Introduction to machine learning II: Titanic survival prediction challenge

September 28, Jialin Yi

[Titanic: Machine Learning from Disaster (https://www.kaggle.com/c/titanic/overview)](https://www.kaggle.com/c/titanic/overview) is a popular data science challenge, which asks the data scientists to predict the survivial probability of the passengers on Titanic given their biographic data.

1. Define the problem

Project Summary: The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this challenge, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

Practice Skills

- Binary **classification**
- Python basics

2. Prepare the tools

Step 1: import libraries

In [1]:

```
#load packages
import sys #access to system parameters https://docs.python.org/3/library/sys.html
print("Python version: {}". format(sys.version))

import pandas as pd #collection of functions for data processing and analysis modeled a
fter R dataframes with SQL like features
print("pandas version: {}". format(pd.__version__))

import matplotlib #collection of functions for scientific and publication-ready visuali
zation
print("matplotlib version: {}". format(matplotlib.__version__))

import numpy as np #foundational package for scientific computing
print("NumPy version: {}". format(np.__version__))

import scipy as sp #collection of functions for scientific computing and advance mathem
atics
print("SciPy version: {}". format(sp.__version__))

import IPython
from IPython import display #pretty printing of dataframes in Jupyter notebook
print("IPython version: {}". format(IPython.__version__))

import sklearn #collection of machine learning algorithms
print("scikit-learn version: {}". format(sklearn.__version__))

#misc libraries
import random
import time

#ignore warnings
import warnings
warnings.filterwarnings('ignore')
print('-'*25)
```

```
Python version: 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit
(AMD64)]
pandas version: 0.24.2
matplotlib version: 3.1.0
NumPy version: 1.16.4
SciPy version: 1.2.1
IPython version: 7.6.1
scikit-learn version: 0.21.2
-------------------------
```

Step 2: import machine learning libraries

We will use the popular *scikit-learn* library to develop our machine learning algorithms. In *sklearn,* algorithms are called Estimators and implemented in their own classes. For data visualization, we will use the *matplotlib* and *seaborn* library. Below are common classes to load.

In [2]:

```
#Common Model Algorithms
from sklearn import svm, tree, linear_model, neighbors, naive_bayes, ensemble, discrimi
nant_analysis, gaussian_process

#Common Model Helpers
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics

#Visualization
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import seaborn as sns
from pandas.plotting import scatter_matrix

#Configure Visualization Defaults
#%matplotlib inline = show plots in Jupyter Notebook browser
%matplotlib inline
mpl.style.use('ggplot')
sns.set_style('white')
pylab.rcParams['figure.figsize'] = 12,8
```

Step 3: Preview Data

In [3]:

```
#import data from file: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.r
ead_csv.html
data_raw = pd.read_csv('./data/train.csv')

data_val  = pd.read_csv('./data/test.csv')


#to play with our data we'll create a copy
#remember python assignment or equal passes by reference vs values, so we use the copy
 function: https://stackoverflow.com/questions/46327494/python-pandas-dataframe-copydee
p-false-vs-copydeep-true-vs
data1 = data_raw.copy(deep = True)

#however passing by reference is convenient, because we can clean both datasets at once
data_cleaner = [data1, data_val]


#preview data
print (data_raw.info()) #https://pandas.pydata.org/pandas-docs/stable/generated/pandas.
DataFrame.info.html
#data_raw.head() #https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFra
me.head.html
#data_raw.tail() #https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFra
me.tail.html
data_raw.sample(10) #https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Data
Frame.sample.html
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
None
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
185	186	0	1	Rood, Mr. Hugh Roscoe	male	NaN	0	0	113767
286	287	1	3	de Mulder, Mr. Theodore	male	30.0	0	0	345774
449	450	1	1	Peuchen, Major. Arthur Godfrey	male	52.0	0	0	113786
784	785	0	3	Ali, Mr. William	male	25.0	0	0	SOTON/O.Q. 3101312
275	276	1	1	Andrews, Miss. Kornelia Theodosia	female	63.0	1	0	13502
267	268	1	3	Persson, Mr. Ernst Ulrik	male	25.0	1	0	347083
769	770	0	3	Gronnestad, Mr. Daniel Danielsen	male	32.0	0	0	8471
149	150	0	2	Byles, Rev. Thomas Roussel Davids	male	42.0	0	0	244310
273	274	0	1	Natsch, Mr. Charles H	male	37.0	0	1	PC 17596
238	239	0	2	Pengelly, Mr. Frederick William	male	19.0	0	0	28665

3. Data preprocessing

3.1. The 4 C's of Data Cleaning: Correcting, Completing, Creating, and Converting

In this stage, we will clean our data by 1) correcting aberrant values and outliers, 2) completing missing information, 3) creating new features for analysis, and 4) converting fields to the correct format for calculations and presentation.

1. **Correcting:** Reviewing the data, there does not appear to be any aberrant or non-acceptable data inputs. In addition, we see we may have potential outliers in age and fare. However, since they are reasonable values, we will wait until after we complete our exploratory analysis to determine if we should include or exclude from the dataset. It should be noted, that if they were unreasonable values, for example age = 800 instead of 80, then it's probably a safe decision to fix now. However, we want to use caution when we modify data from its original value, because it may be necessary to create an accurate model.
2. **Completing:** There are two common methods, either delete the record or populate the missing value using a reasonable input. It is not recommended to delete the record, especially a large percentage of records, unless it truly represents an incomplete record. Instead, it's best to impute missing values. A basic methodology for qualitative data is impute using mode. A basic methodology for quantitative data is impute using mean, median, or mean + randomized standard deviation.
3. **Creating:** Feature engineering is when we use existing features to create new features to determine if they provide new signals to predict our outcome. For this dataset, we will create a title feature to determine if it played a role in survival.
4. **Converting:** Last, but certainly not least, we'll deal with formatting. There are no date or currency formats, but datatype formats. Our categorical data imported as objects, which makes it difficult for mathematical calculations. For this dataset, we will convert object datatypes to categorical dummy variables.

In [4]:

```
print('Train columns with null values:\n', data1.isnull().sum())
print("-"*10)

print('Test/Validation columns with null values:\n', data_val.isnull().sum())
print("-"*10)

data_raw.describe(include = 'all')
```

```
Train columns with null values:
 PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
----------
Test/Validation columns with null values:
 PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
----------
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.0
unique	NaN	NaN	NaN	891	2	NaN	NaN	
top	NaN	NaN	NaN	Badt, Mr. Mohamed	male	NaN	NaN	
freq	NaN	NaN	NaN	1	577	NaN	NaN	
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.3
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.8
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.0
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.0
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.0
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.0
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.0

3.2. Clean data

COMPLETING: complete or delete missing values in train and test/validation dataset

In [5]:

```
for dataset in data_cleaner:
    #complete missing age with median
    dataset['Age'].fillna(dataset['Age'].median(), inplace = True)

    #complete embarked with mode
    dataset['Embarked'].fillna(dataset['Embarked'].mode()[0], inplace = True)

    #complete missing fare with median
    dataset['Fare'].fillna(dataset['Fare'].median(), inplace = True)

#delete the cabin feature/column and others previously stated to exclude in train datas
et
drop_column = ['PassengerId','Cabin', 'Ticket']
data1.drop(drop_column, axis=1, inplace = True)

print(data1.isnull().sum())
print("-"*10)
print(data_val.isnull().sum())
```

```
Survived    0
Pclass      0
Name        0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
----------
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0
dtype: int64
```

CREATE: Feature Engineering for train and test/validation dataset

In [6]:

```
for dataset in data_cleaner:
    #Discrete variables
    dataset['FamilySize'] = dataset ['SibSp'] + dataset['Parch'] + 1

    dataset['IsAlone'] = 1 #initialize to yes/1 is alone
    dataset['IsAlone'].loc[dataset['FamilySize'] > 1] = 0 # now update to no/0 if famil
y size is greater than 1

    #quick and dirty code split title from name: http://www.pythonforbeginners.com/dict
ionary/python-split
    dataset['Title'] = dataset['Name'].str.split(", ", expand=True)[1].str.split(".", e
xpand=True)[0]


    #Continuous variable bins; qcut vs cut: https://stackoverflow.com/questions/3021192
3/what-is-the-difference-between-pandas-qcut-and-pandas-cut
    #Fare Bins/Buckets using qcut or frequency bins: https://pandas.pydata.org/pandas-d
ocs/stable/generated/pandas.qcut.html
    dataset['FareBin'] = pd.qcut(dataset['Fare'], 4)

    #Age Bins/Buckets using cut or value bins: https://pandas.pydata.org/pandas-docs/st
able/generated/pandas.cut.html
    dataset['AgeBin'] = pd.cut(dataset['Age'].astype(int), 5)



#cleanup rare title names
#print(data1['Title'].value_counts())
stat_min = 10 #while small is arbitrary, we'll use the common minimum in statistics: ht
tp://nicholasjjackson.com/2012/03/08/sample-size-is-10-a-magic-number/
title_names = (data1['Title'].value_counts() < stat_min) #this will create a true false
series with title name as index

#apply and lambda functions are quick and dirty code to find and replace with fewer lin
es of code: https://community.modeanalytics.com/python/tutorial/pandas-groupby-and-pyth
on-lambda-functions/
data1['Title'] = data1['Title'].apply(lambda x: 'Misc' if title_names.loc[x] == True el
se x)
print(data1['Title'].value_counts())
print("-"*10)
```

```
Mr        517
Miss      182
Mrs       125
Master     40
Misc       27
Name: Title, dtype: int64
----------
```

preview data again

In [7]:

```
data1.info()
data_val.info()
data1.sample(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 14 columns):
Survived      891 non-null int64
Pclass        891 non-null int64
Name          891 non-null object
Sex           891 non-null object
Age           891 non-null float64
SibSp         891 non-null int64
Parch         891 non-null int64
Fare          891 non-null float64
Embarked      891 non-null object
FamilySize    891 non-null int64
IsAlone       891 non-null int64
Title         891 non-null object
FareBin       891 non-null category
AgeBin        891 non-null category
dtypes: category(2), float64(2), int64(6), object(4)
memory usage: 85.5+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 16 columns):
PassengerId    418 non-null int64
Pclass         418 non-null int64
Name           418 non-null object
Sex            418 non-null object
Age            418 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Ticket         418 non-null object
Fare           418 non-null float64
Cabin          91 non-null object
Embarked       418 non-null object
FamilySize     418 non-null int64
IsAlone        418 non-null int64
Title          418 non-null object
FareBin        418 non-null category
AgeBin         418 non-null category
dtypes: category(2), float64(2), int64(6), object(6)
memory usage: 46.8+ KB
```

Out[7]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	Famil
559	1	3	de Messemaeker, Mrs. Guillaume Joseph (Emma)	female	36.0	1	0	17.4000	S	
253	0	3	Lobb, Mr. William Arthur	male	30.0	1	0	16.1000	S	
569	1	3	Jonsson, Mr. Carl	male	32.0	0	0	7.8542	S	
206	0	3	Backstrom, Mr. Karl Alfred	male	32.0	1	0	15.8500	S	
821	1	3	Lulic, Mr. Nikola	male	27.0	0	0	8.6625	S	
778	0	3	Kilgannon, Mr. Thomas J	male	28.0	0	0	7.7375	Q	
870	0	3	Balkic, Mr. Cerin	male	26.0	0	0	7.8958	S	
834	0	3	Allum, Mr. Owen George	male	18.0	0	0	8.3000	S	
200	0	3	Vande Walle, Mr. Nestor Cyriel	male	28.0	0	0	9.5000	S	
474	0	3	Strandberg, Miss. Ida Sofia	female	22.0	0	0	9.8375	S	

CONVERT: convert objects to category using Label Encoder for train and test/validation dataset

In [8]:

```
#code categorical data
label = LabelEncoder()
for dataset in data_cleaner:
    dataset['Sex_Code'] = label.fit_transform(dataset['Sex'])
    dataset['Embarked_Code'] = label.fit_transform(dataset['Embarked'])
    dataset['Title_Code'] = label.fit_transform(dataset['Title'])
    dataset['AgeBin_Code'] = label.fit_transform(dataset['AgeBin'])
    dataset['FareBin_Code'] = label.fit_transform(dataset['FareBin'])


#define y variable aka target/outcome
Target = ['Survived']

#define x variables for original features aka feature selection
data1_x = ['Sex','Pclass', 'Embarked', 'Title','SibSp', 'Parch', 'Age', 'Fare', 'Family
Size', 'IsAlone'] #pretty name/values for charts
data1_x_calc = ['Sex_Code','Pclass', 'Embarked_Code', 'Title_Code','SibSp', 'Parch', 'A
ge', 'Fare'] #coded for algorithm calculation
data1_xy =  Target + data1_x
print('Original X Y: ', data1_xy, '\n')


#define x variables for original w/bin features to remove continuous variables
data1_x_bin = ['Sex_Code','Pclass', 'Embarked_Code', 'Title_Code', 'FamilySize', 'AgeBi
n_Code', 'FareBin_Code']
data1_xy_bin = Target + data1_x_bin
print('Bin X Y: ', data1_xy_bin, '\n')


#define x and y variables for dummy features original
data1_dummy = pd.get_dummies(data1[data1_x])
data1_x_dummy = data1_dummy.columns.tolist()
data1_xy_dummy = Target + data1_x_dummy
print('Dummy X Y: ', data1_xy_dummy, '\n')



data1_dummy.head()
```

```
Original X Y:  ['Survived', 'Sex', 'Pclass', 'Embarked', 'Title', 'SibSp',
'Parch', 'Age', 'Fare', 'FamilySize', 'IsAlone']

Bin X Y:  ['Survived', 'Sex_Code', 'Pclass', 'Embarked_Code', 'Title_Cod
e', 'FamilySize', 'AgeBin_Code', 'FareBin_Code']

Dummy X Y:  ['Survived', 'Pclass', 'SibSp', 'Parch', 'Age', 'Fare', 'Famil
ySize', 'IsAlone', 'Sex_female', 'Sex_male', 'Embarked_C', 'Embarked_Q',
'Embarked_S', 'Title_Master', 'Title_Misc', 'Title_Miss', 'Title_Mr', 'Tit
le_Mrs']
```

Out[8]:

	Pclass	SibSp	Parch	Age	Fare	FamilySize	IsAlone	Sex_female	Sex_male	Embark
0	3	1	0	22.0	7.2500	2	0	0	1	
1	1	1	0	38.0	71.2833	2	0	1	0	
2	3	0	0	26.0	7.9250	1	1	1	0	
3	1	1	0	35.0	53.1000	2	0	1	0	
4	3	0	0	35.0	8.0500	1	1	0	1	

Last check

In [9]:

```
print('Train columns with null values: \n', data1.isnull().sum())
print("-"*10)
print (data1.info())
print("-"*10)

print('Test/Validation columns with null values: \n', data_val.isnull().sum())
print("-"*10)
print (data_val.info())
print("-"*10)

data_raw.describe(include = 'all')
```

```
Train columns with null values:
 Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Fare             0
Embarked         0
FamilySize       0
IsAlone          0
Title            0
FareBin          0
AgeBin           0
Sex_Code         0
Embarked_Code    0
Title_Code       0
AgeBin_Code      0
FareBin_Code     0
dtype: int64
----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 19 columns):
Survived         891 non-null int64
Pclass           891 non-null int64
Name             891 non-null object
Sex              891 non-null object
Age              891 non-null float64
SibSp            891 non-null int64
Parch            891 non-null int64
Fare             891 non-null float64
Embarked         891 non-null object
FamilySize       891 non-null int64
IsAlone          891 non-null int64
Title            891 non-null object
FareBin          891 non-null category
AgeBin           891 non-null category
Sex_Code         891 non-null int32
Embarked_Code    891 non-null int32
Title_Code       891 non-null int32
AgeBin_Code      891 non-null int32
FareBin_Code     891 non-null int32
dtypes: category(2), float64(2), int32(5), int64(6), object(4)
memory usage: 102.9+ KB
None
----------
Test/Validation columns with null values:
 PassengerId       0
Pclass             0
Name               0
Sex                0
Age                0
SibSp              0
Parch              0
Ticket             0
Fare               0
Cabin            327
Embarked           0
FamilySize         0
```

```
IsAlone           0
Title             0
FareBin           0
AgeBin            0
Sex_Code          0
Embarked_Code     0
Title_Code        0
AgeBin_Code       0
FareBin_Code      0
dtype: int64
----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 21 columns):
PassengerId      418 non-null int64
Pclass           418 non-null int64
Name             418 non-null object
Sex              418 non-null object
Age              418 non-null float64
SibSp            418 non-null int64
Parch            418 non-null int64
Ticket           418 non-null object
Fare             418 non-null float64
Cabin            91 non-null object
Embarked         418 non-null object
FamilySize       418 non-null int64
IsAlone          418 non-null int64
Title            418 non-null object
FareBin          418 non-null category
AgeBin           418 non-null category
Sex_Code         418 non-null int32
Embarked_Code    418 non-null int32
Title_Code       418 non-null int32
AgeBin_Code      418 non-null int32
FareBin_Code     418 non-null int32
dtypes: category(2), float64(2), int32(5), int64(6), object(6)
memory usage: 54.9+ KB
None
----------
```

Out[9]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.0
unique	NaN	NaN	NaN	891	2	NaN	NaN	
top	NaN	NaN	NaN	Badt, Mr. Mohamed	male	NaN	NaN	
freq	NaN	NaN	NaN	1	577	NaN	NaN	
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.3
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.8
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.0
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.0
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.0
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.0
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.0

3.3. Split training and validation data

we will use *sklearn* function to split the training data in two datasets; 75/25 split. This is important, so we don't [overfit our model (https://www.coursera.org/learn/python-machine-learning/lecture/fVStr/overfitting-and-underfitting)](https://www.coursera.org/learn/python-machine-learning/lecture/fVStr/overfitting-and-underfitting).

In [10]:

```
#split train and test data with function defaults
#random_state -> seed or control random number generator: https://www.quora.com/What-is
-seed-in-random-number-generation
train1_x, test1_x, train1_y, test1_y = model_selection.train_test_split(data1[data1_x_c
alc], data1[Target], random_state = 0)
train1_x_bin, test1_x_bin, train1_y_bin, test1_y_bin = model_selection.train_test_split
(data1[data1_x_bin], data1[Target] , random_state = 0)
train1_x_dummy, test1_x_dummy, train1_y_dummy, test1_y_dummy = model_selection.train_te
st_split(data1_dummy[data1_x_dummy], data1[Target], random_state = 0)


print("Data1 Shape: {}".format(data1.shape))
print("Train1 Shape: {}".format(train1_x.shape))
print("Test1 Shape: {}".format(test1_x.shape))

train1_x_bin.head()
```

```
Data1 Shape: (891, 19)
Train1 Shape: (668, 8)
Test1 Shape: (223, 8)
```

Out[10]:

	Sex_Code	Pclass	Embarked_Code	Title_Code	FamilySize	AgeBin_Code	FareBin_Cod
105	1	3	2	3	1	1	
68	0	3	2	2	7	1	
253	1	3	2	3	2	1	
320	1	3	2	3	1	1	
706	0	2	2	4	1	2	

4. Applying machine learning algorithms

We will use a classification algorithm from the *sklearn* library to begin our analysis. We will use cross validation and scoring metrics, discussed in later sections, to rank and compare our algorithms' performance.

**Machine Learning Selection:*

- Sklearn Estimator Overview (http://scikit-learn.org/stable/user_guide.html)
- Sklearn Estimator Detail (http://scikit-learn.org/stable/modules/classes.html)
- Choosing Estimator Mind Map (http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)
- Choosing Estimator Cheat Sheet (https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Scikit_Learn_Cheat_Sheet_Python.pdf)

Now that we identified our solution as a supervised learning classification algorithm. We can narrow our list of choices.

Machine Learning Classification Algorithms:

- Ensemble Methods (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble)
- Generalized Linear Models (GLM) (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model)
- Naive Bayes (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive_bayes)
- Nearest Neighbors (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.neighbors)
- Support Vector Machines (SVM) (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.svm)
- Decision Trees (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.tree)
- Discriminant Analysis (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.discriminant_analysis)

In [11]:

```
#Machine Learning Algorithm (MLA) Selection and Initialization
MLA = [
    #Ensemble Methods
    ensemble.AdaBoostClassifier(),
    ensemble.BaggingClassifier(),
    ensemble.ExtraTreesClassifier(),
    ensemble.GradientBoostingClassifier(),
    ensemble.RandomForestClassifier(),

    #Gaussian Processes
    gaussian_process.GaussianProcessClassifier(),

    #GLM
    linear_model.LogisticRegressionCV(),
    linear_model.PassiveAggressiveClassifier(),
    linear_model.RidgeClassifierCV(),
    linear_model.SGDClassifier(),
    linear_model.Perceptron(),

    #Navies Bayes
    naive_bayes.BernoulliNB(),
    naive_bayes.GaussianNB(),

    #Nearest Neighbor
    neighbors.KNeighborsClassifier(),

    #SVM
    svm.SVC(probability=True),
    svm.NuSVC(probability=True),
    svm.LinearSVC(),

    #Trees
    tree.DecisionTreeClassifier(),
    tree.ExtraTreeClassifier(),

    #Discriminant Analysis
    discriminant_analysis.LinearDiscriminantAnalysis(),
    discriminant_analysis.QuadraticDiscriminantAnalysis(),
    ]
```

In [12]:

```
#split dataset in cross-validation with this splitter class: http://scikit-learn.org/st
able/modules/generated/sklearn.model_selection.ShuffleSplit.html#sklearn.model_selectio
n.ShuffleSplit
#note: this is an alternative to train_test_split
cv_split = model_selection.ShuffleSplit(n_splits = 10, test_size = .3, train_size = .6,
random_state = 0 ) # run model 10x with 60/30 split intentionally leaving out 10%

#create table to compare MLA metrics
MLA_columns = ['MLA Name', 'MLA Parameters','MLA Train Accuracy Mean', 'MLA Test Accura
cy Mean', 'MLA Test Accuracy 3*STD' ,'MLA Time']
MLA_compare = pd.DataFrame(columns = MLA_columns)

#create table to compare MLA predictions
MLA_predict = data1[Target]

#index through MLA and save performance to table
row_index = 0
for alg in MLA:

    #set name and parameters
    MLA_name = alg.__class__.__name__
    MLA_compare.loc[row_index, 'MLA Name'] = MLA_name
    MLA_compare.loc[row_index, 'MLA Parameters'] = str(alg.get_params())

    #score model with cross validation: http://scikit-learn.org/stable/modules/generate
d/sklearn.model_selection.cross_validate.html#sklearn.model_selection.cross_validate
    cv_results = model_selection.cross_validate(alg, data1[data1_x_bin], data1[Target],
cv  = cv_split, return_train_score=True)

    MLA_compare.loc[row_index, 'MLA Time'] = cv_results['fit_time'].mean()
    MLA_compare.loc[row_index, 'MLA Train Accuracy Mean'] = cv_results['train_score'].m
ean()
    MLA_compare.loc[row_index, 'MLA Test Accuracy Mean'] = cv_results['test_score'].mea
n()
    #if this is a non-bias random sample, then +/-3 standard deviations (std) from the
 mean, should statistically capture 99.7% of the subsets
    MLA_compare.loc[row_index, 'MLA Test Accuracy 3*STD'] = cv_results['test_score'].st
d()*3   #let's know the worst that can happen!


    #save MLA predictions - see section 6 for usage
    alg.fit(data1[data1_x_bin], data1[Target])
    MLA_predict[MLA_name] = alg.predict(data1[data1_x_bin])

    row_index+=1


#print and sort table: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Da
taFrame.sort_values.html
MLA_compare.sort_values(by = ['MLA Test Accuracy Mean'], ascending = False, inplace = T
rue)
MLA_compare
#MLA_predict
```

Out[12]:

	MLA Name	MLA Parameters	MLA Train Accuracy Mean	MLA Test Accuracy Mean	MLA Test Accuracy 3*STD	MLA Time
14	SVC	{'C': 1.0, 'cache_size': 200, 'class_weight': ...	0.837266	0.826119	0.0453876	0.0526167
2	ExtraTreesClassifier	{'bootstrap': False, 'class_weight': None, 'cr...	0.895131	0.823507	0.0666115	0.0168145
3	GradientBoostingClassifier	{'criterion': 'friedman_mse', 'init': None, 'l...	0.866667	0.822761	0.0498731	0.0808086
15	NuSVC	{'cache_size': 200, 'class_weight': None, 'coe...	0.835768	0.822761	0.0493681	0.0482544
4	RandomForestClassifier	{'bootstrap': True, 'class_weight': None, 'cri...	0.890075	0.822388	0.0697631	0.0169207
17	DecisionTreeClassifier	{'class_weight': None, 'criterion': 'gini', 'm...	0.895131	0.822015	0.0539292	0.00533757
1	BaggingClassifier	{'base_estimator': None, 'bootstrap': True, 'b...	0.891011	0.819776	0.0699157	0.0269539
13	KNeighborsClassifier	{'algorithm': 'auto', 'leaf_size': 30, 'metric...	0.850375	0.813806	0.0690863	0.00221417
0	AdaBoostClassifier	{'algorithm': 'SAMME.R', 'base_estimator': Non...	0.820412	0.81194	0.0498606	0.0829229
5	GaussianProcessClassifier	{'copy_X_train': True, 'kernel': None, 'max_it...	0.871723	0.810448	0.0492537	0.241416
18	ExtraTreeClassifier	{'class_weight': None, 'criterion': 'gini', 'm...	0.895131	0.809701	0.0673505	0.00689874
20	QuadraticDiscriminantAnalysis	{'priors': None, 'reg_param': 0.0, 'store_cova...	0.821536	0.80709	0.0810389	0.00624931
8	RidgeClassifierCV	{'alphas': array([0.1, 1. , 10.]), 'class_w...	0.796629	0.79403	0.0360302	0.0115865
19	LinearDiscriminantAnalysis	{'n_components': None, 'priors': None, 'shrink...	0.796816	0.79403	0.0360302	0.00781088
16	LinearSVC	{'C': 1.0, 'class_weight': None, 'dual': True,...	0.797753	0.793284	0.0397979	0.0383488

	MLA Name	MLA Parameters	MLA Train Accuracy Mean	MLA Test Accuracy Mean	MLA Test Accuracy 3*STD	MLA Time
6	LogisticRegressionCV	{'Cs': 10, 'class_weight': None, 'cv': 'warn',...	0.797004	0.790672	0.0653582	0.216193
12	GaussianNB	{'priors': None, 'var_smoothing': 1e-09}	0.794757	0.781343	0.0874568	0.00312479
11	BernoulliNB	{'alpha': 1.0, 'binarize': 0.0, 'class_prior':...	0.785768	0.775373	0.0570347	0.00478771
9	SGDClassifier	{'alpha': 0.0001, 'average': False, 'class_wei...	0.751873	0.745896	0.2813	0.0115861
10	Perceptron	{'alpha': 0.0001, 'class_weight': None, 'early...	0.754494	0.744403	0.123667	0.00362582
7	PassiveAggressiveClassifier	{'C': 1.0, 'average': False, 'class_weight': N...	0.736517	0.732463	0.209304	0.00669138

In [13]:

```
#barplot using https://seaborn.pydata.org/generated/seaborn.barplot.html
sns.barplot(x='MLA Test Accuracy Mean', y = 'MLA Name', data = MLA_compare, color = 'm'
)

#prettify using pyplot: https://matplotlib.org/api/pyplot_api.html
plt.title('Machine Learning Algorithm Accuracy Score \n')
plt.xlabel('Accuracy Score (%)')
plt.ylabel('Algorithm')
```

Out[13]:

```
Text(0, 0.5, 'Algorithm')
```