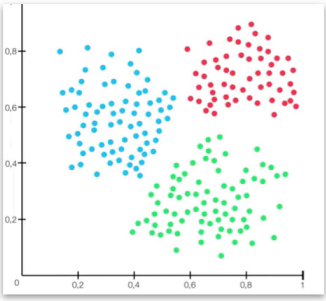
Whizlabs - ML Specialty Exam Course - Algorithms - part 2

Clustering algorithms

AWS Machine Learning - Clustering Algorithms

Clustering Algorithm - Defined

- Unsupervised learning algorithm
- Attempts to find discrete groupings within data, where members of a group are as similar as possible to one another and as different as possible from members of other groups
- Define the attributes that you want the algorithm to use to determine similarity



AWS Machine Learning - Clustering Algorithms

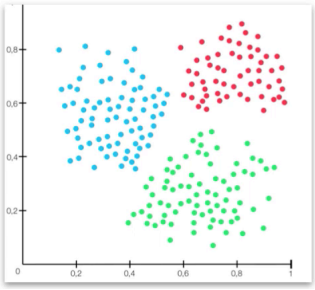
Example Clustering Algorithm Use Cases

- Delivery source location
- Identifying crime centers
- Customer segmentation
- Fraud detection based on clusters of fraud patterns
- Cyber-profiling criminals
- Clustering of IT alerts
- Call center recording analysis

K-means is the only clustering algorithm available in SageMaker so far

SageMaker Clustering Algorithms - K-Means

- K-Means
  - Expects tabular data, where rows represent the observations that you want to cluster, and the columns represent attributes of the observations
  - *n* attributes in each row represent a point in *n*-dimensional space
  - Euclidean distance between these points represents the similarity of the corresponding observations
  - Groups observations with similar attribute values (the points corresponding to these observations are closer together)
  - Example use case: using census data find clusters of populations in counties across the US to focus political activity



K-Means Lab

In [1]:
```
import os
import boto3
import io
import sagemaker

%matplotlib inline

import pandas as pd
import numpy as np
import mxnet as mx
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns
matplotlib.style.use('ggplot')
import pickle, gzip, urllib, json
import csv
from sagemaker import get_execution_role
role = get_execution_role()
s3 = boto3.resource('s3')
bucket_name = 'machine-learning-exam' # place the USCensus1990.data.csv file in a bucket in your account
object_key = 'Absenteeism_at_work.csv'
```

In [2]:
```
# Load our absenteeism data
s3_client = boto3.client('s3')
response = s3_client.get_object(Bucket=bucket_name, Key=object_key)
response_body = response["Body"].read()
absence = pd.read_csv(io.BytesIO(response_body), header=0, delimiter=",", low_memory=False)
```

In [3]:
```
absence.head()
```

Out[3]:

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	...	Disciplinary failure	Education	Son	Social drinker	Social smoker	Pet	Wei
0	11	26	7	3	1	289	36	13	33	239.554	...	0	1	2	1	0	1	
1	36	0	7	3	1	118	13	18	50	239.554	...	1	1	1	1	0	0	
2	3	23	7	4	1	179	51	18	38	239.554	...	0	1	0	1	0	0	
3	7	7	7	5	1	279	5	14	39	239.554	...	0	1	2	1	1	0	
4	11	23	7	5	1	289	36	13	33	239.554	...	0	1	2	1	0	1	

5 rows × 21 columns

In [4]:
```
# Drop the ID column
absence = absence.drop(['ID'], axis=1)
```

In [5]:
```
absence.shape
```

Out[5]:
```
(740, 20)
```

In [6]:
```
# Drop any missing value observations
absence.dropna(inplace=True)
absence.shape
```

Out[6]:
```
(740, 20)
```

Set the Reason for absence as the index

In [7]:
```
# Set the 'Reason for absence' as the index
# and the rest of the numerical features become the attributes of each unique absence
absence.index=absence['Reason for absence']
absence.head()
```

Out[7]:

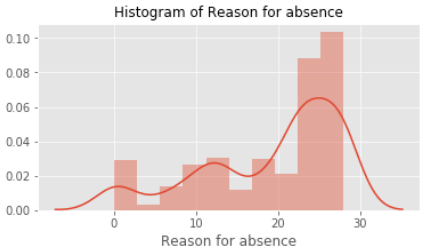
Reason for absence	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	Hit target	Disciplinary failure	Education	Son	Social drinker	Social smoker	Pe
26	26	7	3	1	289	36	13	33	239.554	97	0	1	2	1	0	
0	0	7	3	1	118	13	18	50	239.554	97	1	1	1	1	0	
23	23	7	4	1	179	51	18	38	239.554	97	0	1	0	1	0	
7	7	7	5	1	279	5	14	39	239.554	97	0	1	2	1	1	
23	23	7	5	1	289	36	13	33	239.554	97	0	1	2	1	0	

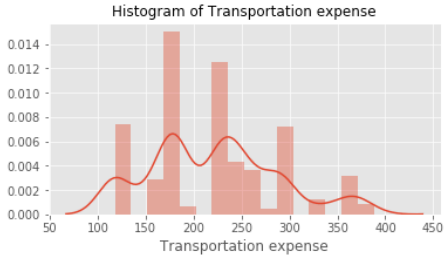
In [8]:
```
absence.shape
```

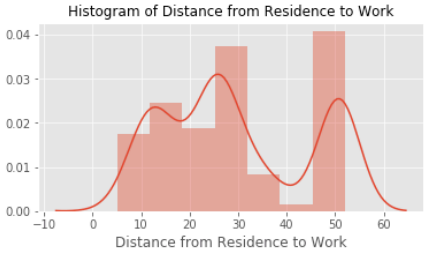
Out[8]:
```
(740, 20)
```

In [9]:
```
# Visualize the data for some of our columns to see what the distribution looks like
import seaborn as sns

for a in ['Reason for absence', 'Transportation expense', 'Distance from Residence to Work']:
    ax=plt.subplots(figsize=(6,3))
    ax=sns.distplot(absence[a])
    title="Histogram of " + a
    ax.set_title(title, fontsize=12)
    plt.show()
```







Now to compare the relative distance between the data points, we need to normalize them
Using MinMaxScaler() to normalize our columns to be between 0 and 1

In [10]:
```
# Standardize the scaling of the columns in order to use any distance based analytical methods
# so that we can compare the relative distances between different feature columns
# Use minmaxscaler to transform the columns so that they also fall between 0 and 1
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
absence_scaled=pd.DataFrame(scaler.fit_transform(absence))
absence_scaled.columns=absence.columns
absence_scaled.index=absence.index
```

In [11]:
```
# See that all of our columns are now between 0 and 1
absence_scaled.describe()
```

Out[11]:

	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	Hit target	Disciplinary failure	Education
count	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000
mean	0.686293	0.527027	0.478716	0.514865	0.382703	0.524066	0.412645	0.304839	0.379108	0.715149	0.054054	0.097297
std	0.301193	0.286357	0.355419	0.370610	0.247971	0.315676	0.156603	0.208993	0.225813	0.198911	0.226277	0.224413
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.464286	0.250000	0.250000	0.333333	0.225926	0.234043	0.285714	0.129032	0.222412	0.631579	0.000000	0.000000
50%	0.821429	0.500000	0.500000	0.666667	0.396296	0.446809	0.428571	0.322581	0.337244	0.736842	0.000000	0.000000
75%	0.928571	0.750000	0.750000	1.000000	0.525926	0.957447	0.535714	0.419355	0.510502	0.842105	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Reduce dimensionality with PCA

In [12]:
```
# Use principal component analysis (PCA) to reduce the dimensionality of our data
bucket='machine-learning-exam'
from sagemaker import PCA
num_components=19

pca_SM = PCA(role=role,
             train_instance_count=1,
             train_instance_type='ml.c4.xlarge',
             output_path='s3://'+ bucket +'/absence/',
             num_components=num_components)
```

In [13]:
```
# Extract the numpy array from the DataFrame and explicitly casting to float32
train_data = absence_scaled.values.astype('float32')
```

Train the model

```
In [14]: %%time
         pca_SM.fit(pca_SM.record_set(train_data))
```

```
2020-02-18 02:03:36 Starting - Starting the training job...
2020-02-18 02:03:37 Starting - Launching requested ML instances......
2020-02-18 02:04:41 Starting - Preparing the instances for training......
2020-02-18 02:06:02 Downloading - Downloading input data
2020-02-18 02:06:02 Training - Downloading the training image..Docker entrypoint called with argument(s): train
[02/18/2020 02:06:18 INFO 140378775521088] Reading default configuration from /opt/amazon/lib/python2.7/site-packages/algorithm/resources/default-conf.json: {u'_num_gpus': u'auto', u'_log_level': u'info', u'subtract_mean': u'true', u'force_dense': u'true', u'epochs': 1, u'algorithm_mode': u'regular', u'extra_components': u'-1', u'_kvstore': u'dist_sync', u'_num_kv_servers': u'auto'}
[02/18/2020 02:06:18 INFO 140378775521088] Reading provided configuration from /opt/ml/input/config/hyperparameters.json: {u'feature_dim': u'20', u'mini_batch_size': u'500', u'num_components': u'19'}
[02/18/2020 02:06:18 INFO 140378775521088] Final configuration: {u'num_components': u'19', u'_num_gpus': u'auto', u'_log_level': u'info', u'subtract_mean': u'true', u'force_dense': u'true', u'epochs': 1, u'algorithm_mode': u'regular', u'feature_dim': u'20', u'extra_components': u'-1', u'_kvstore': u'dist_sync', u'_num_kv_servers': u'auto', u'mini_batch_size': u'500'}
[02/18/2020 02:06:18 WARNING 140378775521088] Loggers have already been setup.

#metrics {"Metrics": {"finalize.time": {"count": 1, "max": 20.998001098632812, "sum": 20.998001098632812, "min": 20.998001098632812}}, "EndTime": 1581991579.423485, "Dimensions": {"Host": "algo-1", "Operation": "training", "Algorithm": "PCA"}, "StartTime": 1581991579.402002}

[02/18/2020 02:06:19 INFO 140378775521088] Test data is not provided.
#metrics {"Metrics": {"totaltime": {"count": 1, "max": 816.9910907745361, "sum": 816.9910907745361, "min": 816.9910907745361}, "setuptime": {"count": 1, "max": 39.095163345336914, "sum": 39.095163345336914, "min": 39.095163345336914}}, "EndTime": 1581991579.430532, "Dimensions": {"Host": "algo-1", "Operation": "training", "Algorithm": "PCA"}, "StartTime": 1581991579.423543}


2020-02-18 02:06:27 Uploading - Uploading generated training model
2020-02-18 02:06:27 Completed - Training job completed
Training seconds: 46
Billable seconds: 46
CPU times: user 415 ms, sys: 22.9 ms, total: 438 ms
Wall time: 3min 12s
```

Store the model artifact on S3

```
In [15]: # Store model artifacts on S3
         job_name = pca_SM.latest_training_job.name
         model_key = "absence/" + job_name + "/output/model.tar.gz"

         boto3.resource('s3').Bucket(bucket).download_file(model_key, 'model.tar.gz')
         os.system('tar -zxvf model.tar.gz')
```

```
Out[15]: 0
```

We can now load it using MXnet.
In preparation for K-means, extract variance vector

Variance is explained by the largest component(s) that we want to keep

```
In [16]: # After the model is decompressed, we can load the ND array using MXNet
         import mxnet as mx
         pca_model_params = mx.ndarray.load('model_algo-1')
```

```
In [17]: # Exact explained-variance-ratio vector
         s=pd.DataFrame(pca_model_params['s'].asnumpy())
         v=pd.DataFrame(pca_model_params['v'].asnumpy())
```

```
In [18]: # Calculate the variance explained by the largest n components that we want to keep.
         # Take the top 5 components
         # The largest 5 components explain ~62% of the total variance in our dataset
         s.iloc[14:,:].apply(lambda x: x*x).sum()/s.apply(lambda x: x*x).sum()
```

```
Out[18]: 0    0.622284
         dtype: float32
```

Top 5 components give us 62% of the variance

In [19]:
```
# After keeping the top 5 components take only the 5 largest components from original s and v matrix
s_5=s.iloc[14:,:]
v_5=v.iloc[:,14:]
v_5.columns=[0,1,2,3,4]
```

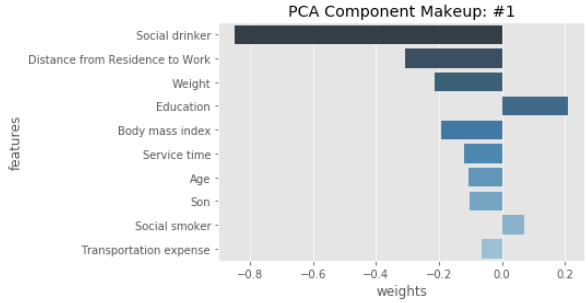
In [20]:
```
v_5.head()
```

Out[20]:

	0	1	2	3	4
0	0.407281	-0.448665	0.193074	0.289915	-0.059501
1	0.217295	-0.079499	-0.023497	-0.452234	-0.036570
2	-0.584830	-0.525670	0.547078	-0.047001	-0.035020
3	0.160280	-0.319543	-0.043324	-0.762302	0.045018
4	0.098019	0.332768	0.286093	-0.152991	-0.065728

In [21]:
```
# Examine the makeup of one of the PCA components based on the weightings of the original features
# that are included in the component
# Understand what the key positive and negative attributes are for the component
component_num=1

first_comp = v_5[5-component_num]
comps = pd.DataFrame(list(zip(first_comp, absence_scaled.columns)), columns=['weights', 'features'])
comps['abs_weights']=comps['weights'].apply(lambda x: np.abs(x))
ax=sns.barplot(data=comps.sort_values('abs_weights', ascending=False).head(10), x="weights", y="features", palette="Blu
ax.set_title("PCA Component Makeup: #" + str(component_num))
plt.show()
```



Deploy PCA

In [22]:
```
%%time
pca_predictor = pca_SM.deploy(initial_instance_count=1,
                              instance_type='ml.t2.medium')
# Deploy the PCA model
```

```
-----------------------!CPU times: user 354 ms, sys: 29.7 ms, total: 384 ms
Wall time: 11min 32s
```

In [23]:
```
%%time
result = pca_predictor.predict(train_data)
# Pass our original dataset to the model so that we can transform the data using the model we created
# Take the largest 5 components and this will reduce the dimensionality of our data from 19 to 5
```

```
CPU times: user 52.7 ms, sys: 0 ns, total: 52.7 ms
Wall time: 259 ms
```

In [24]:
```
# Create a dataset where each absenteeism is described by the 5 principle components
PCA_list=['Social drinker', 'Distance from Residence to Work', 'Weight', 'Education', 'Body mass index']
absence_transformed=pd.DataFrame()
for a in result:
    b=a.label['projection'].float32_tensor.values
    absence_transformed=absence_transformed.append([list(b)])
absence_transformed.index=absence_scaled.index
absence_transformed=absence_transformed.iloc[:,14:]
absence_transformed.columns=PCA_list
```

In [25]:
```
absence_transformed.head()
```

Out[25]:

Reason for absence	Social drinker	Distance from Residence to Work	Weight	Education	Body mass index
26	0.220163	0.277262	0.023891	0.416488	-0.574165
0	-0.813139	0.473745	-0.941796	0.071819	-0.446411
23	0.119235	-0.130268	-0.052720	0.539152	-0.643213
7	-0.839607	0.590692	0.207264	0.134280	-0.147293
23	-0.115692	0.061643	0.277216	0.362107	-0.585175

In [26]:
```
# Use the Kmeans algorithm to segment the absenteeism by the 5 PCA attributes we have created.
train_data = absence_transformed.values.astype('float32')
```

Find the top 7 clusters from our dataset

In [27]:
```
# Find the top 7 clusters from our dataset
from sagemaker import KMeans

num_clusters = 7
kmeans = KMeans(role=role,
                train_instance_count=1,
                train_instance_type='ml.c4.xlarge',
                output_path='s3://'+ bucket +'/absence/',
                k=num_clusters)
```

In [28]:
```
%%time
kmeans.fit(kmeans.record_set(train_data))
```

```
2020-02-18 02:25:53 Starting - Starting the training job...
2020-02-18 02:25:55 Starting - Launching requested ML instances......
2020-02-18 02:26:59 Starting - Preparing the instances for training...
2020-02-18 02:27:43 Downloading - Downloading input data...
2020-02-18 02:27:57 Training - Downloading the training image......
2020-02-18 02:29:22 Uploading - Uploading generated training model
2020-02-18 02:29:22 Completed - Training job completed
Docker entrypoint called with argument(s): train
[02/18/2020 02:29:13 INFO 140491501401920] Reading default configuration from /opt/amazon/lib/python2.7/site-packages/algorithm/resources/default-input.json: {u'_enable_profiler': u'false', u'_tuning_objective_metric': u'', u'_num_gpus': u'auto', u'local_lloyd_num_trials': u'auto', u'_log_level': u'info', u'_kvstore': u'auto', u'local_lloyd_init_method': u'kmeans++', u'force_dense': u'true', u'epochs': u'1', u'init_method': u'random', u'local_lloyd_tol': u'0.0001', u'local_lloyd_max_iter': u'300', u'_disable_wait_to_read': u'false', u'extra_center_factor': u'auto', u'eval_metrics': u'["msd"]', u'_num_kv_servers': u'1', u'mini_batch_size': u'5000', u'half_life_time_size': u'0', u'_num_slices': u'1'}
[02/18/2020 02:29:13 INFO 140491501401920] Reading provided configuration from /opt/ml/input/config/hyperparameters.json: {u'feature_dim': u'5', u'k': u'7', u'force_dense': u'True'}
[02/18/2020 02:29:13 INFO 140491501401920] Final configuration: {u'_tuning_objective_metric': u'', u'extra_center_factor': u'auto', u'local_lloyd_init_method': u'kmeans++', u'force_dense': u'True', u'epochs': u'1', u'feature_dim':
```

```
[02/18/2020 02:29:13 INFO 140491501401920] Test data is not provided.
#metrics {"Metrics": {"totaltime": {"count": 1, "max": 419.1310405731201, "sum": 419.1310405731201, "min": 419.1310405731201}, "setuptime": {"count": 1, "max": 13.462066650390625, "sum": 13.462066650390625, "min": 13.462066650390625}}, "EndTime": 1581992953.978951, "Dimensions": {"Host": "algo-1", "Operation": "training", "Algorithm": "AWS/KMeansWebscale"}, "StartTime": 1581992953.978695}


Training seconds: 99
Billable seconds: 99
CPU times: user 552 ms, sys: 33.4 ms, total: 586 ms
Wall time: 3min 42s
```

Now deploy the K-Means model

In [29]:
```
%%time
kmeans_predictor = kmeans.deploy(initial_instance_count=1,
                                 instance_type='ml.t2.medium')
# Deploy the model
```

```
----------------------!CPU times: user 358 ms, sys: 30 ms, total: 388 ms
Wall time: 11min 32s
```

In [30]:
```
%%time
result=kmeans_predictor.predict(train_data)
```

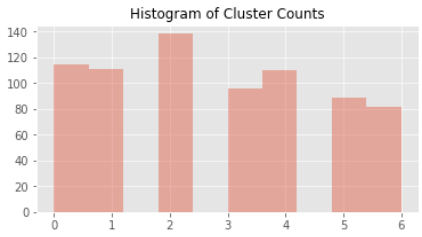
```
CPU times: user 51.9 ms, sys: 4.45 ms, total: 56.3 ms
Wall time: 324 ms
```

In [31]:
```
# Breakdown of cluster counts and the distribution of clusters
cluster_labels = [r.label['closest_cluster'].float32_tensor.values[0] for r in result]
```

In [32]:
```
pd.DataFrame(cluster_labels)[0].value_counts()
```

Out[32]:
```
2.0    138
0.0    114
1.0    111
4.0    110
3.0     96
5.0     89
6.0     82
Name: 0, dtype: int64
```

In [33]:
```
ax=plt.subplots(figsize=(6,3))
ax=sns.distplot(cluster_labels, kde=False)
title="Histogram of Cluster Counts"
ax.set_title(title, fontsize=12)
plt.show()
```



Now get the centroid of each cluster

In [34]:
```
# Access the underlying model to get the cluster centers
# Centers will help describe which features characterize each cluster
# First retrieve the K-Means model attributes
job_name = kmeans.latest_training_job.name
model_key = "absence/" + job_name + "/output/model.tar.gz"

boto3.resource('s3').Bucket(bucket).download_file(model_key, 'model.tar.gz')
os.system('tar -zxvf model.tar.gz')
```

Out[34]: 0

In [35]:
```
Kmeans_model_params = mx.ndarray.load('model_algo-1')
```

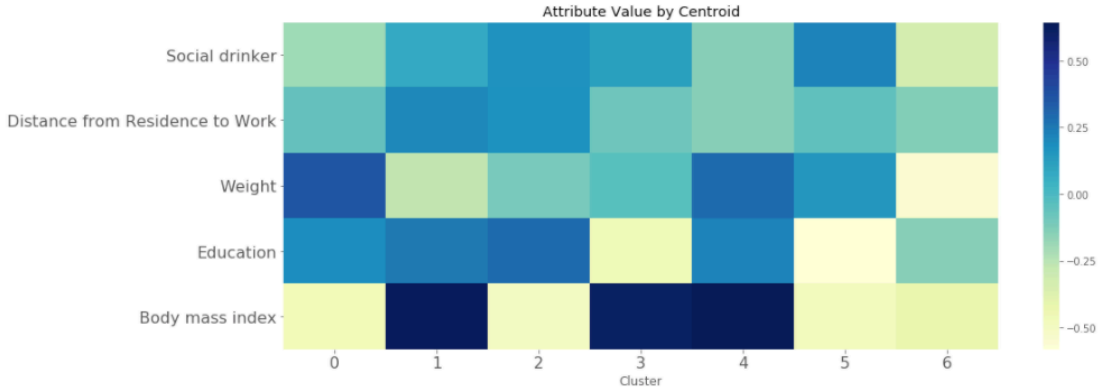
In [36]:
```
# Cluster Centroid Locations: The location of the centers of each cluster identified by the Kmeans algorithm
cluster_centroids=pd.DataFrame(Kmeans_model_params[0].asnumpy())
cluster_centroids.columns=absence_transformed.columns
```

In [37]:
```
cluster_centroids
```

Out[37]:

	Social drinker	Distance from Residence to Work	Weight	Education	Body mass index
0	-0.194499	-0.061751	0.357695	0.185599	-0.471168
1	0.072919	0.208198	-0.263245	0.256122	0.633057
2	0.170031	0.178539	-0.110450	0.290928	-0.498988
3	0.117853	-0.079949	-0.034841	-0.466216	0.614537
4	-0.145866	-0.142386	0.292171	0.223818	0.644043
5	0.222085	-0.052123	0.152401	-0.581722	-0.477426
6	-0.341128	-0.128244	-0.547628	-0.137222	-0.414144

In [38]:
```
# Plot a heatmap of the centroids and their location in the transformed feature space
plt.figure(figsize = (16, 6))
ax = sns.heatmap(cluster_centroids.T, cmap = 'YlGnBu')
ax.set_xlabel("Cluster")
plt.yticks(fontsize = 16)
plt.xticks(fontsize = 16)
ax.set_title("Attribute Value by Centroid")
plt.show()
```



We can see the 7 clusters with how 5 features explained the absence from work.

Body mass seems to be the biggest influencer

In [39]:
```
# Map the cluster labels back to each individual absenteeism and examine which were naturally grouped together
absence_transformed['labels']=list(map(int, cluster_labels))
absence_transformed.head()
```

Out[39]:

Reason for absence	Social drinker	Distance from Residence to Work	Weight	Education	Body mass index	labels
26	0.220163	0.277262	0.023891	0.416488	-0.574165	2
0	-0.813139	0.473745	-0.941796	0.071819	-0.446411	6
23	0.119235	-0.130268	-0.052720	0.539152	-0.643213	2
7	-0.839607	0.590692	0.207264	0.134280	-0.147293	0
23	-0.115692	0.061643	0.277216	0.362107	-0.585175	0

In [40]:
```
# Examine one of the clusters in more detail
cluster=absence_transformed[absence_transformed['labels']==1]
cluster.head(5)
```

Out[40]:

Reason for absence	Social drinker	Distance from Residence to Work	Weight	Education	Body mass index	labels
22	0.168640	0.200160	-0.450106	0.484331	0.634288	1
23	-0.030780	-0.020388	-0.108434	0.353993	0.284006	1
10	-0.050468	0.298167	-0.049213	0.243537	0.644102	1
18	0.045163	0.521368	-0.352631	0.338169	0.385478	1
23	0.520496	0.189885	0.255469	0.403811	0.571548	1

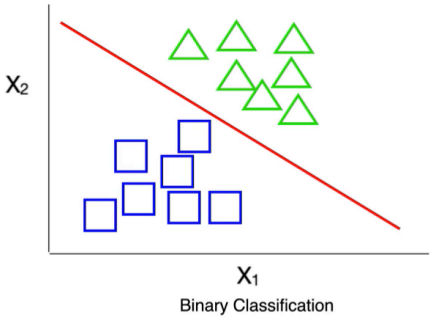
In [41]:
```
# Delete the PCA endpoint
sagemaker.Session().delete_endpoint(pca_predictor.endpoint)
```

In [42]:
```
# Delete the K-Means endpoint
sagemaker.Session().delete_endpoint(kmeans_predictor.endpoint)
```

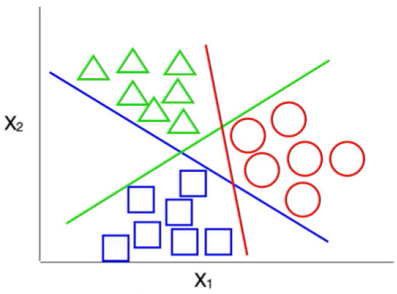
Classification algorithms

Classification Algorithm - Defined

- Supervised learning algorithm
- Learns from training data then uses the result to classify new observations
- Two types: binary-class or multi-class



Binary Classification



Multi-Class Classification

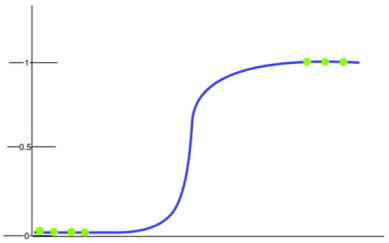
Example Classification Algorithm Use Cases

- Voter prediction
- Customer loan default
- Object detection
- Image classification
- Fraud detection
- Customer segmentation
- Product classification

LinearLearner as a Classifier

SageMaker Classification Algorithms - Linear Learner

- Linear Learner
  - Input a set of high-dimensional vectors including a numeric target, or label
  - Target is 0 or 1 for binary classification
  - Learns a linear threshold function and maps a vector to an approximation of the target
  - Good classification model based on Linear Learner optimizes
    - Discrete objectives suited for classification, such as F1 measure, precision, recall, or accuracy.
  - Requires a data matrix of observations across dimension of features
  - Also requires a target column across the observations
  - Example use case: predict event outcome - win or lose



Binary Classification

SageMaker Classification Algorithms - Blazing Text

- Blazing Text
  - Implements the Word2vec and text classification algorithms
  - Useful for many downstream natural language processing (NLP) tasks, such as sentiment analysis, named entity recognition, machine translation
  - Also useful for applications that perform web searches, information retrieval, ranking, and document classification
  - Maps words to vectors
  - Resulting vector representation of a word is called a word embedding
  - Words that are semantically similar correspond to vectors that are close together, resulting that word embeddings capture the semantic relationships between words.
  - Example use case: spam detection using detection of common spam phrases like "you're a winner!"

SageMaker Classification Algorithms - XGBoost

- XGBoost
  - Implementation of gradient boosted trees algorithm
  - Supervised learning algorithm for predicting a target by combining the estimates from a set of simpler models
  - Requires a data matrix of observations across dimension of features
  - Also requires a target column across the observations
  - Can differentiate the importance of features through weights
  - Example use case: predict default on credit card payments

SageMaker Classification Algorithms - K-Nearest Neighbors

- K-Nearest Neighbors
  - Finds the k closest points to the sample point and gives a prediction of the average of their features
  - Index based
  - Objective: build k-NN index to allow for efficient determination of the distance between points
  - Train to construct the index
  - Use dimensionality reduction to avoid the "curse of dimensionality"
  - Example use case: predict wilderness tree types from geological and forest service data

SageMaker Classification Algorithms - Factorization Machines

- Factorization Machines
  - Extension of linear model used on high dimensional sparse datasets
  - Typically used for sparse datasets such as click prediction and item recommendation
  - Scored using Binary Cross Entropy (Log Loss), Accuracy (at threshold=0.5) and F1 Score (at threshold=0.5)
  - Example use case: item recommendation

SageMaker Classification Algorithms - Image Classification

- Image Classification
  - Supervised learning algorithm that supports multi-label classification
  - Takes an image as input and outputs one or more labels assigned to that image
  - Uses a convolutional neural network (ResNet) that can be trained from scratch or trained using transfer learning when a large number of training images are not available
  - Recommended input format is RecordIO, but can also use raw images in .jpg or .png format.
  - Example use case: classify image content as offensive or not for Twitter feed posts

SageMaker Classification Algorithms - Random Cut Forest

- Random Cut Forest
  - Unsupervised algorithm for detecting anomalous data points within a data set
  - Uses an anomaly score
    - Low score indicates data point is considered normal, high score indicates the presence of an anomaly in the data
    - Definition of "low" and "high" depend on the application, common practice: scores beyond three standard deviations from the mean score are considered anomalous
  - Requires a target column across the observations
  - Example use case: find exceptions in streaming trade data

SageMaker Classification Algorithms - Important Hyperparameters

- Linear Learner
  - feature_dim: number of feature in the input
  - predictor_type: type of the target variable (binary_classifier or multiclass_classifier)
  - loss: specifies the loss function (auto, logistic, hinge_loss, softmax_loss)
- XGBoost
  - num_round: number of rounds the training runs
  - num_class: the number of classes
  - objective: learning task and learning objective (i.e. reg:logistic, multi:softmax)

SageMaker Classification Algorithms - Important Hyperparameters

- K-Nearest Neighbors
  - feature_dim: number of feature in the input
  - k: number of nearest neighbors
  - predictor_type: classifier for classification problems
  - sample_size: number of data points to be samples from the training dataset
  - dimensionality_reductoin_target: target dimension to reduce to
- Factorization Machines
  - feature_dim: number of feature in the input
  - num_factors: dimensionality of factorization
  - predictor_type: binary_classifier for classification problems

SageMaker Classification Algorithms - Important Hyperparameters

- Blazing Text
  - mode: Word2vec architecture used for training (batch_skipgram, skipgram, cbow)
- Image Classification
  - num_classes: the number of output classes
  - num_training_samples: Number of training examples in the input dataset
- Random Cut Forest
  - feature_dim: number of feature in the input
  - eval_metrics: List of metrics used to score a labeled test data set (accuracy, precision_recall_fscore)
  - num_trees: Number of trees in the forest

Classification Algorithm - Lab

Building an XGBoost classification model, with urvine bank dataset

In [1]:
```
# import libraries
import boto3, re, sys, math, json, os, sagemaker, urllib.request
from sagemaker import get_execution_role
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import Image
from IPython.display import display
from time import gmtime, strftime
from sagemaker.predictor import csv_serializer

# Define IAM role
role = get_execution_role()
prefix = 'bank-marketing'
# each region has its XGBoost container
containers = {'us-west-2': '433757028032.dkr.ecr.us-west-2.amazonaws.com/xgboost:latest',
              'us-east-1': '811284229777.dkr.ecr.us-east-1.amazonaws.com/xgboost:latest',
              'us-east-2': '825641698319.dkr.ecr.us-east-2.amazonaws.com/xgboost:latest',
              'eu-west-1': '685385470294.dkr.ecr.eu-west-1.amazonaws.com/xgboost:latest'}
my_region = boto3.session.Session().region_name # set the region of the instance
print("Great! - your SageMaker Instance is in the " + my_region + " region. You will use the " + containers[my_region] +
```

```
Great! - your SageMaker Instance is in the us-east-1 region. You will use the 811284229777.dkr.ecr.us-east-1.amazonaw
s.com/xgboost:latest container for your SageMaker endpoint to make inference requests.
```

In [2]:
```
# Download from your S3 bucket the bank marketing data CSV file based on the publically available census data from the
from io import StringIO
s3 = boto3.resource('s3')
bucket_name = 'machine-learning-exam' # place the bank-additional-full.csv file in a bucket in your account
object_key = 'bank-additional-full.csv'

# Load the data into a pandas dataframe

csv_obj = s3.Object(bucket_name, object_key)
csv_string = csv_obj.get()['Body'].read().decode('utf-8')

raw_data = pd.read_csv(StringIO(csv_string), sep=';')
raw_data.head()
```

Out[2]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	co
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1.1	

5 rows × 21 columns

One Hot encoding of categorical data

In [3]:
```
model_data = pd.get_dummies(raw_data)
model_data.head()
```

Out[3]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	...	day_of_week_fri	day_of_week_mon	day_c
0	56	261	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	0	1	
1	57	149	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	0	1	
2	37	226	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	0	1	
3	40	151	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	0	1	
4	56	307	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	0	1	

5 rows × 65 columns

Out[3]:

_of_week_fri	day_of_week_mon	day_of_week_thu	day_of_week_tue	day_of_week_wed	poutcome_failure	poutcome_nonexistent	poutcome_success	y_no	y_yes
0	1	0	0	0	0	1	0	1	0
0	1	0	0	0	0	1	0	1	0
0	1	0	0	0	0	1	0	1	0
0	1	0	0	0	0	1	0	1	0
0	1	0	0	0	0	1	0	1	0

Target column was moved from “y” - we will us “y_yes” as target

In [4]:
```
# Randomize the data and split it between train and test datasets on a 70% 30% split respectively
train_data, test_data = np.split(model_data.sample(frac=1, random_state=1729), [int(0.7 * len(model_data))])
print(train_data.shape, test_data.shape)
```

```
(28831, 65) (12357, 65)
```

In [5]:
```
# Reformat the header and first column of the training data,
# save the new train dataset to your S3 bucket as train.csv and load the data from the S3 bucket
pd.concat([train_data['y_yes'], train_data.drop(['y_no', 'y_yes'], axis=1)], axis=1).to_csv('train.csv', index=False, h
boto3.Session().resource('s3').Bucket(bucket_name).Object(os.path.join(prefix, 'train/train.csv')).upload_file('train.c
s3_input_train = sagemaker.s3_input(s3_data='s3://{}/{}/train'.format(bucket_name, prefix), content_type='csv')
```

Set our XGBoost estimator and set hyper parameters: binary:logistic

In [6]:
```
# Set up the SageMaker session, create an instance of the XGBoost model (an estimator),
# and define the model's hyperparameters
session_sm = sagemaker.Session()
xgb = sagemaker.estimator.Estimator(containers[my_region],
                                    role,
                                    train_instance_count=1,
                                    train_instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(bucket_name, prefix),
                                    sagemaker_session=session_sm)

xgb.set_hyperparameters(eta=0.1,
                        objective='binary:logistic',
                        num_round=25)
```

In [7]:
```
# After the data is loaded and the XGBoost estimator is set up,
# train the model using gradient optimization on a ml.m4.xlarge instance
xgb.fit({'train': s3_input_train})
```

```
2020-02-19 11:10:58 Starting - Starting the training job...
2020-02-19 11:11:00 Starting - Launching requested ML instances......
2020-02-19 11:12:03 Starting - Preparing the instances for training...
2020-02-19 11:12:53 Downloading - Downloading input data...
2020-02-19 11:13:11 Training - Downloading the training image..Arguments: train
[2020-02-19:11:13:32:INFO] Running standalone xgboost training.
[2020-02-19:11:13:32:INFO] Path /opt/ml/input/data/validation does not exist!
[2020-02-19:11:13:32:INFO] File size need to be processed in the node: 4.36mb. Available memory size in the node: 851
0.23mb
[2020-02-19:11:13:32:INFO] Determined delimiter of CSV input is ','
[11:13:32] S3DistributionType set as FullyReplicated
[11:13:32] 28831x63 matrix with 1816353 entries loaded from /opt/ml/input/data/train?format=csv&label_column=0&delimi
ter=,
[11:13:32] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 120 extra nodes, 0 pruned nodes, max_depth=6
[0]#011train-error:0.078353
[11:13:32] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 122 extra nodes, 0 pruned nodes, max_depth=6
[1]#011train-error:0.077625
[11:13:32] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 120 extra nodes, 0 pruned nodes, max_depth=6
[2]#011train-error:0.077486
[11:13:32] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 122 extra nodes, 0 pruned nodes, max_depth=6
[3]#011train-error:0.077729
[11:13:32] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 120 extra nodes, 0 pruned nodes, max_depth=6
[11:13:33] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 118 extra nodes, 0 pruned nodes, max_depth=6
[23]#011train-error:0.073775
[11:13:33] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 96 extra nodes, 0 pruned nodes, max_depth=6
[24]#011train-error:0.073428

2020-02-19 11:13:44 Uploading - Uploading generated training model
2020-02-19 11:13:44 Completed - Training job completed
Training seconds: 51
Billable seconds: 51
```

Deploy the model to an end point

In [8]:
```
# Deploy your model and create an endpoint that you can access
xgb_predictor = xgb.deploy(initial_instance_count=1,
                           instance_type='ml.m4.xlarge')
```

```
-------------------!
```

Create a test data array from test data, and use xgboost predictor, and pass the test data

In [9]:
```
# Predict whether bank customers in the test dataset will subscribe to a term deposit
test_data_array = test_data.drop(['y_no', 'y_yes'], axis=1).values #load the data into an array
xgb_predictor.content_type = 'text/csv' # set the data type for an inference
xgb_predictor.serializer = csv_serializer # set the serializer type

predictions = xgb_predictor.predict(test_data_array).decode('utf-8') # predict!

predictions_array = np.fromstring(predictions[1:], sep=',') # and turn the prediction into an array
print(predictions_array.shape)
```

```
(12357,)
```

Evaluate our model with a confusion matrix

```
In [10]: # Evaluate the performance and accuracy of the model
cm = pd.crosstab(index=test_data['y_yes'],
                 columns=np.round(predictions_array),
                 rownames=['Observed'],
                 colnames=['Predicted'])

tn = cm.iloc[0,0]; fn = cm.iloc[1,0]; tp = cm.iloc[1,1]; fp = cm.iloc[0,1];
p = (tp+tn)/(tp+tn+fp+fn)*100

print("\n{0:<20}{1:<4.1f}%\n".format("Overall Classification Rate: ", p))
print("{0:<15}{1:<15}{2:>8}".format("Predicted", "Not Subscribed", "Subscribed"))
print("Observed")
print("{0:<15}{1:<2.0f}% ({2:<}){3:>6.0f}% ({4:<})".format("Not Subscribed", tn/(tn+fn)*100,tn, fp/(tp+fp)*100, fp))
print("{0:<16}{1:<1.0f}% ({2:<}){3:>7.0f}% ({4:<}) \n".format("Subscribed", fn/(tn+fn)*100,fn, tp/(tp+fp)*100, tp))
```

```
Overall Classification Rate: 91.7%

Predicted      Not Subscribed Subscribed
Observed
Not Subscribed 94% (10575)    32% (361)
Subscribed      6% (666)     68% (755)
```

Our model could classify correctly 91.7%

Client code to use the end point

client.py

```
import boto3
import json
import csv
import pandas as pd

ENDPOINT_NAME = 'xgboost-2020-02-19-11-10-58-119'
runtime= boto3.client('runtime.sagemaker')

# Convert categorical date field
observation_no = "0,54,99,3,999,0,-1.1,94.601,-49.5,0.987,4963.6,0,0,0,0,0,0,1,0,0,0,0,0,0,1
observation_yes = "0,27,139,1,1,1,-1.8,92.89299999999999,-46.2,1.266,5099.1,0,0,0,0,0,0,0,1,
#observation = observation.replace("-","")

data_no = {"data":observation_no}
actual_no = {"data":"0"}
payload_no = data_no['data']

data_yes = {"data":observation_yes}
actual_yes = {"data":"1"}
payload_yes = data_yes['data']
# print(payload)
```

```
24
25  response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
26                                     ContentType='text/csv',
27                                     Body=payload_no)
28  # print(response)
29
30  result = json.loads(response['Body'].read().decode())
31  #print('prediction: ', result['predictions'][0]['score'], '\t\tactual: ', str(actual['data']
32  print('result: ', result, 'prediction: ', str(actual_no['data']))
33
```

```
Office-Mac-mini:linearLearnerClient vincentbloise$ python client.py
result:  0.224411174655 prediction:  0
```