

HW3 - Michael Lepore

1.

(15 points) Exercise 3.8: The following table summarizes a data set with three different attributes A, B, C and two class labels +, -. Build a two-level decision tree.

a) According to the classification error rate, which attribute would be chosen as the first splitting attribute? For each attribute, show the contingency table and the gains in classification error rate.

Overall error rate (before splitting), is 50/100 (since we have 50 right answers and 50 wrong answers).

If we split on A we get the following results:

	A=T	A=F
+	25	25
-	0	50

Error Rate = Number wrong / Number total

$$E_{A=T} = 0$$

$$E_{A=F} = 25 / 75$$

$$E = E_{A=T} * (25 / 100) + E_{A=F} * (75 / 100)$$

$$E = 0 + (25 / 75) * (75 / 100)$$

$$E = 25/100 = .25$$

$$E_{orig} - E_A = .50 - .25 = .25$$

$$\text{Change for Splitting on A} = .25$$

If we split on B

	B=T	B=F
+	30	30
-	20	20

$$E_{B=T} = 20/50$$

$$E_{B=F} = 20/50$$

$$E = E_{B=T} * (50 / 100) + E_{B=F} * (50 / 100)$$

$$E = (20/50)*(50/100) + (20 / 50) * (50 / 100)$$

$$E = 40/100 = .40$$

$$E_{orig} - E_B = .50 - .40 = .10$$

Change for splitting on B = .10

If we split on C

	C=T	C=F
+	25	25
-	25	25

$$E_{B=T} = 25/50$$

$$E_{B=F} = 25/50$$

$$E_C = E_{C=T} * (50 / 100) + E_{C=F} * (50 / 100)$$

$$E_C = (25/50)*(50/100) + (25 / 50) * (50 / 100)$$

$$E_C = 50/100 = .50$$

$$E_{orig} - E_C = .50 - .50 = 0$$

$$\Delta C = 0$$

Based on the classification rate algorithm, we should split on A

(b) (3 points) Repeat for the two children of the root node.

On the A=T side, we don't need to do any splitting at all, since they are all of class Positive.

0/25

On the A=F side, splitting on B

	A=F, B=T	A=F, B=F
+	25	0
-	20	30

So the error rate on A=F, B=T is

$$E_{A=F, B=T} = 20/45$$

Our new overall error rate is:

$$E = 20/45 * 45/75 + 0$$

$$E_{A=F} = 20/75$$

$$E_{\text{total}} = 0/25 * 25/100 + 20 / 75 * 75 / 100 = 20 / 100 = .20$$

So our gain is 0.05 based on the second layer

(c) (3 points) How many instances are misclassified by the resulting decision tree?

20

(d) (3 points) Repeat parts (a), (b), and (c) using C as the splitting attribute.

We already computed E_C and ΔC As part of (a). So we know that the table looks like:

	C=T	C=F
+	25	25
-	25	25

From that split, here are the tables for B

	C=T,B=T	C=T,B=F
+	5	20
-	20	5

And for A (on the C=T) side:

	C=T,A=T	C=T,A=F
+	25	0
-	0	25

$$E_{C=T,A} = 0/25 * 25 / 50 + 0 / 25 * 25 / 50 = 0/50$$

	C=F,B=T	C=F,B=F
+	25	0
-	0	25

$$E_{C=F} = 0/25 * 25 / 50 + 0 / 25 * 25 / 50 = 0/50$$

So if we split on C, and then on the C=T, we split on A, and C=F we split on B, we can get a 0% error rate.

(e) (3 points) Use the results in parts (c) and (d) to conclude about the greedy nature of the decision tree induction algorithm.

Based on this data, we aren't always going to get an optimal split for decision trees - because we take the highest gain even if there is a better split later. That means that our trees might be too deep to get to the gain we want.

2. (10 points) Exercise 3.12:

(a) (5 points) Based on the accuracies shown in the table above, which classification model would you expect to have better performance on unseen instances?

The models were run on a previously unseen dataset (the validation set, in this case B). So the results for data set B should be similar to the results for an unseen set.

Therefore I'd expect T10 to be the better performer - seems like T100 is overfit.

(b) (5 points) Now, you tested T_{10} and T_{100} on the entire data set $pA \cup B$ and found that the classification accuracy of T_{10} on data set $pA \cup B$ is 0.85, whereas the classification accuracy of T_{100} on the data set $pA \cup B$ is 0.87. Based on this new information and your observations from Table 3.7, which classification model would you finally choose for classification?

I would still consider T10 to be the better performer. Since A and B are the same size, the overall results should just be the average of the results on A and B. I still expect that model T10 is better for unseen data.

3. Exercise 3.13 : Summarize the performance of the classifiers given in Table 3.8.

Data Set	Size	Decision Tree	Bayes	SVM	Z DT vs Bayes	Z Bayes vs SVM	Z SVM vs DT	Highest /Lowest Z
Anneal	898	92.09%	79.62%	87.19%	7.58	-4.31	-3.41	7.58
Australia	690	85.51%	76.81%	84.78%	4.13	-3.76	-0.38	4.13
Auto	205	81.95%	58.05%	70.73%	5.28	-2.68	-2.67	5.28
BCleve	303	76.24%	83.50%	84.49%	-2.23	-0.33	2.56	2.56
Breast	699	95.14%	95.99%	96.42%	-0.77	-0.42	1.19	1.19
Credit	690	85.80%	77.54%	85.07%	3.97	-3.59	-0.38	3.97
Diabetes	768	72.40%	75.91%	76.82%	-1.57	-0.42	1.99	1.99
German	1000	70.90%	74.70%	74.40%	-1.91	0.15	1.76	1.91
Glass	214	67.29%	48.59%	59.81%	3.92	-2.33	-1.61	3.92
Heart	270	80.00%	84.07%	83.70%	-1.23	0.12	1.12	1.23
Hepatitis	155	81.94%	83.23%	87.10%	-0.30	-0.96	1.26	1.26
Horse	368	85.33%	78.80%	82.61%	2.31	-1.31	-1.01	2.31
Ionosphere	351	94.67%	82.34%	88.89%	5.12	-2.47	-2.79	5.12
Iris	150	78.95%	95.33%	96.00%	-4.24	-0.28	4.46	4.46
Labor	57	73.34%	94.74%	92.98%	-3.12	0.39	2.80	3.12
Led7	3200	77.03%	73.16%	73.56%	3.58	-0.36	-3.22	3.58
Lymphography	148	74.35%	83.11%	86.49%	-1.84	-0.81	2.63	2.63
PIMA	768	74.35%	76.04%	76.95%	-0.77	-0.42	1.19	1.19
Sonar	208	78.85%	69.71%	76.92%	2.13	-1.66	-0.47	2.13
Tic-tac-toe	958	83.72%	70.04%	98.33%	7.10	-16.97	11.19	16.97
Vehicle	846	71.04%	45.04%	74.94%	10.84	-12.55	1.81	12.55
Wine	178	94.38%	96.63%	98.88%	-1.02	-1.43	2.35	2.35
Zoo	101	93.07%	93.07%	96.04%	0.00	-0.93	0.93	0.93

Observation Summaries	
	There are only 2 datasets where SVM performs significantly better than the others (Wine, Tic-Tac-Toe)
	There is only 1 dataset where bayes performs significantly better than the other models (Labor)
	There are 7 datasets where we don't see a significant performance difference between the models.
	For the other datasets, picking a decision tree seems like a great starting point

CS548 Homework 3 Michael Lepore

Loading data, and analysis

```
In [67]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.model_selection import (cross_validate, KFold, ShuffleSplit)
import math

train_data = pd.read_csv("train.csv")
train_data
```


Out [67]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376

891 rows x 12 columns

From what I know of the Titanic, I'm going to intuit that perhaps "women and children" were the first ones off, along with folks that maybe paid more.

So, lets see if there's any relationship between age, sex, fare (how much they paid) and survival

```
In [68]: train_data.groupby('Sex').Survived.mean().plot(kind='bar')
plt.title("Survival based on sex")
plt.xlabel("Sex")
plt.ylabel("Percentage Survived")
plt.show()

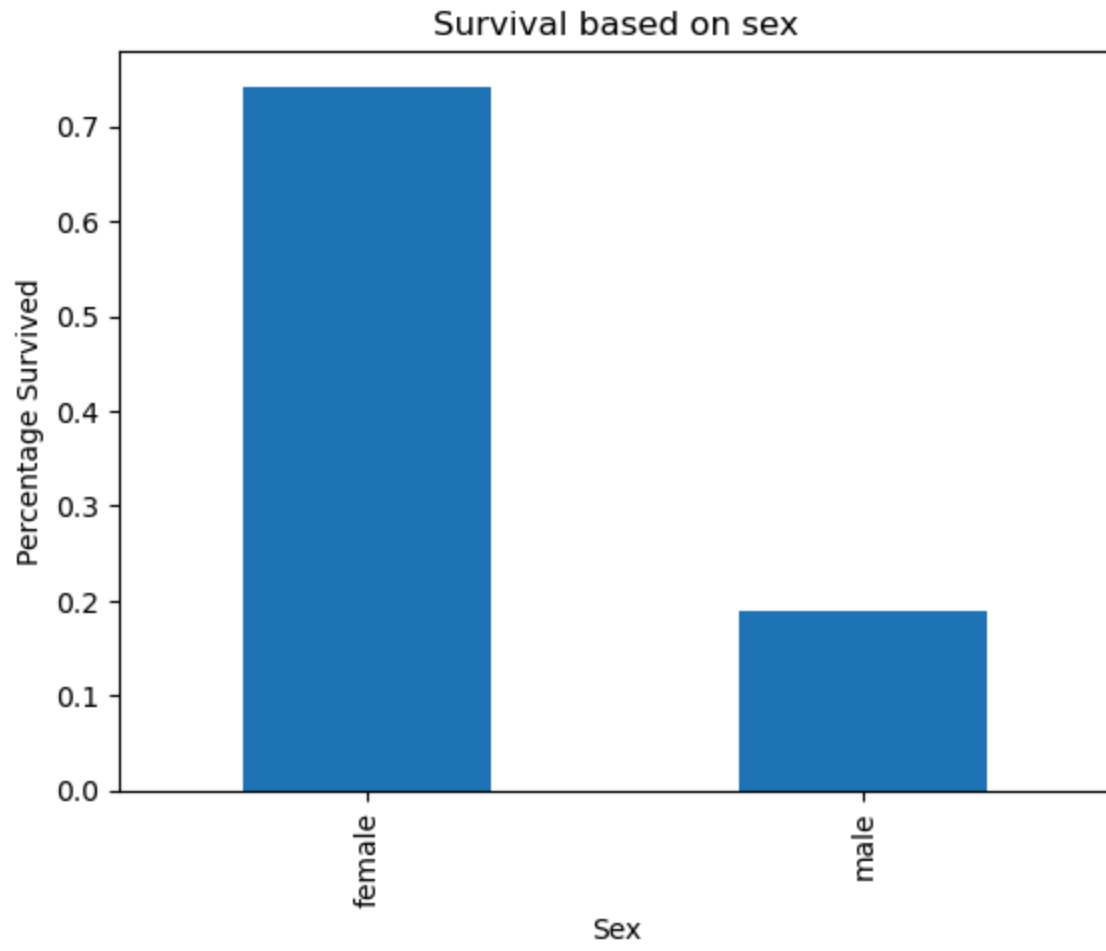
# Define bins for value ranges
agebins = [0, 20, 40, 60, 80, 100]
train_data['age_bin'] = pd.cut(train_data['Age'], bins=agebins, right=True)

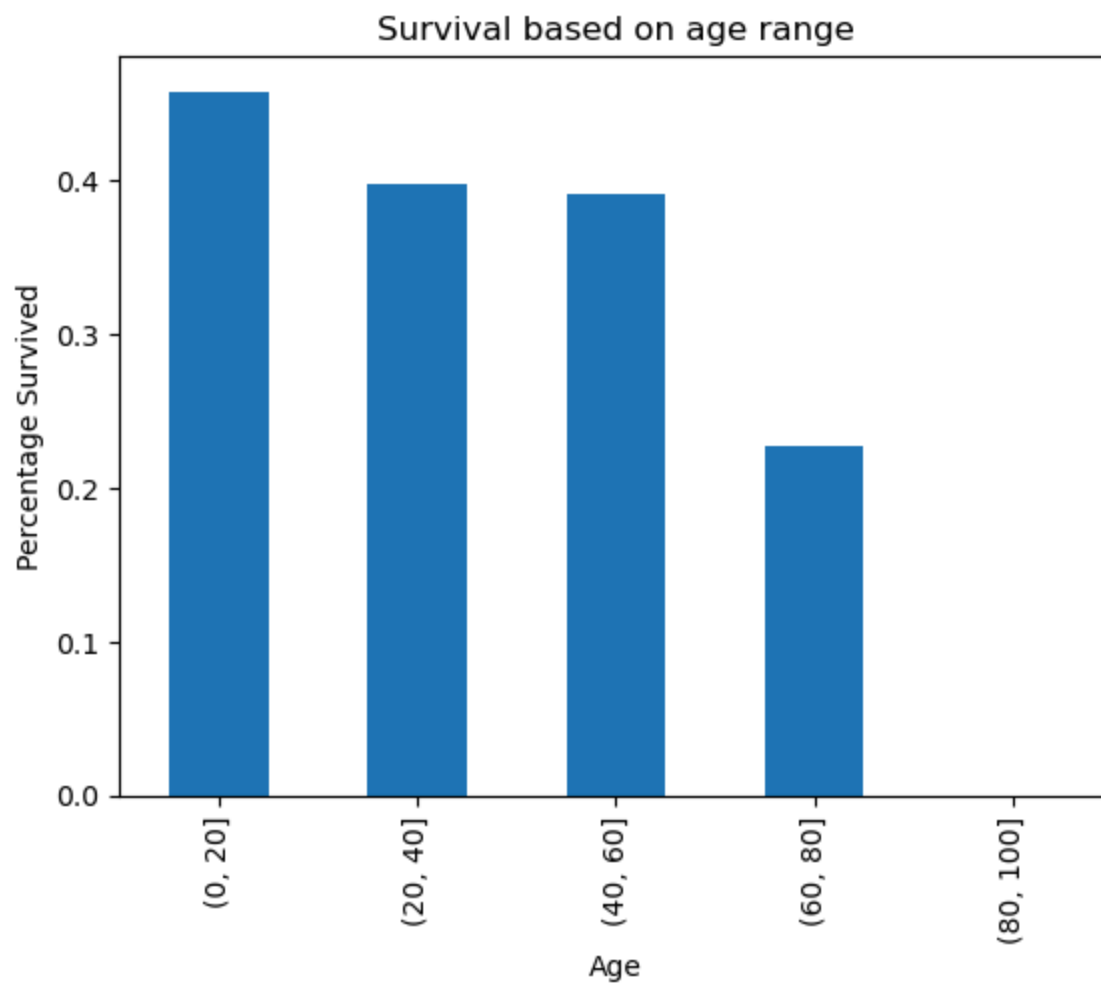
train_data.groupby('age_bin', observed=False).Survived.mean().plot(kind='bar')
plt.title("Survival based on age range")
plt.xlabel("Age")
plt.ylabel("Percentage Survived")
plt.show()

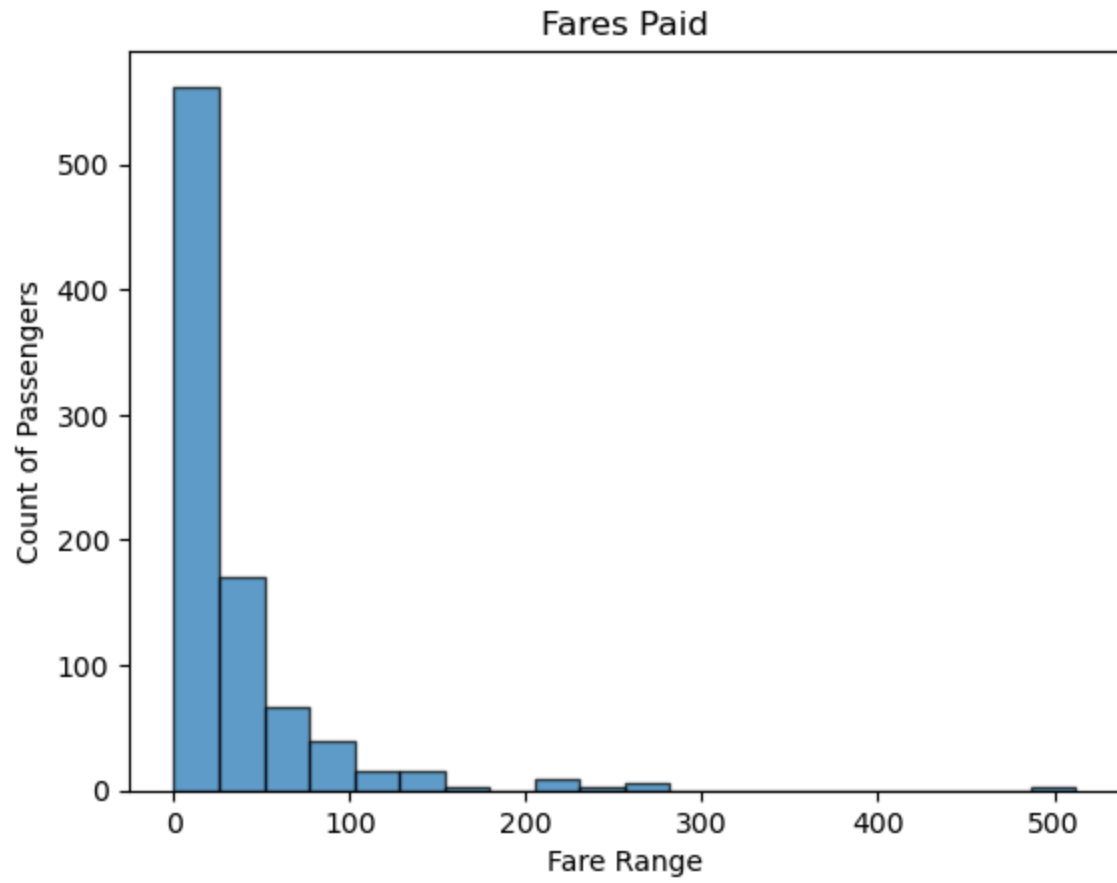
plt.hist(train_data['Fare'], bins=20, edgecolor='black', alpha=0.7)
plt.title("Fares Paid")
plt.xlabel("Fare Range")
plt.ylabel("Count of Passengers")
plt.show()

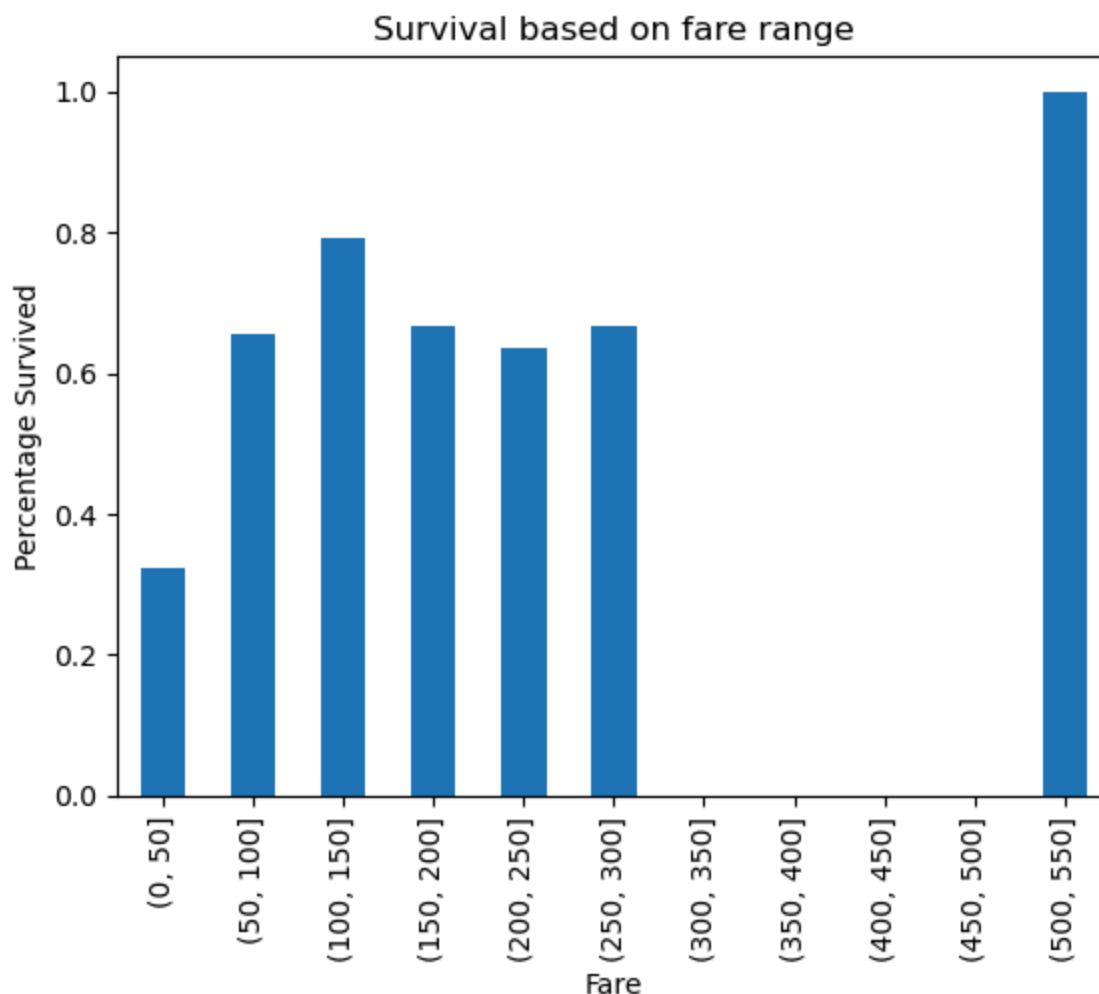
farebins = range(0, 600, 50)
train_data['fare_bin'] = pd.cut(train_data['Fare'], bins=farebins, right=True)

train_data.groupby('fare_bin', observed=False).Survived.mean().plot(kind='bar')
plt.title("Survival based on fare range")
plt.xlabel("Fare")
plt.ylabel("Percentage Survived")
plt.show()
```









Building the first model

So this tells us that there is definitely a relationship with each of the values here. After the homework exercises seemed to suggest that decision trees are a good starting point, let's start with those. Originally I thought about using a `RandomForestClassifier`, but our dataset is fairly simple, and it's not very large.

We will start with the following columns:

- age
- sex
- fare paid

and see how accurate that will be

My original version of this was doing `dropna()` so that we'd get rid of any numbers that were NA values. That won't work because our testing data has NAs in it. Let's fill in those as well. For now, let's just use zeros.

```
In [69]: # OK, lets convert our dataset to make sure we have numbers for everything -
# We'll need to do this same processing on our submission data later, so let
def process_data(df):
    df = df.copy();
    df['sex_int'] = df['Sex'].map({'male': 0, 'female': 1})
    df['Age'] = df['Age'].fillna(0);
    df['Fare'] = df['Fare'].fillna(0);
    return df

train_data_processed = process_data(train_data)

y = train_data_processed.dropna()['Survived']
X = train_data_processed.dropna().drop('Survived', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train.shape
```

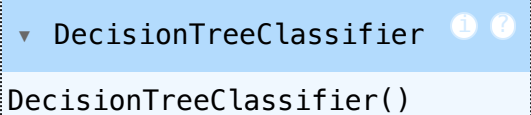
Out[69]: (144, 14)

Lets build an SVC model based on the columns we want to try

```
In [70]: from sklearn.tree import DecisionTreeClassifier

columns = ['sex_int', 'Fare', 'Age']

decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train[columns], y_train)
```

Out[70]: 

```
In [71]: # Lets see how we did
score = decision_tree.score(X_test[columns], y_test)
print(score)
```

0.7567567567567568

Now lets run our testing data through the same thing and create a prediction file

```
In [72]: test_file = pd.read_csv("test.csv")
# print(test_file)

test_data = process_data(test_file)
answers = decision_tree.predict(test_data[columns])

final_data = pd.DataFrame()
final_data['PassengerId'] = test_data['PassengerId']
final_data['Survived'] = answers

final_data.to_csv('decisiontree_output1.csv', index=False)
```

Model Results

Ok. So that model - against our segmented test data did ok (~70%). But against the data in the competition did significantly worse (56%). I'd like to try some things things:

- Change our model - maybe use an SVM
- Check our hyper-parameters on the model
- Add some more fields to the training set (Maybe working with Class of Service, And # of family members (both parents and children/spouses))
- If we can't figure anything else out, maybe we can change the way we predict the missing age data - since that seems to have a pretty big impact based on our exploration - maybe there are a ton of zeroes in the test data?

```
In [73]: # So lets try a bunch of SVMs - we'll try all the kernels to see which is be
# Lets play with some hyperparameters, and the same data
from sklearn.svm import SVC

for kernel in ["linear", "poly", "rbf", "sigmoid"] :
    svm_kernel = SVC(kernel=kernel)
    svm_kernel.fit(X_train[columns], y_train)
    score = svm_kernel.score(X_test[columns], y_test)
    print(kernel + " Kernel Score: ", score)
```

```
linear Kernel Score:  0.7567567567567568
poly Kernel Score:   0.5405405405405406
rbf Kernel Score:    0.5405405405405406
sigmoid Kernel Score: 0.5405405405405406
```

Ok, using the same columns for predictions as a decision tree, our svm seems to give slightly better performance (75% vs 70%) with the linear kernel.

So at least for the kernel, our guess at using a linear kernel seems to be right with the other default hyperparameters. The other main hyperparameter seems to be C

```
In [74]: for c in [ 1.0, 5.0, 10.0, 50.0, 100.0 ] :
    svm_c = SVC(kernel='linear', C=c)
    svm_c.fit(X_train[columns], y_train)
    score = svm_c.score(X_test[columns], y_test)
    print(c, " C score: ", score)
```

```
1.0  C score:  0.7567567567567568
5.0  C score:  0.7567567567567568
10.0 C score:  0.7567567567567568
50.0 C score:  0.7567567567567568
100.0 C score: 0.7567567567567568
```

OK. So that didn't seem to make any difference at all. Lets move onto changing our data

```
In [75]: columns2 = ['sex_int', 'Fare', 'Age', 'SibSp', 'Parch']
svm_model2 = SVC(kernel='linear', C=1.0)
svm_model2.fit(X_train[columns2], y_train)
```



```
score = svm_model2.score(X_test[columns2], y_test)
print("Added columns, score = ", score)
```

Added columns, score = 0.7297297297297297

So that is really interesting - we have actually made the model worse by adding those fields. What if we also add Pclass

```
In [76]: columns2 = ['sex_int', 'Fare', 'Age', 'SibSp', 'Parch', 'Pclass']
svm_model2 = SVC(kernel='linear', C=1.0)
svm_model2.fit(X_train[columns2], y_train)
score = svm_model2.score(X_test[columns2], y_test)
print("Added columns, score = ", score)
```

Added columns, score = 0.7567567567567568

Ok, back to ~75%. Maybe we have some other fields that are making things worse... can we check to see which fields are useful and which aren't?

```
In [77]: for i in range(0,6):
        cols=columns2.copy()
        dropped = cols[i]
        del cols[i]
        svm_model2 = SVC(kernel='linear', C=1.0)
        svm_model2.fit(X_train[cols], y_train)
        score = svm_model2.score(X_test[cols], y_test)
        print("Dropped Col = ", dropped, ", score = ", score)
```

```
Dropped Col = sex_int , score = 0.5405405405405406
Dropped Col = Fare , score = 0.7567567567567568
Dropped Col = Age , score = 0.7567567567567568
Dropped Col = SibSp , score = 0.7567567567567568
Dropped Col = Parch , score = 0.7297297297297297
Dropped Col = Pclass , score = 0.7297297297297297
```

Seems like the most important columns are sex, Parch and Pclass, what happens if we just use those?

```
In [78]: columns3 = ['sex_int', 'Parch', 'Pclass']
svm_model3 = SVC(kernel='linear', C=1.0)
svm_model3.fit(X_train[columns3], y_train)
score = svm_model3.score(X_test[columns3], y_test)
print("Only 3 columns, score = ", score)
```

Only 3 columns, score = 0.7567567567567568

Filing in the missing data

Seems really strange that age doesn't seem predictive - especially given what we saw in the original data. Right now we're just setting age to zero if we don't know it. What if we set it to the mean of age, or if we try and predict the age from the other values - seems

like the number of siblings/parents might have some value here. While we are at it, set the fare to mean as well.

```
In [79]: # OK, lets convert our dataset to make sure we have numbers for everything -
# We'll need to do this same processing on our submission data later, so let
def process_data_mean(df):
    df = df.copy();
    df['sex_int'] = df['Sex'].map({'male': 0, 'female': 1})
    df['Age'] = df['Age'].fillna(df['Age'].mean());
    df['Fare'] = df['Fare'].fillna(df['Fare'].mean());
    return df

train_data_processed = process_data_mean(train_data)
y = train_data_processed.dropna()['Survived']
X = train_data_processed.dropna().drop('Survived', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train.shape
```

Out[79]: (144, 14)

```
In [80]: columns4 = ['sex_int', 'Parch', 'Pclass', 'Age']
svm_model4 = SVC(kernel='linear', C=1.0)
svm_model4.fit(X_train[columns4], y_train)
score = svm_model4.score(X_test[columns4], y_test)
print("Age set to mean, score = ", score)
```

Age set to mean, score = 0.7297297297297297

OK, now that is promising. Moving the needle a little bit. What about Fare?

```
In [81]: columns5 = ['sex_int', 'Parch', 'Pclass', 'Fare']
svm_model5 = SVC(kernel='linear', C=1.0)
svm_model5.fit(X_train[columns5], y_train)
score = svm_model5.score(X_test[columns5], y_test)
print("Fare set to mean, score = ", score)
```

Fare set to mean, score = 0.7567567567567568

Now with both?

```
In [82]: columns6 = ['sex_int', 'Parch', 'Pclass', 'Fare', 'Age']
svm_model6 = SVC(kernel='linear', C=1.0)
svm_model6.fit(X_train[columns6], y_train)
score = svm_model6.score(X_test[columns6], y_test)
print("Age and Fare set to mean, score = ", score)
```

Age and Fare set to mean, score = 0.7567567567567568

Well, lets see if that helps at all with our scoring in the other test data

```
In [83]: test_file = pd.read_csv("test.csv")

test_data = process_data(test_file)
answers = svm_model6.predict(test_data[columns6])
```

```
final_data = pd.DataFrame()
final_data['PassengerId'] = test_data['PassengerId']
final_data['Survived'] = answers

final_data.to_csv('model6_output.csv', index=False)
```

Ok. Last try - lets use the new updated data and columns and see if a decision tree would do better (and maybe our SVM isn't the right model):

```
In [84]: columns7 = ['sex_int', 'Parch', 'Pclass', 'Fare', 'Age']
decision_tree7 = DecisionTreeClassifier()
decision_tree7.fit(X_train[columns7], y_train)
score = decision_tree7.score(X_test[columns7], y_test)
print("Decision Tree with update columns = ", score)

test_file = pd.read_csv("test.csv")

test_data = process_data(test_file)
answers = decision_tree7.predict(test_data[columns7])

final_data = pd.DataFrame()
final_data['PassengerId'] = test_data['PassengerId']
final_data['Survived'] = answers

final_data.to_csv('model7_output.csv', index=False)
```

Decision Tree with update columns = 0.7027027027027027

Score doesn't seem great, lets see our submission... Only 54%, so we're definitely doing better with the SVM. Going to call it here.

Leaderboard position for first submission - with decision tree

Search

Titanic - Machine Learning from Disaster

Submit Prediction

...

Overview

Data

Code

Models

Discussion

Leaderboard

Rules

Team

Submissions

15165	Youssef Elebiary		0.56220	1	2mo
15166	Michael Lepore		0.56220	4	13s
<div><div></div><div>Your Best Entry! Your submission scored 0.55980, which is not an improvement of your previous score. Keep trying!</div></div>					
15167	Kashif Mohammad		0.55980	1	2mo
15168	Tiến Thành Nguyễn		0.55741	1	1mo
15169	Junhyuk Kwon123		0.55502	1	23d
15170	Abnaz Hyderabadwalla		0.55023	2	2mo
15171	Henry Kravisc		0.55023	1	2mo
15172	mdaugsid		0.55023	1	1mo
15173	Prisha Birla		0.55023	1	1mo

Leaderboard position for SVM that has been tuned

🔍 Search



Titanic - Machine Learning from Disaster

Submit Prediction



Overview Data Code Models Discussion **Leaderboard** Rules Team Submissions

15153	paresh494		0.57655	1	20d
15154	Michael Lepore		0.57655	5	22s
<div> Your Best Entry! Your most recent submission scored 0.57655, which is an improvement of your previous score of 0.56220. Great job!</div> <div>Tweet this</div>					
15155	data_set#69420		0.57416	1	20d
15156	Hamza Kamelen		0.57416	2	20d
15157	CloverPark		0.57416	5	3d
15158	matunoki1234		0.57177	1	6d
15159	Junya Shozui		0.57177	1	3d
15160	東純ノ介		0.57177	1	3d
15161	Holly Lynch		0.56937	6	1mo