City University of New York (CUNY)
CUNY Academic Works

Open Educational Resources | City College of New York

2018

Jupyter: Intro to Data Science - Lecture 9 Decision Trees

Grant Long
CUNY City College

NYC Tech-in-Residence Corps

How does access to this work benefit you? Let us know!

More information about this work at: https://academicworks.cuny.edu/cc_oers/161
Discover additional works at: https://academicworks.cuny.edu

This work is made publicly available by the City University of New York (CUNY).
Contact: AcademicWorks@cuny.edu

Data Dive Week 9: Decision Trees

This week we take a look at *decision trees*, our second type of classification model that brings deeper into the machine learning territory. We'll be using `scikit-learn` in today's exercise.

This week we'll be illustrating how decision trees work using the Titanic survivor dataset available on Kaggle (https://www.kaggle.com/c/titanic/data). We'll look at a create variety of variables to help us learn predict whether a given passenger on the Titanic was able to survive. There is a ton out on the web (including here (https://triangleinequality.wordpress.com/2013/09/08/basic-feature-engineering-with-the-titanic-data/)) about this dataset, as it's a popular among those just coming up to speed on machine learning classification models. Play around and use what you learn in class to join the Kaggle competition (https://www.kaggle.com/c/titanic)!.

Data Dictionary

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg (France), Q = Queenstown (Ireland), S = Southampton (England)

In [ ]:
```
import numpy as np
import pandas as pd
import warnings
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precision_scor
e, confusion_matrix
from sklearn.model_selection import KFold, cross_val_score

# Used for visualizing trees, but not strictly necessary
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

%matplotlib inline
warnings.filterwarnings("ignore")
```

Load and summarize data

In [ ]:
```
df = pd.read_csv('https://grantmlong.com/data/titanic.csv')
```

In [ ]:
```
df.head()
```

In [ ]:
```
df.Survived.describe()
```

Summarize survival by age.

In [ ]:
```
df.loc[(df.Survived==0), 'Age'].hist(bins=20, alpha=.6, color='red', fig
size=[15, 5])
df.loc[(df.Survived==1), 'Age'].hist(bins=20, alpha=.6, color='blue')
```

Summarize survival by sex.

In [ ]:
```
df[['Sex', 'Survived']].groupby('Sex').agg(['mean', 'count'])
```

Find and Count Nulls

In [ ]:
```
df.isna().sum()
```

TODO: Summarize by Pclass, point of embarkment

In [ ]:

In [ ]:

In [ ]:

Data Cleaning and Feature Engineering

Sadly `sci-kit learn` will only let use numeric or boolean variables to train our decision tree, so let's transform some of our variables to address that.

- Create booleans for each of the Embarkment points.
- Create a boolean for is_male.
- Create a boolean for whether someone has a cabin.
- **TODO, time permitting:** create identifiers for passengers in A, B, C, and D cabins

Moreover, some of our ages are missing, so let's enter the missing values as 100 for now.

In [ ]:
```
# Embarkment booleans
for k in df.Embarked.unique():
    if type(k)==str:
        df['emb_' + k] = (df.Embarked==k)*1

# Sex boolean
df['is_male'] = (df.Sex=='male')*1

# Has cabin boolean
df.loc[:, 'has_cabin'] = 0
df.loc[df.Cabin.isna(), 'has_cabin'] = 1

# Age fill
df.loc[df.Age.isna(), 'Age'] = 100

print(list(df))
df.head()
```

Let's assign a list of our clean and model ready features to a list so we can call them easily while training our model.

In [ ]:
```
features = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare',
            'emb_S', 'emb_C', 'emb_Q', 'is_male', 'has_cabin']

valid = df[features].notna().all(axis=1)
print(len(df), sum(valid))
```

In [ ]:

In [ ]:

Building a Decision Tree

Now that we have variables in good shape, we can start modeling. Let's train a simple tree and see how it performs.
Note: for the documentation on `DecisionTreeClassifier`, see here (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html).

In [ ]:
```
dtree=DecisionTreeClassifier(
    criterion='entropy',
    random_state=20181105,
    max_depth=5,
    #min_samples_split=2,
    #min_samples_leaf=1,
    #max_features=None,
    #max_leaf_nodes=None,
)

dtree.fit(df[features], df['Survived'])
```

Visualize the tree. ***Note: there's a strong chance this will not work if you do not have `graphviz` installed.***

For more on visualizing decision trees see here (https://chrisalbon.com/machine_learning/trees_and_forests/visualize_a_decision_tree/), and for more on installing graphviz see here (https://graphviz.gitlab.io). To install `graphviz` on my Macbook Air, I used `brew install graphviz`.

In [ ]:
```
dot_data = StringIO()
export_graphviz(dtree,
                out_file=dot_data,
                filled=True,
                rounded=True,
                feature_names=features,
                special_characters=True
               )
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Calculate metrics from in-sample performance

In [ ]:
```
pred_survival = dtree.predict(df[features])

print(confusion_matrix(df.Survived, pred_survival), '\n')
print('Accuracy:   %0.3f' % accuracy_score(df.Survived, pred_survival))
print('Precision:  %0.3f' % precision_score(df.Survived, pred_survival))
print('Recall:     %0.3f' % recall_score(df.Survived, pred_survival))
```

Wait, are nonlinear models actually doing better here?

- Let's run a logistic regression to compare

In [ ]:
```
logreg = LogisticRegression(random_state=20181105, solver='lbfgs')
logreg.fit(df[features], df['Survived'])
pred_survival = logreg.predict(df[features])

print(confusion_matrix(df.Survived, pred_survival), '\n')
print('Accuracy:   %0.3f' % accuracy_score(df.Survived, pred_survival))
print('Precision:  %0.3f' % precision_score(df.Survived, pred_survival))
print('Recall:     %0.3f' % recall_score(df.Survived, pred_survival))
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Selecting Hyperparameters with Cross Validation

- First, we use the `KFold` function from `sci-kit learn` to generate five folds for cross validation. We can show the balance of the survivor rate among the different folds to get a better idea of what's going on.
- Next, we train a different decision tree model against each of the folds and track our performance.
- Finally, we track average cv metrics for different values of our hyperparameters.

In [ ]:
```
k_fold = KFold(n_splits=5, random_state=20181105)
```

In [ ]:
```
# Print the number of observations and survivor rate  for
for train_indices, test_indices in k_fold.split(df[features]):
    print('Train: n=%i, s_rate=%0.2f | test: n=%i, s_rate=%0.2f ' %
          (df.loc[train_indices, 'Survived'].count(),
           df.loc[train_indices, 'Survived'].mean(),
           df.loc[test_indices, 'Survived'].count(),
           df.loc[test_indices, 'Survived'].mean(),
          )
         )
```

Creating a function to fit our model and return relevant metrics makes it easy to track cross validation performance over different values of our parameters.

In [ ]:
```
def get_cv_results(classifier):
    results = []
    for train, test in k_fold.split(df[features]):
        classifier.fit(df.loc[train, features], df.loc[train, 'Survived'
])
        y_predicted = classifier.predict(df.loc[test, features])
        accuracy = accuracy_score(df.loc[test, 'Survived'], y_predicted)
        results.append(accuracy)

    return np.mean(results), np.std(results)
```

Let's track mean and variance of accuracy for different values of the minimum samples per split.

In [ ]:
```
hp_values = [2, 5, 7, 10, 15, 20, 50, 60, 70, 80, 90, 100, 120, 150]
all_mu = []
all_sigma = []

for m in hp_values:

    dtree=DecisionTreeClassifier(
        criterion='entropy',
        random_state=20181105,
        min_samples_split=m,
        #max_depth=m,
        #min_samples_leaf=m,
        #max_features=m,
        #max_leaf_nodes=m,
    )

    mu, sigma = get_cv_results(dtree)
    all_mu.append(mu)
    all_sigma.append(sigma)

    print(m, mu, sigma)
```

In [ ]:
```
plt.figure(figsize=(14, 5))
plt.plot(hp_values, all_mu)
plt.ylabel('Cross Validation Accuracy')
plt.xlabel('Minimum Samples Per Leaf')
```

In [ ]:
```
plt.figure(figsize=(14, 5))
plt.plot(hp_values, all_sigma)
plt.ylabel('Cross Validation Std Dev.')
plt.xlabel('Minimum Samples Per Leaf')
```

In [ ]:
```
```

Pretty cool, right? We can take a quick look again at how these results compare to logistic regression.

- What do you make of these results?
- Is this a better model? Why or why not?

In [ ]:
```
logreg = LogisticRegression(random_state=20181105, solver='lbfgs')
get_cv_results(logreg)
```

In [ ]:
```

```

In [ ]:
```

```

Selecting Our Model and Applying It to Our Test Set

From this, it seems like `min_samples_split=70` might provide our best fit. We can train our best model using that value.

We can then read in our holdout test set from the Kaggle competition to enter our predictions. We'll first double check and see if our model makes sense by taking a closer look at our predictions.

In [ ]:
```
dtree=DecisionTreeClassifier(
        criterion='entropy',
        random_state=20181105,
        min_samples_split=90,
    )

# Here we train our final model against all of our validation data.
dtree.fit(df.loc[:, features], df.loc[:, 'Survived'])
```

Read in our test data and apply the same transformations as our training set.

In [ ]:
```
test_df = pd.read_csv('https://grantmlong.com/data/titanic_test.csv')

# Embarkment booleans
for k in test_df.Embarked.unique():
    if type(k)==str:
        test_df['emb_' + k] = (test_df.Embarked==k)*1

# Sex boolean
test_df['is_male'] = (test_df.Sex=='male')*1

# Has cabin boolean
test_df.loc[:, 'has_cabin'] = 0
test_df.loc[test_df.Cabin.isna(), 'has_cabin'] = 1

# Age fill
test_df.loc[test_df.Age.isna(), 'Age'] = 100

# Fare fill
test_df.loc[test_df.Fare.isna(), 'Fare'] = test_df.loc[test_df.Fare.notn
a(), 'Fare'].median()

print(list(test_df))
test_df.head()
```

Rank the most likely to survive according to our model.

In [ ]:
```
# Calculate the probability of
test_probabilities = dtree.predict_proba(test_df[features])[:,1]
test_df['survival_likelihood'] = test_probabilities

readable_features =  ['Name', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch',
                      'Ticket', 'Fare', 'Cabin', 'Embarked', 'survival_l
ikelihood']

# Find the rankings based on the probabilities
probability_rankings = np.argsort(test_probabilities)
```

Most Likely to Survive:

In [ ]:
```
test_df.loc[probability_rankings[-20:], readable_features]
```

Most Likely to Die:

In [ ]:
```
test_df.loc[probability_rankings[:20], readable_features]
```

In [ ]:
```

```

In [ ]:

In [ ]:

In [ ]: