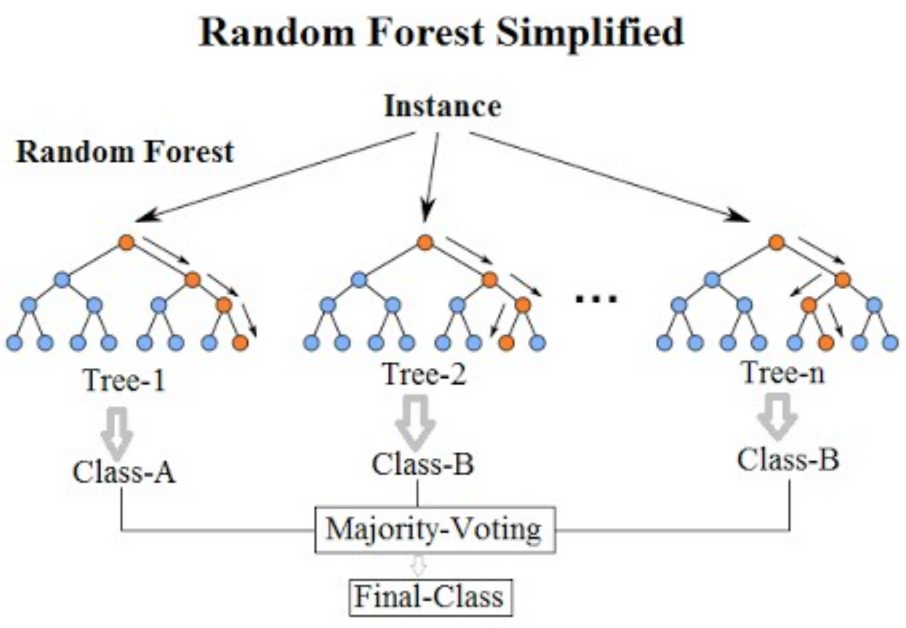
Classification

Random Forest Classification

posted Jan 24, 2018, 12:25 PM by Atul Rana **[updated an hour ago]**

Introduction

We can see it from its name, which is to create a forest in some way and make it random. There is a direct relationship between the number of trees in the forest and the results it can get: the larger the number of trees, the more accurate the result



```
# importing the libraries
```

```
import numpy as np
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

```
dataset.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
X
```

```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
       [    19,  76000],
       [    27,  58000],
       [    27,  84000],...])
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, ....])
```

Splitting the dataset into the Training set and Test set

```
from sklearn.cross_validation import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.25, random_state =
0)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing

Fitting Random Forest Classification to the Training set

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10,criterion='entropy',random_state=
0)
classifier.fit(X_train, y_train)
```

Default parameters used by the model function of Random Forest.
Criterion: The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

Evaluation of Model

```
y_pred = classifier.predict(X_test)
classifier.score(X_test,y_test
```

```
  0.92000000000000004
```

```
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(y_test, y_pred))
```

```
[[63 5]
 [ 3 29]]
```

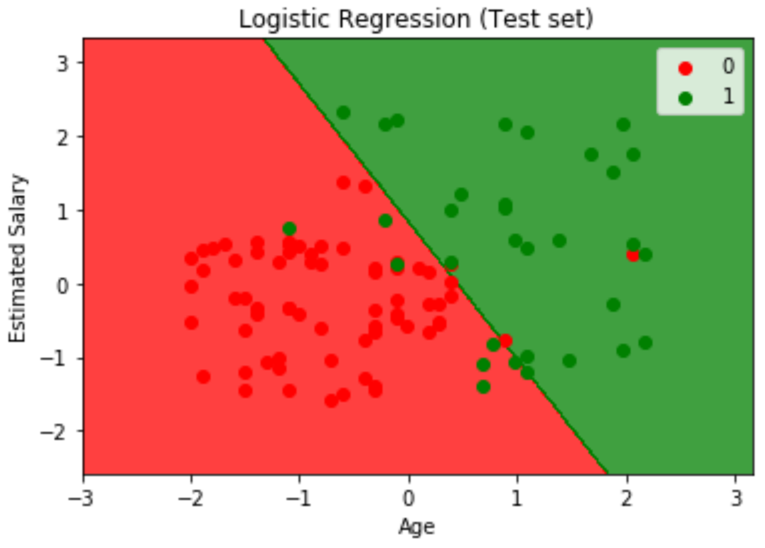
```
print(classification_report(y_test,y_pred))
```

```
             precision    recall  f1-score   support

          0       0.95      0.93      0.94        68
          1       0.85      0.91      0.88        32

avg / total       0.92      0.92      0.92       100
```

Test set prediction looks like below where red/green are the two classes identified and red in green or green in red are the misclassified predictions (error).

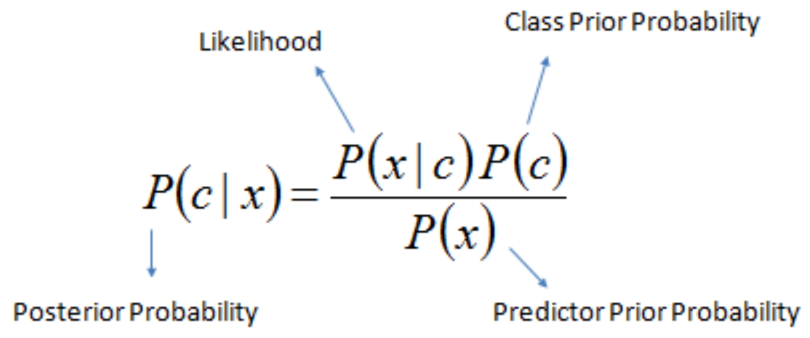


Naive Bayes

posted Jan 24, 2018, 12:13 PM by Atul Rana **[updated an hour ago]**

Introduction

The whole idea is the conditional probability with strong (naive) independence assumptions between the features.


Likelihood
Class Prior Probability
$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$
Posterior Probability
Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering.

```
import numpy as np
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

```
dataset.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
X
```

```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
       [    19,  76000],
       [    27,  58000],
       [    27,  84000],...])
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, ....])
```

Splitting the dataset into the Training set and Test set

Always recommended to split the data and test the accuracy of the data.You can play around with test_size, and random_state, they define the size and random selection of data point from the data respectively.

```
from sklearn.cross_validation import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.25, random_state =
0)
```

Feature Scaling

Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Fitting Naive Bayes to the Training set

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Prediction and evaluation of the Test set results

```
y_pred = classifier.predict(X_test)
classifier.score(X_test,y_test)
```

```
    0.9000000000000002
```

```
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(y_test, y_pred))
```

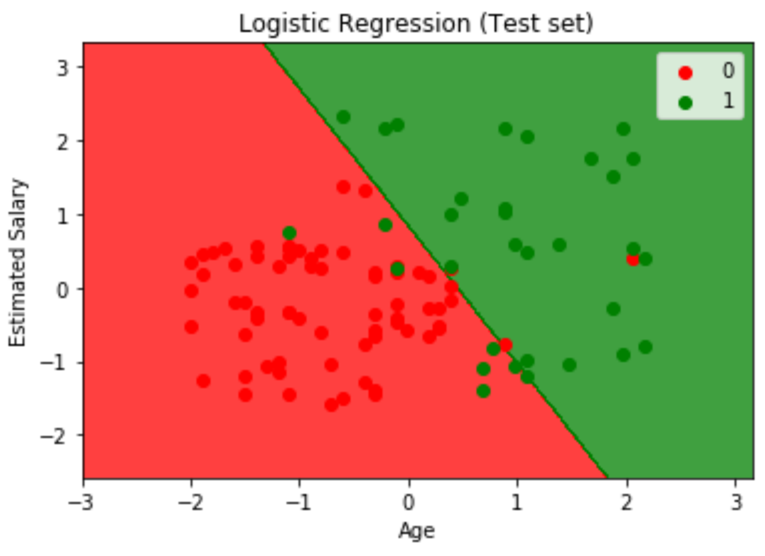
```
[[65 3]
 [ 7 25]]
```

```
print(classification_report(y_test,y_pred))
```

```
             precision    recall  f1-score   support

          0       0.90      0.96      0.93        68
          1       0.89      0.78      0.83        32

avg / total       0.90      0.90      0.90       100
```



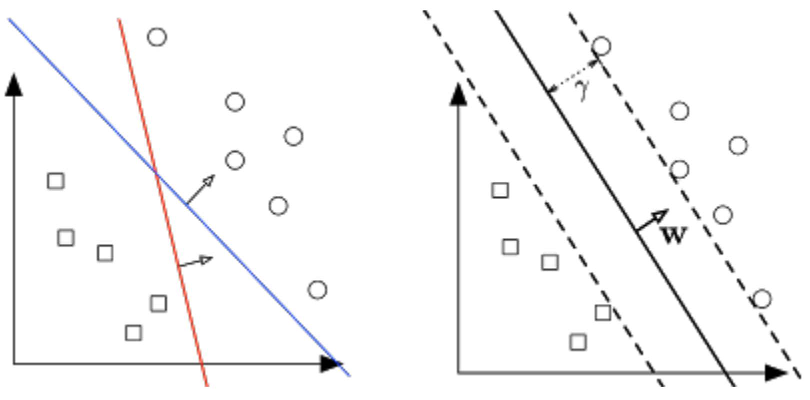
Test set prediction looks like above where red/green are the two classes identified and red in green or green in red are the misclassified predictions (error).

Support Vector Machine (SVM)

posted Jan 24, 2018, 11:57 AM by Atul Rana **[updated an hour ago]**

Introduction

Support Vector Machine also called Large Margin Intuition as it separates the different classes with the margin which is as far as from classes.



this is how svm converts the input space to the feature space and identifies the support vector and make a margin to separate the classes.

Importing the libraries

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

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

```
dataset.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
X
```

```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
       [    19,  76000],
       [    27,  58000],
       [    27,  84000],...])
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, ....])
```

Splitting the dataset into the Training set and Test set

Always recommended to split the data and test the accuracy of the data.You can play around with test_size, and random_state, they define the size and random selection of data point from the data respectively.

```
from sklearn.cross_validation import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.25, random_state =
0)
```

Feature Scaling

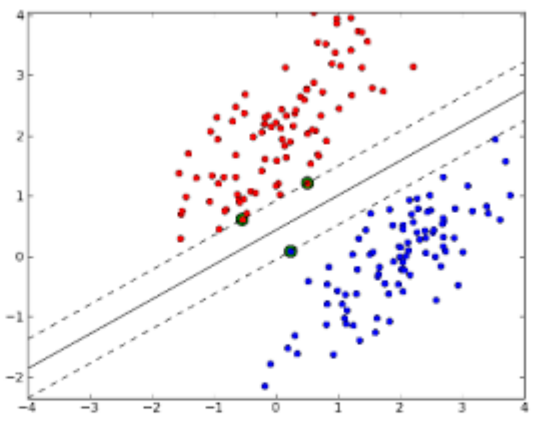
in Data, all column may not have the similar size of scale, like the number of rooms to the size of the room it is always good practice to have a feature scaling on the data before feeding to the training model.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Fitting SVM to the Training set

Defining the SVM object and fitting it with our training data.

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```



other kernel are ***'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'*** that we can use in SVM model.
Init signature:
SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True,probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
classifier.score(X_test,y_test)
```

```
0.9000000000000002
```

```
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(y_test, y_pred))
```

```
[[66 2]
 [ 8 24]]
```

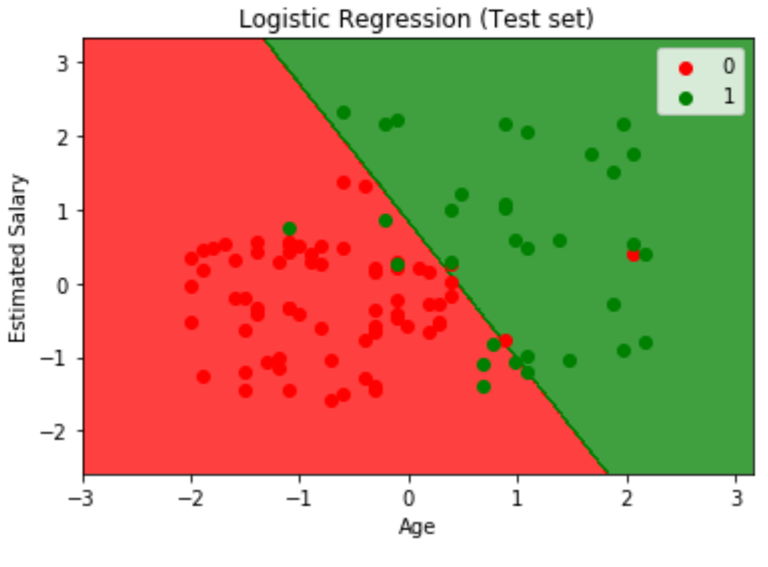
```
print(classification_report(y_test,y_pred))
```

```
             precision    recall  f1-score   support

          0       0.89      0.97      0.93        68
          1       0.92      0.75      0.83        32

avg / total       0.90      0.90      0.90       100
```

Test set prediction looks like below where red/green are the two classes identified and red in green or green in red are the misclassified predictions(error).

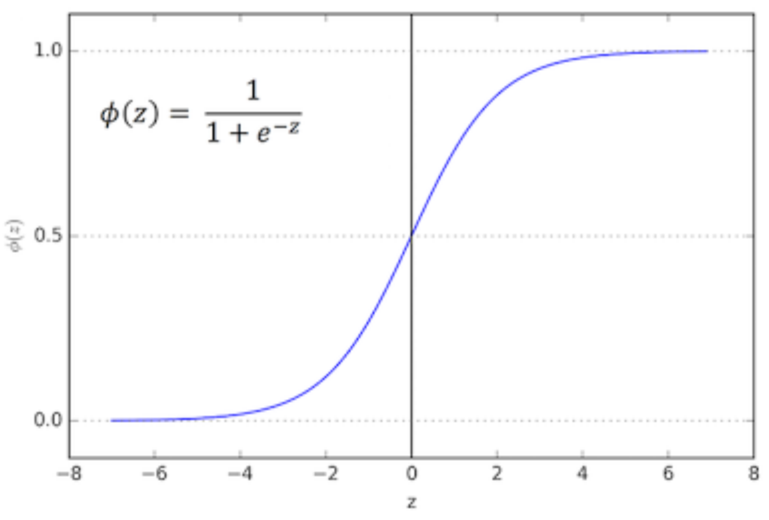


Logistic Regression

posted Jan 24, 2018, 11:31 AM by Atul Rana **[updated an hour ago]**

Indroduction

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome, features are fit into the deciding function something like below and have only two possible outcomes with some probability.



Import a few libraries you think you'll need (Or just import them as you go along!)

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

Getting the data, if data in CSV file we can import as below, there are many other functions included in PANDAS library to work with data importing and preprocessing.

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

Let's see how our data looks in the data frame after importing, **iloc(),** return the data in NUMPY array.

```
dataset.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
X
```

```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
       [    19,  76000],
       [    27,  58000],
       [    27,  84000],...])
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, ....])
```

Splitting the dataset into the Training set and Test set

Always recommended to split the data and test the accuracy of the data.You can play around with test_size, and random_state, they define the size and random selection of data point from the data respectively.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.25, random_state =
0)
```

Feature Scaling

in Data, all column may not have the similar size of scale, like the number of rooms to the size of the room it is always good practice to have a feature scaling on the data before feeding to the training model.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Fitting Logistic Regression to the Training set

We can import the library as we need, it's not like c++ :) while defining object classifier for logistic regression, we can play with the value of random_state.

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
classifier.score(X_test,y_test)
```

```
0.89000000000000001
```

```
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(y_test, y_pred))
```

```
[[65 3]
 [ 8 24]]
```

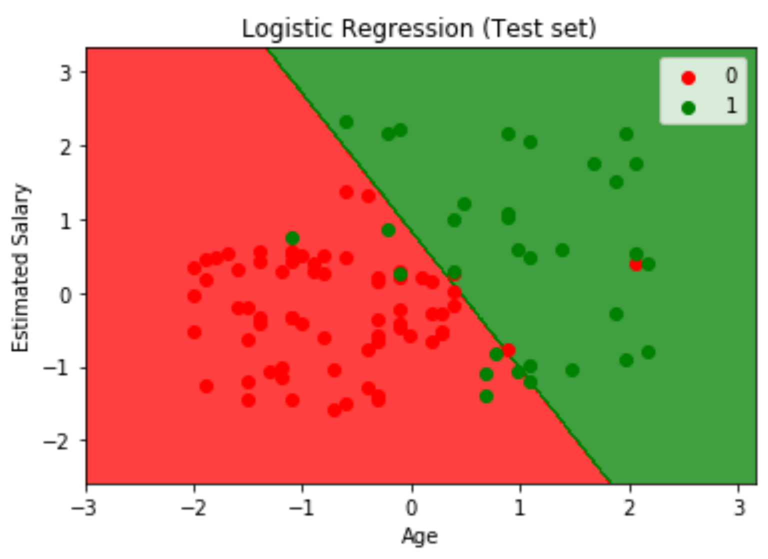
```
print(classification_report(y_test,y_pred))
```

```
             precision    recall  f1-score   support

          0       0.89      0.96      0.92        68
          1       0.89      0.75      0.81        32

avg / total       0.89      0.89      0.89       100
```

- **Visualization of Logistic Regression Working**



Test set prediction looks like above where red/green are the two classes identified and red in green or green in red are the misclassified predictions (error).