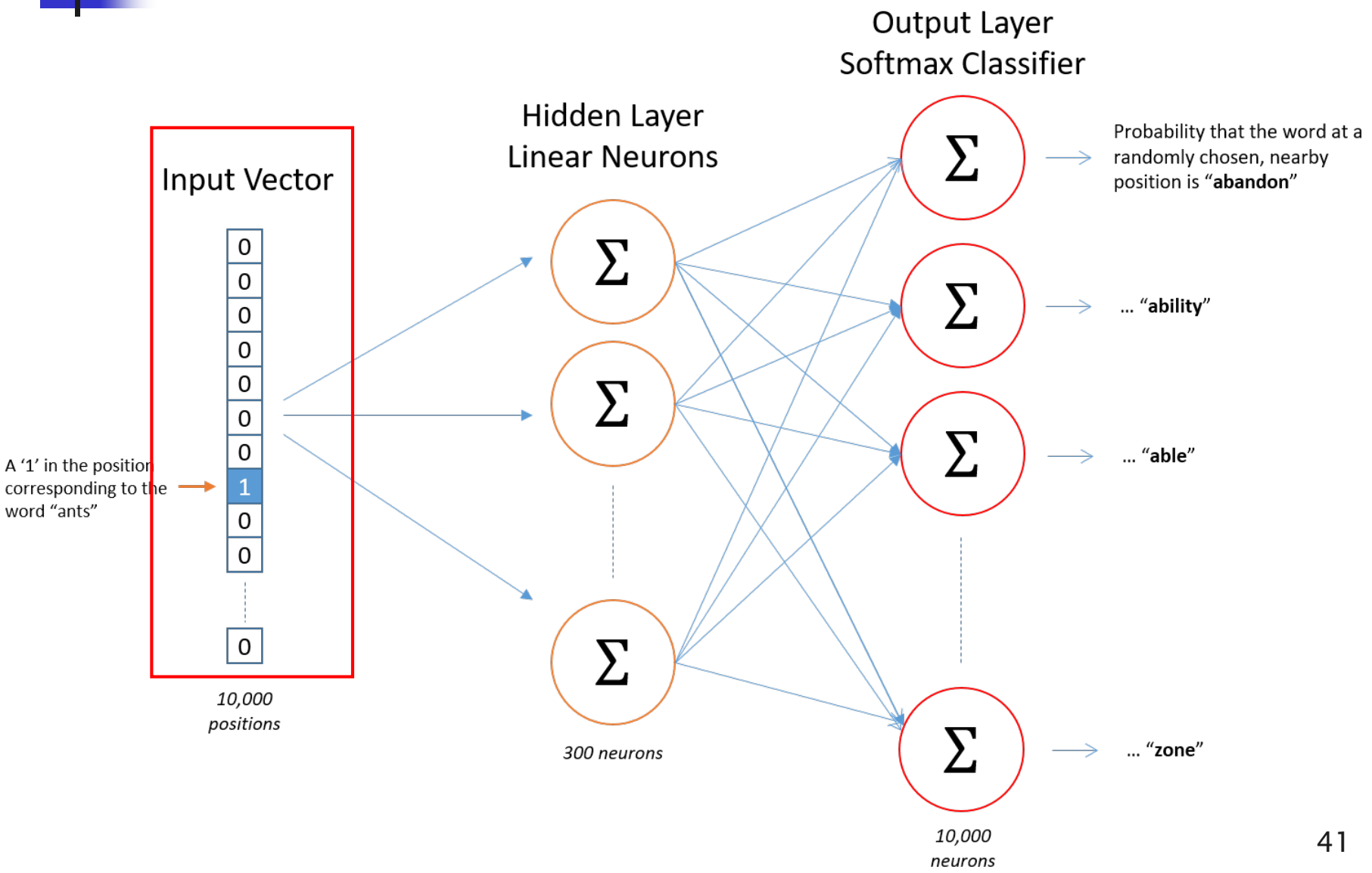
Convert Categorical Data Values to Features (2/2)

- Called “One-Hot Encoding”
- A categorical feature with N possible values is expanded to N new features (i.e., RDB columns)
- Each new feature can have either 0 or 1 as values.
- data mining/data mining algorithms cannot handle categorical “text” data, and so this conversion is required.
- Causes two problems
  - increase in the number of features (and compute time)
  - sparse matrix (all those 0’s take up storage/memory)

Convert a Word in a Corpus into a Vector (1/2)

- For text processing (e.g., next word prediction), we need to convert each word into a vector.
- For example, we have (e.g.) 10,000 unique words in a corpus.
  - Each word is one-hot encoded in a 10,000-element vector (array).
  - The vector is input to a neural network for Deep Learning.

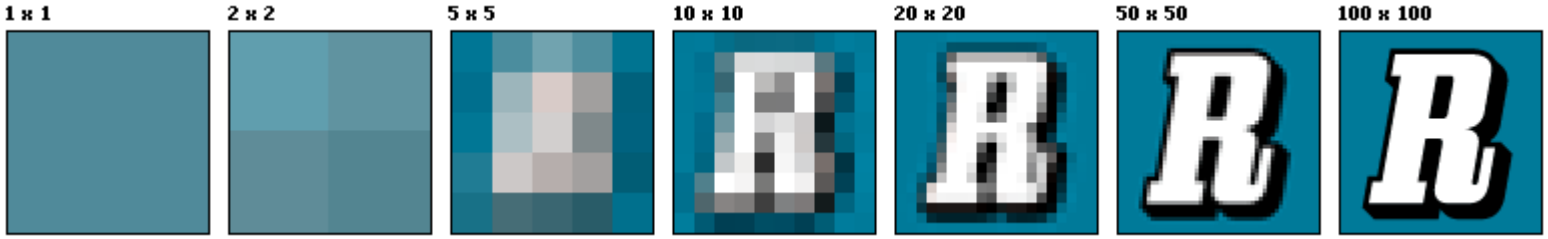
Convert a Word in a Corpus into a Vector (2/2)



Convert Each Image Pixel to a Feature

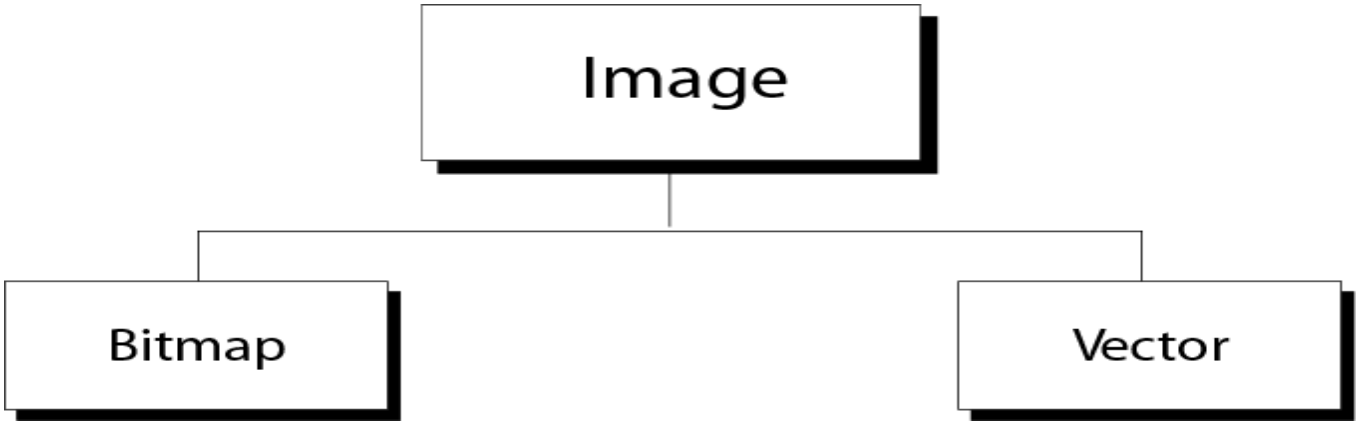
- An image with N pixels is expanded to N new features (i.e., RDB columns).
- Each pixel is a binary pattern (i.e., number).
- Data mining algorithms cannot handle "image" data, and so this conversion is required.
- "N" is large (10,000, 1,000,000,…), and causes two big problems
  - GIGANTIC increase in the number of features
  - Many features with the same or very similar data take up storage/memory

Background: Pixels (Picture Elements)



- Pixel counts can be expressed as a single number
  - (e.g.) A "three-megapixel" digital camera has a nominal three million pixels.
- Or they can be expressed as a pair of numbers
  - A "640 by 480 display", which has 640 pixels from side to side, and 480 from top to bottom
  - Has total 640×480 = 307,200 pixels or 0.3 megapixels.

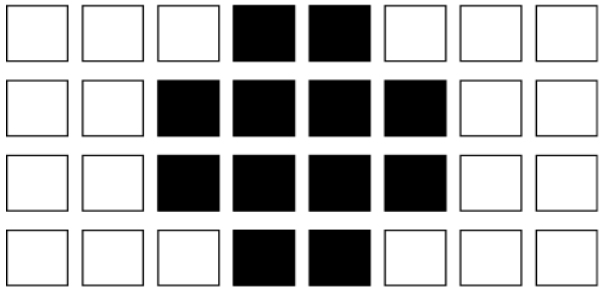
Background:
Image Representation Methods



- Bitmap graphic method
  - divides an image into pixels (picture elements)
- Vector graphics
  - represents and stores an image as a mathematical formula (combination of curves and lines).

Background:
Bitmap Graphic Representation of Black and White Pixels

- Each pixel is assigned a bit pattern (0 for white, 1 for black)



Image

0 0 0 1 1 0 0 0
0 0 1 1 1 1 0 0
0 0 1 1 1 1 0 0
0 0 0 1 1 0 0 0

Matrix Representation

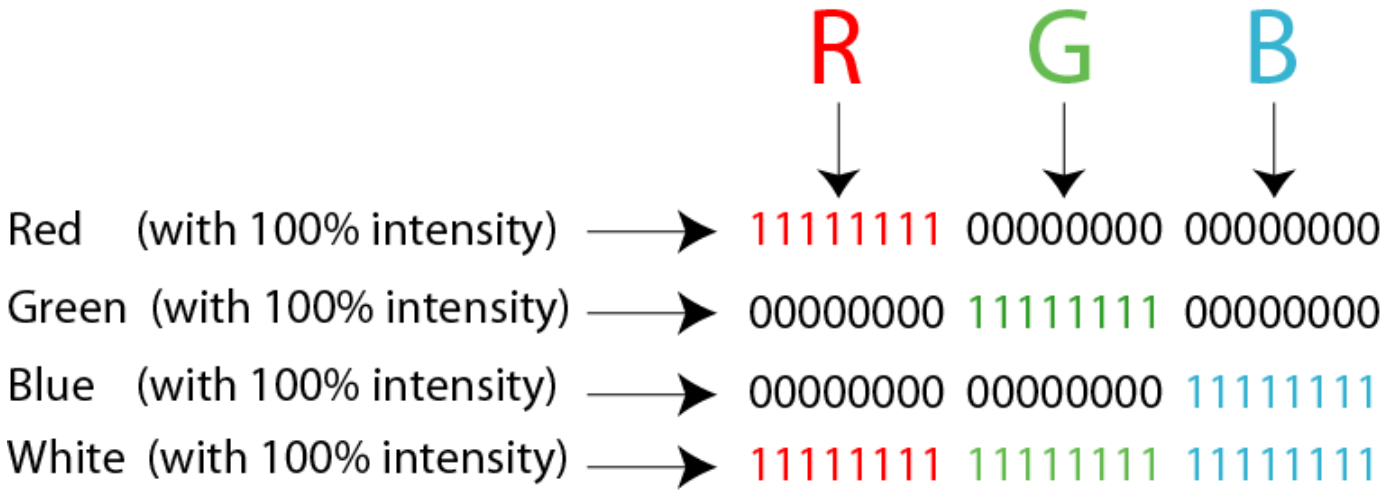
00011000 00111100 00111100 00011000

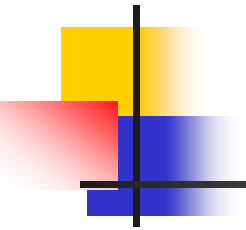
Linear Representation

- To show 4 levels of gray scale, a pixel is represented by a 2-bit pattern {00 black, 01 dark gray, 10 light dark gray, 11white}

Background: Representation of Color Pixels

- Each colored pixel is decomposed into red, green, blue.
- Intensity of each color is measured and a bit pattern (usually 8-bit) is assigned to it.
- 24 bits per pixel x total number of pixels = total number of features !!





Roadmap: Feature Engineering

- Feature Creation
- Feature Selection
- Feature Reduction

Curse of Dimensionality (1/2)

- Often, the dataset contains too many features.

  (100s, 1000s, 1000000s)
  - feature = dimension
- Many features mean huge computation (training) time and computing resources.
- It also makes it impossible to visualize them, or interpret (make sense of) them.

Illustration

- In the following product dataset, price is the target variable, and all others are independent variables.

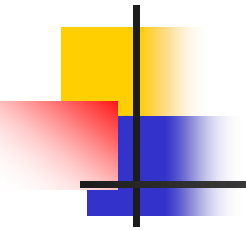
ram size	camera pixel	n_cores	price

- We want to determine which independent variables affect the price most.
- Now suppose the number of independent variables is 100. Can you imagine the difficulties this will cause?

Curse of Dimensionality (2/2)

(* don't worry about the following for now *)

- As the number of features increases,
  - the learning models become more complex, and may result in the overfitting of the learning models (i.e. decreased accuracy)
  - data becomes sparse, and sparse data makes it difficult to achieve statistical significance for many data mining methods
  - definitions of density and distance (critical for clustering and other methods) become less useful
    - All distances start to converge to a common value



Feature Selection

- Objective is to select k, a minimum subset of the original set of N features in the dataset.
- Feature selection includes two primary components.
  - Search technique for proposing new feature subsets
  - Evaluation measure for scoring the feature subsets

Feature Selection Methods

- Filter Methods
  - Selects the best feature subset before data mining algorithm is run
  - (i.e.) feature selection and machine learning are separate.
- Wrapper Methods
  - Use a data mining algorithm to do the feature selection
  - (After feature selection, a different data mining algorithm suitable for the data science project is run.)
- Embedded Methods
  - Feature selection occurs as part of a machine learning algorithm chosen for the data science project
  - example: L1-regularized linear regression

Feature Selection Techniques with Python

- https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e
- Three of the techniques
  - Univariate selection
  - Feature importance scoring
  - Correlation matrix with heatmap
- These compute/visualize correlation between independent variables (features) and the target variable (feature); that is, how important a feature is to predicting the target.

Example Dataset:
mobile price prediction dataset

- price_range: This is the target variable with a value of 0(low cost), 1(medium cost), 2(high cost) and 3(very high cost).
- battery_power: total energy a battery can store in one time measured in mAh
- blue: has Bluetooth or not
- clock_speed: the microprocessor speed
- dual_sim: has dual sim support or not
- fc: front camera megapixels
- four_g: has 4G or not
- int_memory: main memory in Gigabytes
- m_dep: mobile depth in cm
- mobile_wt: weight of mobile phone
- n_cores: number of cores of the processor
- pc: primary camera megapixels
- px_height
- pixel resolution height
- px_width: pixel resolution width
- ram: random access memory in megabytes
- sc_h: screen height of mobile in cm
- sc_w: screen width of mobile in cm
- talk_time: longest time that a single battery charge will last
- three_g: has 3G or not
- touch_screen: has touch screen or not
- wifi: has WiFi or not

Univariate Selection

- univariate data → one-dimensional data
  - (e.g., exam scores, people's ages, ···)
- The Scikit-learn library provides the SelectKBest class for use with some statistical tests to select a specified number of features.
- The example below uses the chi-squared (chi^2) statistical test to select 10 of the best features from the example dataset.

Code Example (1/3)

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# download data
from   https://www.kaggle.com/iabhishekofficial/mobile-price-classification#train.csv

data = pd.read_csv("D://Blogs//train.csv")
X = data.iloc[:,0:20]  #independent columns
y = data.iloc[:,-1]    #target column, i.e., price range
```

Code Example (2/3)

```
#apply SelectKBest class to extract top 10 best features

bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,y)
dfcolumns = pd.DataFrame(X.columns)
dfscores = pd.DataFrame(fit.scores_)

#concatenate two dataframes for better visualization

featureScores = pd.concat([dfcolumns, dfscores],axis=1)
featureScores.columns = ['Specs',' Score']  #name the
        dataframe columns

print(featureScores.nlargest(10,'Score'))  #print 10 best
        features
```

Code Example (3/3): Result

	Specs	Score
13	ram	931267.519053
11	px_height	17363.569536
0	battery_power	14129.866576
12	px_width	9810.586750
8	mobile_wt	95.972863
6	int_memory	89.839124
15	sc_w	16.480319
16	talk_time	13.236400
4	fc	10.135166
14	sc_h	9.614878

Feature Importance Scoring

- Feature importance gives you a score for each feature of your data.
  - The higher the score more important or relevant is the feature towards your target variable.
- Feature importance is a built-in class that comes with tree-based classifiers (e.g. decision tree, random forest).
- In the example below, we use ExtraTreesClassifier for extracting the top 10 features for the dataset.

Code Example (1/3)

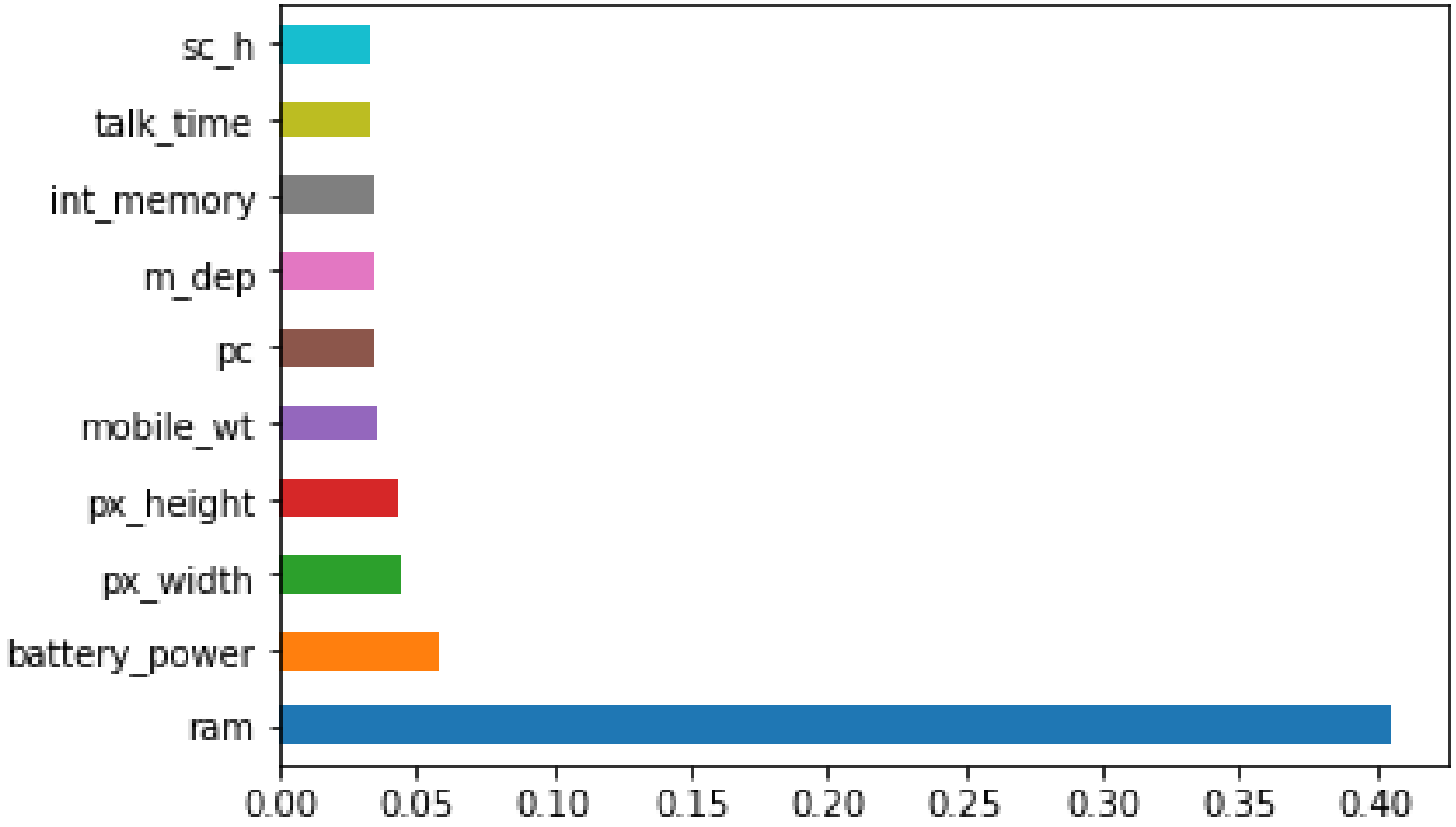
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv("D://Blogs//train.csv")
X = data.iloc[:,0:20]  #independent columns
y = data.iloc[:,-1]    #target column, i.e., price range
```

Code Example (2/3)

```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(X,y)

print(model.feature_importances_) #use built-in class
      feature_importances of tree-based classifiers

#plot graph of feature importances for better visualization

feat_importances = pd.Series(model.feature_importances_,
index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')

plt.show()
```

horizontal bar

Code Example (3/3): Result



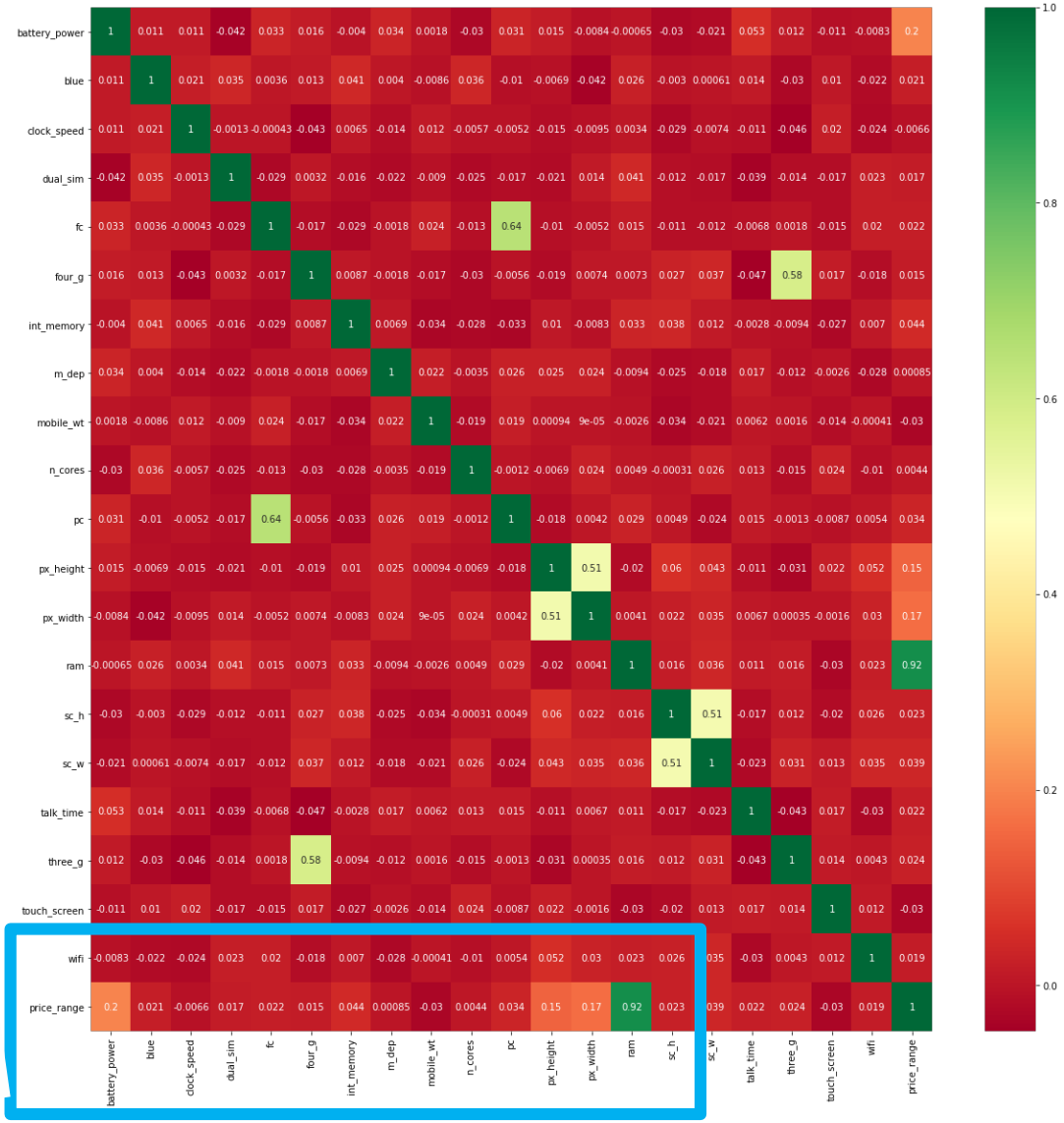
Correlation Matrix with Heatmap

- Heatmap makes it easy to identify which variables are most related to the target variable.
- The heatmap provides a 2-dimensional colored visual summary of data.
  - https://likegeeks.com/seaborn-heatmap-tutorial/
- We will plot heatmap of correlated variables using the seaborn library.
  - The seaborn library is built on top of Matplotlib.

Code Example

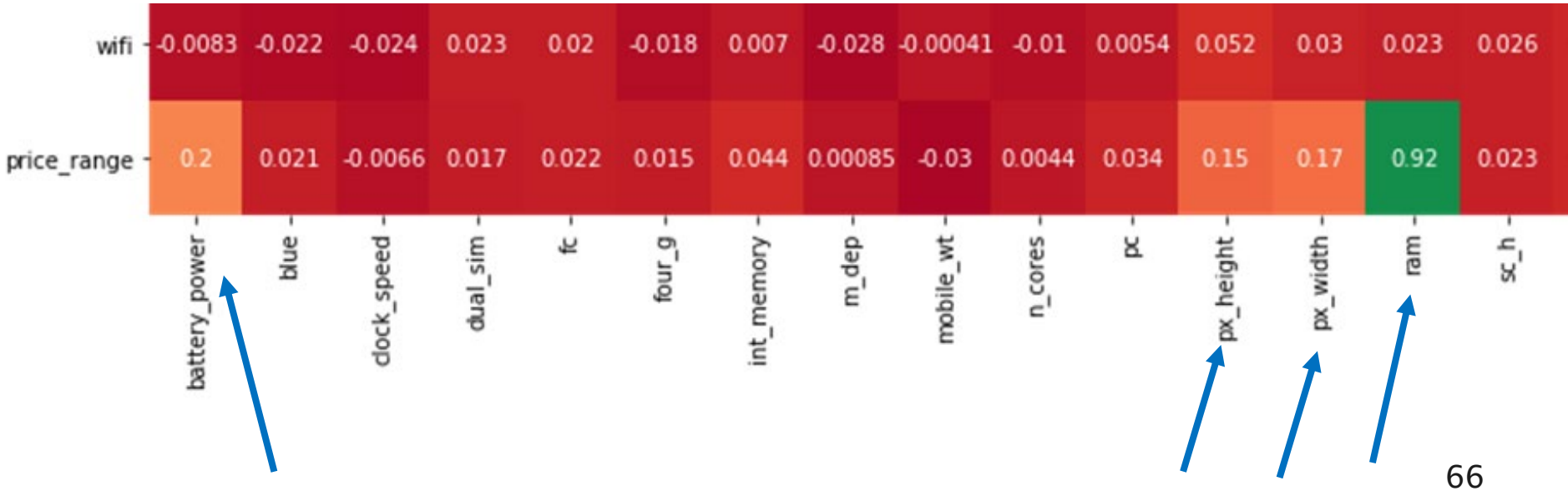
```
import pandas as pd
import numpy as np
import seaborn as sns

data = pd.read_csv("D://Blogs//train.csv")
X = data.iloc[:,0:20]  #independent columns
y = data.iloc[:,-1]    #target column, i.e., price range

#get correlations of pairs of features in the dataset
corrmat = data.corr() #corr() computes pairwise
        correlations of features in a Data Frame
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot the heat map
g=sns.heatmap(data[top_corr_features].corr(),annot=
True,cmap="RdYlGn")
```

Code Example (1/2): Result



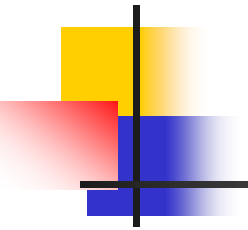
Code Example (2/2): Result (Zoomed)

- Note the last row (price range) shows how the price range is correlated with other features.
- ram is most highly correlated with price range, followed by battery power, pixel height and pixcel width.
- m_dep, clock_speed and n_cores are least correlated with price_range.



Homework

- Download the California housing prices dataset from Kaggle.
  - sklearn.datasets.fetch_california_housing(*, data_home=None, download_if_missing=True, return_X_y=False, as_frame=False)
- Do a univariate analysis, using the following techniques:
  - univariate selection with SelectKBest class
  - feature importance scoring
  - correlation matrix heatmap
- Submit the code and the analysis results.



End of Class