

Global Terrorism Problem — Data Science Project

CS 210 Project Step 3 – Machine Learning Techniques

 Global Terrorism Problem  18 May 2019  Leave a comment

Machine Learning Techniques

Feature selection before ML Techniques

Feature Selection is one of the core concepts in machine learning which hugely impacts the performance of your model. Redundant features decrease training speed, decrease model interpretability, decrease generalization performance on the test set.

Useful feature should;

- Not possess a high percentage of missing value (We got rid of missing values by selecting relevant columns and deleting Nan values, also by selecting specific attack type “assassination” we got rid of other missing values and focused on certain attack type.)
- Fair spreadness
- Not be highly correlated with another feature

```

1 import numpy as np
2 import pandas as pd
3 import csv
4 from sklearn.metrics import accuracy_score,
5 from sklearn.model_selection import train_te
6
7 df=pd.read_csv('master.csv',encoding='ISO-88
8 df.rename(columns={'eventid':'Event_ID','iye
9 df=df[['Event_ID','Year','Country','Region',
10
11 df.head(10)

```

	Event_ID	Year	Country	Region	AttackType	Killed	Wounded	Group	Target_type	Weapon_type	success
0	197000000001	1970	Dominican Republic	Central America & Caribbean	Assassination	1.0	0.0	MANO-D	Private Citizens & Property	Unknown	1
1	197000000002	1970	Mexico	North America	Hostage Taking (Kidnapping)	0.0	0.0	23rd of September Communist League	Government (Diplomatic)	Unknown	1
2	197001000001	1970	Philippines	Southeast Asia	Assassination	1.0	0.0	Unknown	Journalists & Media	Unknown	1
3	197001000002	1970	Greece	Western Europe	Bombing/Explosion	NaN	NaN	Unknown	Government (Diplomatic)	Explosives	1
4	197001000003	1970	Japan	East Asia	Facility/Infrastructure Attack	NaN	NaN	Unknown	Government (Diplomatic)	Incendiary	1
5	197001010002	1970	United States	North America	Armed Assault	0.0	0.0	Black Nationalists	Police	Firearms	1
6	197001020001	1970	Uruguay	South America	Assassination	0.0	0.0	Tupamaros (Uruguay)	Police	Firearms	0
7	197001020002	1970	United States	North America	Bombing/Explosion	0.0	0.0	Unknown	Utilities	Explosives	1
8	197001020003	1970	United States	North America	Facility/Infrastructure Attack	0.0	0.0	New Year's Gang	Military	Incendiary	1
9	197001030001	1970	United States	North America	Facility/Infrastructure Attack	0.0	0.0	New Year's Gang	Government (General)	Incendiary	1

```

1 df.isnull().sum()

```

```

Event_ID      0
Year          0
Country       0
Region        0
AttackType    0
Killed       10313
Wounded      16311
Group         0
Target_type   0
Weapon_type   0
success       0
dtype: int64

```

```

1 df=df.dropna()
2 df.index = pd.RangeIndex(len(df.index))
3 df.shape

```

```
Out[14]: (164817, 11)
```

```

1 df=df[df.AttackType == 'Assassination']
2 df.head()

```

	Event_ID	Year	Country	Region	AttackType	Killed	Wounded	Group	Target_type	Weapon_type	success
0	1970000000001	1970	Dominican Republic	Central America & Caribbean	Assassination	1.0	0.0	MANO-D	Private Citizens & Property	Unknown	1
2	1970010000001	1970	Philippines	Southeast Asia	Assassination	1.0	0.0	Unknown	Journalists & Media	Unknown	1
4	1970010200001	1970	Uruguay	South America	Assassination	0.0	0.0	Tupamaros (Uruguay)	Police	Firearms	0
22	1970012000001	1970	Guatemala	Central America & Caribbean	Assassination	1.0	0.0	Unknown	Government (Diplomatic)	Unknown	1
105	1970030500003	1970	United States	North America	Assassination	2.0	0.0	Armed Commandos of Liberation	Military	Firearms	1

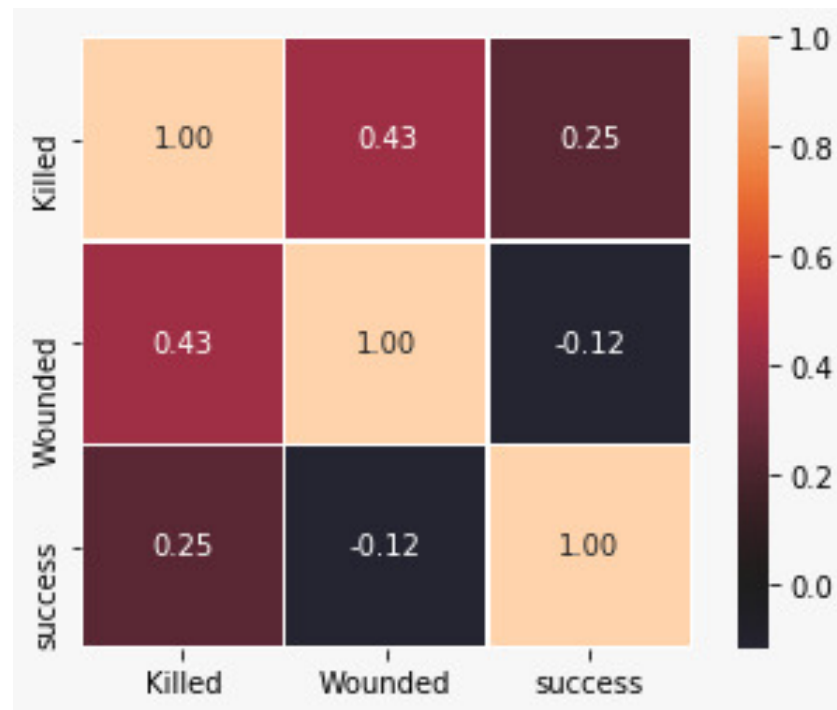
```
1 | df.shape
```

```
Out[16]: (17967, 11)
```

```
1 | df=df[['Killed', 'Wounded', 'success']]
2 | df.index = pd.RangeIndex(len(df.index))
3 | df.dtypes
```

```
Out[19]: Killed      float64
         Wounded      float64
         success      int64
         dtype: object
```

```
1 | sns.heatmap(df.corr(), vmax=1.0, center=0, fm
2 | plt.show()
```



Correlation graph of features can be seen above. There is low correlation between them since they are independent.

1-Gaussian Naïve Bayes

```
1 | from sklearn.naive_bayes import GaussianNB
```

Most of the implementations in scikit-learn library requires numeric attributes in order to apply the models. So, we need utility functions to transform the categorical attributes in our datasets.

Naive Bayesian is known as not very powerful estimation tool and it assumes that variables are independent. However, It performs fast and works well with multiclass predictions. Naive Bayes have probabilistic approach to make classifications.

```
1 features = df.drop(["success"], axis=1).valu
2 target = df["success"].values
3
4 # creating the model
5 clf = GaussianNB()
6
7 # train-test split for the dataset
8 X_train, X_test, y_train, y_test = train_tes
9
10 # fit the training data
11 clf.fit(X_train, y_train)
```

```
Out[76]: GaussianNB(priors=None)
```

The column “success” is labeled as decision label. Other data as features is to predict the decision label. %60 of data will be train set and %40 will be set to evaluate our model.

```
1 | # predict test data
2 | y_pred = clf.predict(X_test)
3 |
4 | # getting the accuracy score
5 | accuracy_score(y_test, y_pred)
```

```
Out[78]: 0.738555725615695
```

```
1 | # let's check the f1 score
2 | f1_score(y_test, y_pred)
```

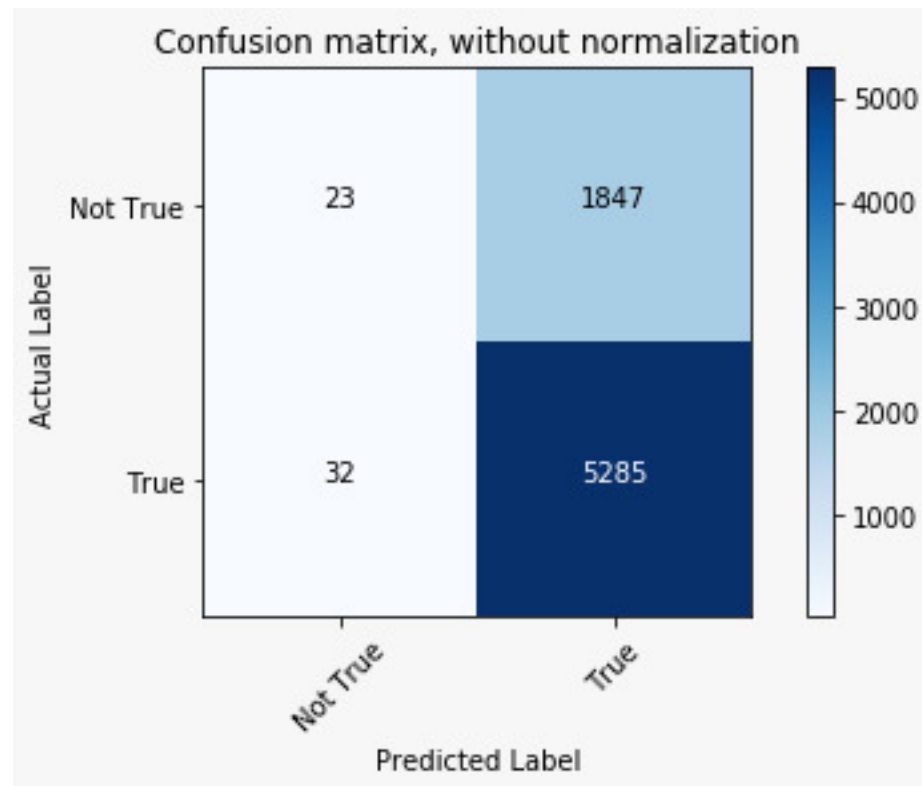
```
Out[79]: 0.8490641818619971
```

Evaluating the model is one of them most important tasks in data science projects which depicts how good your predictions are. A confusion matrix

is a table that often used to describe the performance of a classification model on a set of test data for which true values are known.

Actual Class	Predicted class	
	Class = Yes	Class = No
Class = Yes	True Positive	False Negative
	False Positive	True Negative

```
1 from sklearn.metrics import confusion_matrix
2 cnf_matrix = confusion_matrix(y_test, y_pred)
3 plt.figure()
4 plot_confusion_matrix(cnf_matrix, classes=["N
```



Accuracy: Ratio of correctly predicted observation to the total observation

$$(23+5285)/(23+32+1847+5285)=0.73$$

Precision: Ratio of correctly predicted positive observations to the total predicted positive observations

$$5285/(5285+1847) = 0.74$$

It talks about how precise our model is out of those predicted positive. How many of them are **actual positive**.

Recall(Sensitivity): Ratio of correctly predicted positive observations to all observations in actual class(yes).

It calculates how many of the actual positive of our model capture through labeling it as positive true.

$$5285/(5285+32) = 0.99$$

F1 Score: Weighted average of precision and recall. This might be better measure to use if we need to seek a balance between precision and recall and there is an uneven class distribution. (Large number of actual negatives)

F1 Score = 0.84



Accurate and Precise

1.1 Pros and Cons of Naïve Bayes

Pros

It is easy and fast to predict the class of the test data set. It also performs well in multi-class prediction.

When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.

It perform well in case of categorical input variables compared to numerical variable(s).

Cons

If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction.

On the other side naive Bayes is also known as a bad estimator, so the probability outputs are not to be taken too seriously

.Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

2-Decision Tree without Normalization

Decision trees are supervised Machine Learning algorithms which can be used for both classification and regression problems. Goal of using these algorithm to create a model – which is a tree – that predicts the value of a target variable by learning simple decision rules inferred from the data features.

```
1 | features = df.drop(["success"], axis=1).value
2 | target = df["success"].values
3 |
4 | # train-test split for the dataset
5 | X_train, X_test, y_train, y_test = train_test
```

```
6  
7 from sklearn import tree  
8 model = tree.DecisionTreeClassifier()  
9 model.fit(X_train, y_train)
```

```
Out[16]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,  
                                max_features=None, 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, presort=False, random_state=None,  
                                splitter='best')
```

```
1 y_pred = model.predict(X_test)  
2 accuracy_score(y_test, y_pred)
```

```
Out[18]: 0.9185679836764978
```

```
1 # let's check the f1 score  
2 f1_score(y_test, y_pred)
```

```
Out[19]: 0.9450219160926738
```

```
1 def plot_confusion_matrix(cm, classes,  
2                             normalize=False,  
3                             title='Confusion m  
4                             cmap=plt.cm.Blues)
```

```

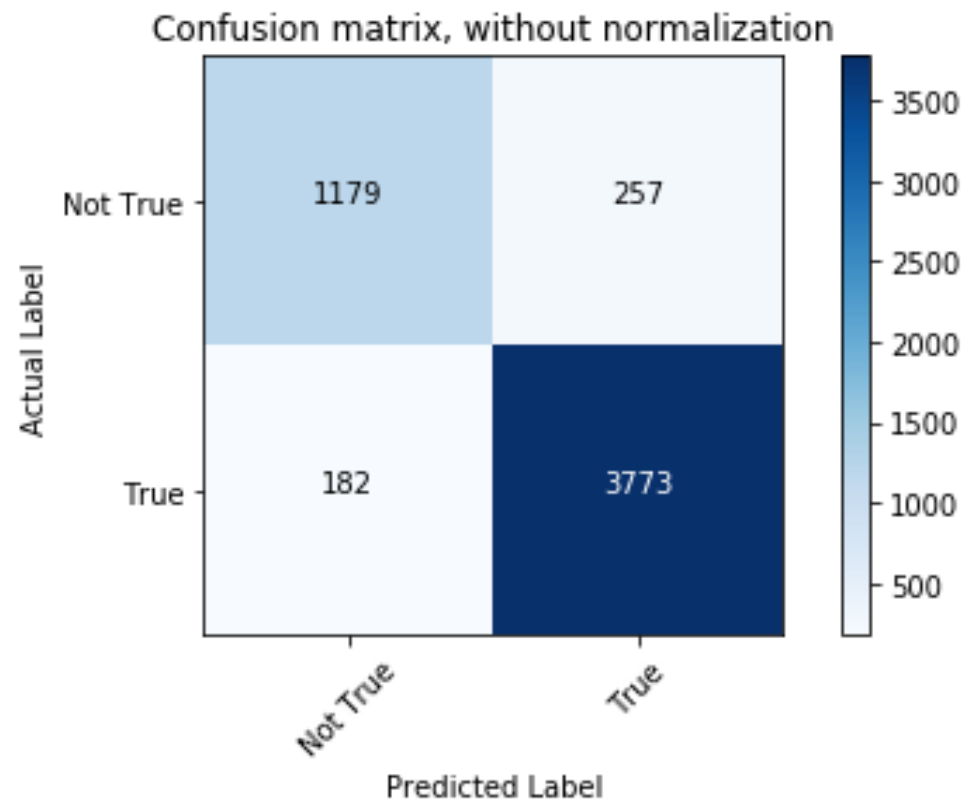
5     """
6     This function prints and plots the confu
7     Normalization can be applied by setting
8     """
9     if normalize:
10         cm = cm.astype('float') / cm.sum(axi
11         print("Normalized confusion matrix")
12     else:
13         print('Confusion matrix, without nor
14
15     print(cm)
16
17     plt.imshow(cm, interpolation='nearest',
18     plt.title(title)
19     plt.colorbar()
20     tick_marks = np.arange(len(classes))
21     plt.xticks(tick_marks, classes, rotation
22     plt.yticks(tick_marks, classes)
23
24     fmt = '.2f' if normalize else 'd'
25     thresh = cm.max() / 2.
26     for i, j in itertools.product(range(cm.s
27         plt.text(j, i, format(cm[i, j], fmt)
28                 horizontalalignment="center
29                 color="white" if cm[i, j] >
30
31     plt.tight_layout()
32     plt.ylabel('Actual Label')

```



```
33 | plt.xlabel('Predicted Label')
```

```
1 | from sklearn.metrics import confusion_matrix  
2 | cnf_matrix = confusion_matrix(y_test, y_pred)  
3 | plt.figure()  
4 | plot_confusion_matrix(cnf_matrix, classes=["N
```



Accuracy: Ratio of correctly predicted observation to the total observation

$$(1179+3773)/(1179+257+182+3773) = 0.91$$

Precision: Ratio of correctly predicted positive observations to the total predicted positive observations

$$3773/(3773+257) = 0.93$$

It talks about how precise our model is out of those predicted positive. How many of them are **actual positive**.

Recall(Sensitivity): Ratio of correctly predicted positive observations to all observations in actual class(yes).

It calculates how many of the actual positive of our model capture through labeling it as positive true.

$$3773/(3773+182) = 0.95$$

F1 Score: F1 is an overall measure of a model's accuracy that combines precision and recall.

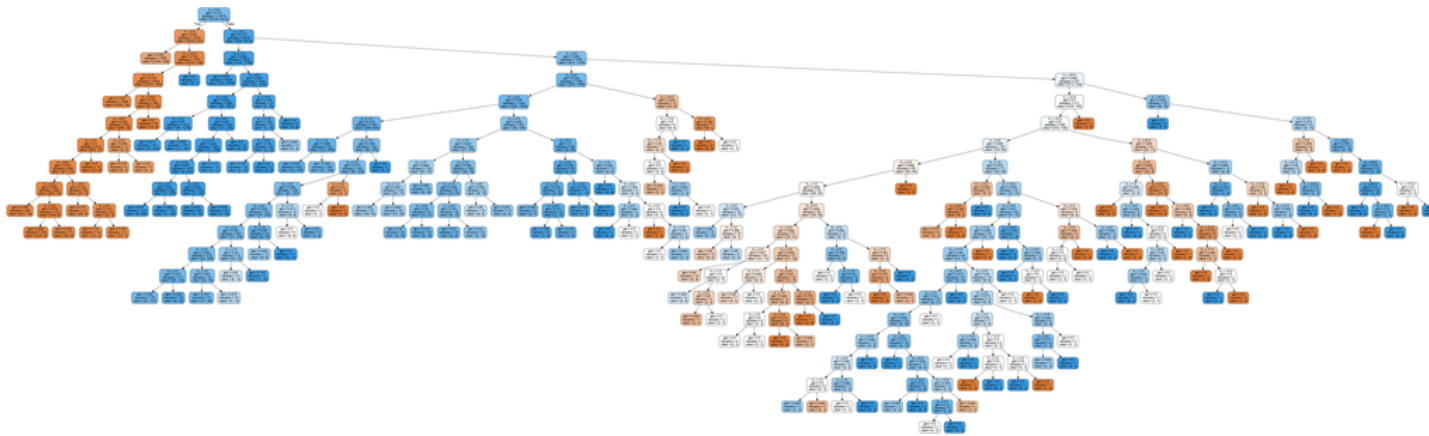
A good F1 score means that you have low false positives and low false negatives, so you're correctly identifying real threats and you are not disturbed by false alarms. An F1 score is considered perfect when it's 1, while the model is a total failure when it's 0.

F1 Score = 0.93



Accurate and Precise

```
1 from sklearn.externals.six import StringIO
2 from IPython.display import Image
3 from sklearn.tree import export_graphviz
4 import pydotplus
5 dot_data = StringIO()
6
7 export_graphviz(model, out_file=dot_data, fi
8
9 graph=pydotplus.graph_from_dot_data(dot_data
10 Image(graph.create_png()))
```



You can find a larger version of the decision tree (model without normalization) below: <https://ibb.co/LxWwf3F>

2.1 Pros and Cons of Decision Tree without Normalization

Pros

Are simple to understand and interpret.

Help determine worst, best and expected values for different scenarios.

Can be combined with other decision techniques.

Cons

They are unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal decision tree.

They are often relatively inaccurate. Many other predictors perform better with similar data.

Calculations can get very complex, particularly if many values are uncertain and/or if many outcomes are linked.

3- Decision Tree with Normalization

```
1 features = df.drop(["success"], axis=1).valu
2 target = df["success"].values
3
4 # train-test split for the dataset
5 X_train, X_test, y_train, y_test = train_tes
6
7 from sklearn import tree
8 model = tree.DecisionTreeClassifier(max_dept
9
10 model.fit(X_train, y_train)
```

```
Out[16]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=6,
                                max_features=None, 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, presort=False, random_state=None,
                                splitter='best')
```

```
1 y_pred = model.predict(X_test)
2 accuracy_score(y_test, y_pred)
```

```
Out[18]: 0.9102207382674828
```

```
1 | # let's check the f1 score
2 | f1_score(y_test, y_pred)
```

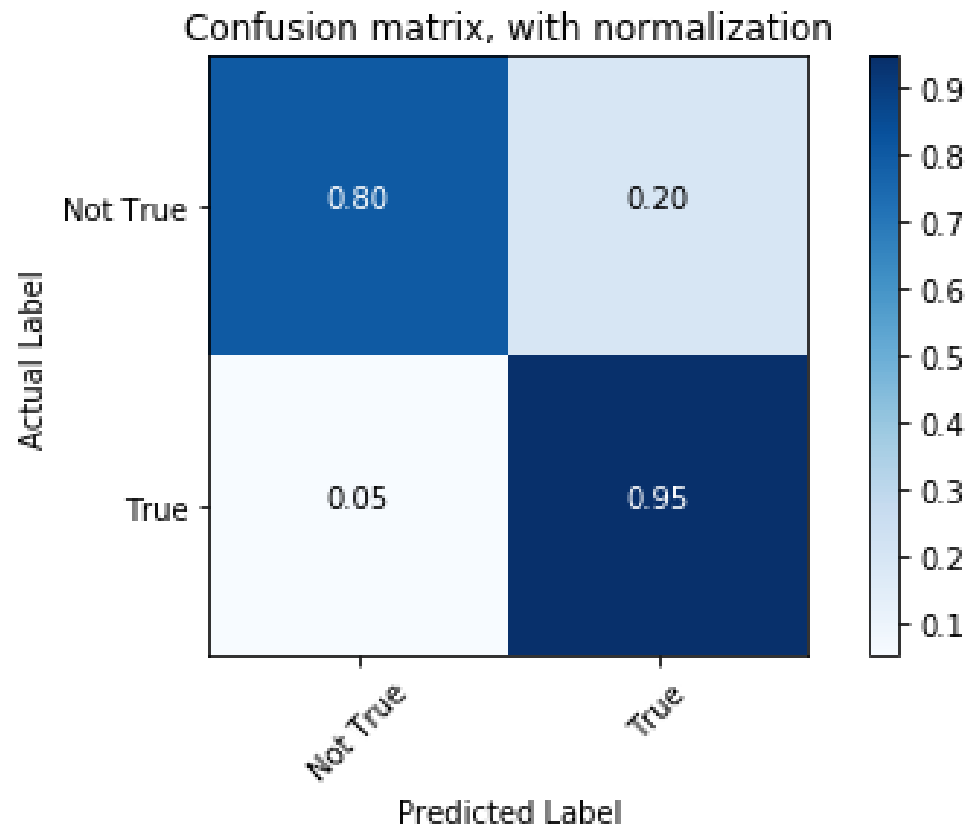
```
Out[19]: 0.9399503722084368
```

```
1 | def plot_confusion_matrix(cm, classes,
2 |                           normalize=False,
3 |                           title='Confusion m
4 |                           cmap=plt.cm.Blues)
5 |     """
6 |     This function prints and plots the confu
7 |     Normalization can be applied by setting
8 |     """
9 |     if normalize:
10 |         cm = cm.astype('float') / cm.sum(
11 |         print("Normalized confusion matrix")
12 |     else:
13 |         print('Confusion matrix, without nor
14 |
15 |     print(cm)
16 |
17 |     plt.imshow(cm, interpolation='nearest',
18 |     plt.title(title)
```

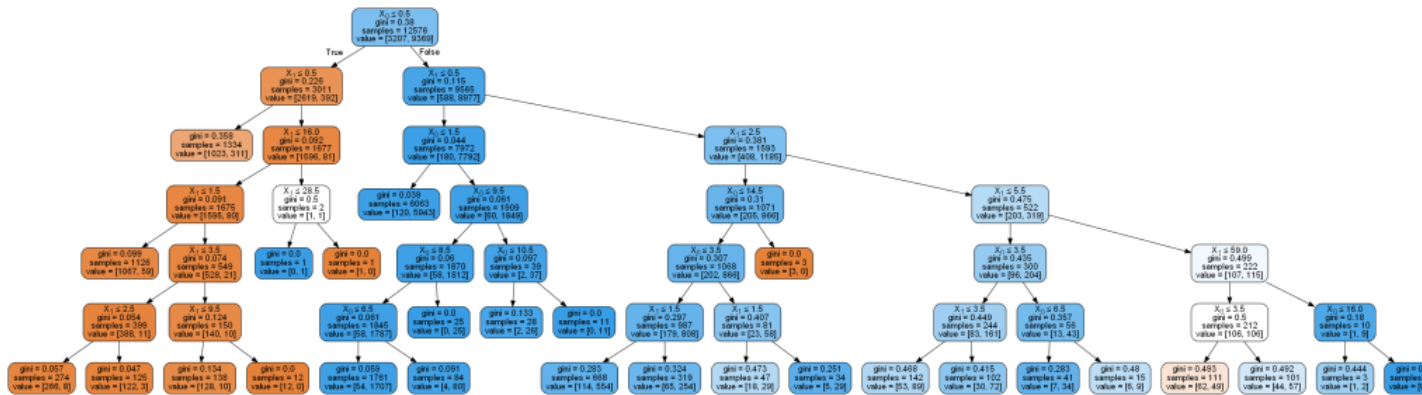


```
19 plt.colorbar()
20 tick_marks = np.arange(len(classes))
21 plt.xticks(tick_marks, classes, rotation
22 plt.yticks(tick_marks, classes)
23
24 fmt = '.2f' if normalize else 'd'
25 thresh = cm.max() / 2.
26 for i, j in itertools.product(range(cm.s
27     plt.text(j, i, format(cm[i, j], fmt)
28             horizontalalignment="center
29             color="white" if cm[i, j] >
30
31 plt.tight_layout()
32 plt.ylabel('Actual Label')
33 plt.xlabel('Predicted Label')
```

```
1 from sklearn.metrics import confusion_matrix
2 cnf_matrix = confusion_matrix(y_test, y_pred)
3 plt.figure()
4 plot_confusion_matrix(cnf_matrix, classes=["N
```



```
1 from sklearn.externals.six import StringIO
2 from IPython.display import Image
3 from sklearn.tree import export_graphviz
4 import pydotplus
5 dot_data = StringIO()
6
7 export_graphviz(model, out_file=dot_data, f
8
9 graph=pydotplus.graph_from_dot_data(dot_data
```



You can find a larger version of the decision tree (model without normalization) below: <https://ibb.co/Ws58Hy5>

We used two models for our predictions:

- Model without normalization
- Normalized model (with maximum depth 6)
- We applied our tests and train sets on two models and got the accuracy score as 0.910 for the model without normalization and 0.918 for the normalized model.

Why did we pick Decision Tree Algorithm ?

- “They easily handle feature interactions and they’re non-parametric, so you don’t have to worry about outliers or whether the data is linearly separable.
- <http://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/>
- It is fast
- The tree can be represented visually

Which ML technique performs better and why ?

Decision Trees are very flexible and easy to apply. They work well with classification problems and regression problems.

Trees will handle both problems. One of the main things about Decision Trees is they only require a table of data and they will build a classifier

directly from that data without needing any up front design work to take place. Naive Bayes requires you build a classification by hand. Picking which features matter is up to you. Naive bayes does quite well when the training data doesn't contain all possibilities so it can be very good with low amounts of data. Decision trees work better with lots of data compared to Naive Bayes.

Decision trees can guide us to find if there is a statistical relationship between a given input to the output and how strong that relationship is. If we are trying to decide between using decision trees vs naive bayes to solve a problem, it is better to test each of them. Which ever performs best will more likely perform better in the field.









Accuracy of Naive Bayes is 0.73 at the same time accuracy for Decision Tree is 0.91. Their F1 Scores are quite high enough. 0.84 and 0.93 respectively. It came out both models are accurate and precise.

Advertisements

AUTOMATTIC

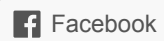
**You don't need to go to
an office to write code.
Work with us!**

APPLY



REPORT THIS AD

Bunu paylaş:



Be the first to like this.

 Global Terrorism Problem  18 May 2019  Genel

Next Post—

Project Evaluation

—Previous Post

**CS 210 Project Step 2 – Data Exploration,
Hypothesis Testing and Single Linear Regression**

Leave a comment

Enter your comment here...

Global Terrorism Problem, Create a free website or blog at WordPress.com.

