mldb.ai
(/)

This page is part of the documentation for the Machine Learning Database (http://mldb.ai).

It is a static snapshot of a Notebook which you can play with interactively by trying MLDB online now (/doc/#builtin/Running.md.html).
It's free and takes 30 seconds to get going.

Predicting Titanic Survival

From the description of a Kaggle Machine Learning Challenge at https://www.kaggle.com/c/titanic (https://www.kaggle.com/c/titanic)

> 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.

In this demo we will use MLDB to train a classifier to predict whether a passenger would have survived the Titanic disaster.

Initializing pymldb and other imports

In this demo, we will use pymldb to interact with the REST API (../../../../doc/#builtin/WorkingWithRest.md.html): see the Using pymldb Tutorial (../../../../doc/nblink.html#_tutorials/Using pymldb Tutorial) for more details.

In [1]:
```
from pymldb import Connection
mldb = Connection("http://localhost")

#we'll need these also later!
import numpy as np
import pandas as pd, matplotlib.pyplot as plt, seaborn, ipywidgets
%matplotlib inline
```

Checking out the Titanic dataset

From https://www.kaggle.com/c/titanic (https://www.kaggle.com/c/titanic)

Load up the data

See the Loading Data Tutorial (../../../../doc/nblink.html#_tutorials/Loading Data Tutorial) guide for more details on how to get data into MLDB.

In [2]:
```
mldb.put('/v1/procedures/import_titanic_raw', {
    "type": "import.text",
    "params": {
        "dataFileUrl": "http://public.mldb.ai/titanic_train.csv",
        "outputDataset": "titanic_raw",
        "runOnCreation": True
    }
})
```

Out[2]:

PUT http://localhost/v1/procedures/import_titanic_raw

201 Created


```
{
 "status": {
  "firstRun": {
   "runStarted": "2017-01-24T22:01:37.3471007Z",
   "status": {
    "rowCount": 891,
    "numLineErrors": 0
   },
   "runFinished": "2017-01-24T22:01:37.3718717Z",
   "id": "2017-01-24T22:01:37.346948Z-463496b56263af05",
   "state": "finished"
  }
 },
 "config": {
  "params": {
   "outputDataset": "titanic_raw",
   "runOnCreation": true,
   "dataFileUrl": "http://public.mldb.ai/titanic_train.csv"
  },
  "type": "import.text",
  "id": "import_titanic_raw"
 },
 "state": "ok",
 "type": "import.text",
 "id": "import_titanic_raw"
}
```

Let's look at the data

See the Query API (../../../../doc/#builtin/sql/QueryAPI.md.html) documentation for more details on SQL queries.

In [3]:
```
mldb.query("select * from titanic_raw limit 5")
```

Out[3]:

	Age	Embarked	Fare	Name	Parch	PassengerId	Pclass	Sex	SibSp	Ticke
_rowName										
2	22	S	7.2500	BraundMr.OwenHarris	0	1	3	male	1	A/521
3	38	C	71.2833	CumingsMrs.JohnBradley(FlorenceBriggsThayer)	0	2	1	female	1	PC17
4	26	S	7.9250	HeikkinenMiss.Laina	0	3	3	female	0	STON
5	35	S	53.1000	FutrelleMrs.JacquesHeath(LilyMayPeel)	0	4	1	female	1	11380
6	35	S	8.0500	AllenMr.WilliamHenry	0	5	3	male	0	37345

As a first step in the modelling process, it is often very useful to look at summary statistics to get a sense of the data. To do so, we will create a Procedure (../../../../doc/#builtin/procedures/Procedures.md.html) of type summary.statistics (../../../../doc/#builtin/procedures/SummaryStatisticsProcedure.md.html) and store the results in a new dataset called titanic_summary_stats:

In [4]:
```
print mldb.post("/v1/procedures", {
    "type": "summary.statistics",
    "params": {
        "inputData": "SELECT * FROM titanic_raw",
        "outputDataset": "titanic_summary_stats",
        "runOnCreation": True
    }
})
```

```
<Response [201]>
```

We can take a look at numerical columns:

In [5]:
```
mldb.query("""
    SELECT * EXCLUDING(value.most_frequent_items*)
    FROM titanic_summary_stats
    WHERE value.data_type='number'
""").transpose()
```

Out[5]:

_rowName	Fare	SibSp	PassengerId	label	Age	Pclass	Parch
value.1st_quartile	7.8958	0	223	0	20	2	0
value.3rd_quartile	31	1	669	1	38	3	0
value.avg	32.2042	0.523008	446	0.383838	29.6991	2.30864	0.381594
value.data_type	number	number	number	number	number	number	number
value.max	512.329	8	891	1	80	3	6
value.median	14.4542	0	446	0	28	3	0
value.min	0	0	1	0	0.42	1	0
value.num_null	0	0	0	0	177	0	0
value.num_unique	248	7	891	2	88	3	7
value.stddev	49.6934	1.10274	257.354	0.486592	14.5265	0.836071	0.806057

Training a classifier

We will create another Procedure (../../../../doc/#builtin/procedures/Procedures.md.html) of type classifier.experiment (../../../../doc/#builtin/procedures/ExperimentProcedure.md.html). The configuration parameter defines a Random Forest algorithm.

In [6]:

```
result = mldb.put('/v1/procedures/titanic_train_scorer', {
    "type": "classifier.experiment",
    "params": {
        "experimentName": "titanic",
        "inputData": """
            select
                {Sex, Age, Fare, Embarked, Parch, SibSp, Pclass} as features,
                label
            from titanic_raw
        """,
        "configuration": {
            "type": "bagging",
            "num_bags": 10,
            "validation_split": 0,
            "weak_learner": {
                "type": "decision_tree",
                "max_depth": 10,
                "random_feature_propn": 0.3
            }
        },
        "kfold": 3,
        "modelFileUrlPattern": "file://models/titanic.cls",
        "keepArtifacts": True,
        "outputAccuracyDataset": True,
        "runOnCreation": True
    }
})

auc = np.mean([x["resultsTest"]["auc"] for x in result.json()["status"]["firstRun"]["status"]["folds"]])
print "\nArea under ROC curve = %0.4f\n" % auc
```

```
Area under ROC curve = 0.8357
```

We automatically get a REST API for predictions

The procedure above created for us a Function (../../../../doc/#builtin/functions/Functions.md.html) of type classifier (../../../../doc/#builtin/functions/ClassifierApply.md.html).

In [7]:
```
@ipywidgets.interact
def score( Age=[0,80],Embarked=["C", "Q", "S"], Fare=[1,100], Parch=[0,8], Pclass=[1,3],
          Sex=["male", "female"], SibSp=[0,8]):
    return mldb.get('/v1/functions/titanic_scorer_0/application', input={"features": locals()})
```

GET http://localhost/v1/functions/titanic_scorer_0/application?
input=%7B%22features%22%3A+%7B%22Fare%22%3A+50%2C+%22Embarked%22%3A+%22C%22%2C+%22Age%22%3A+40%2C+%22P
200 OK


```
{
  "output": {
    "score": 0.4552461504936218
  }
}
```


What's in a score?

Scores aren't probabilities, but they can be used to create binary classifiers by applying a cutoff threshold. MLDB's classifier.experiment procedure outputs a dataset which you can use to figure out where you want to set that threshold.

In [8]:
```
test_results = mldb.query("select * from titanic_results_0 order by score desc")
test_results.head()
```

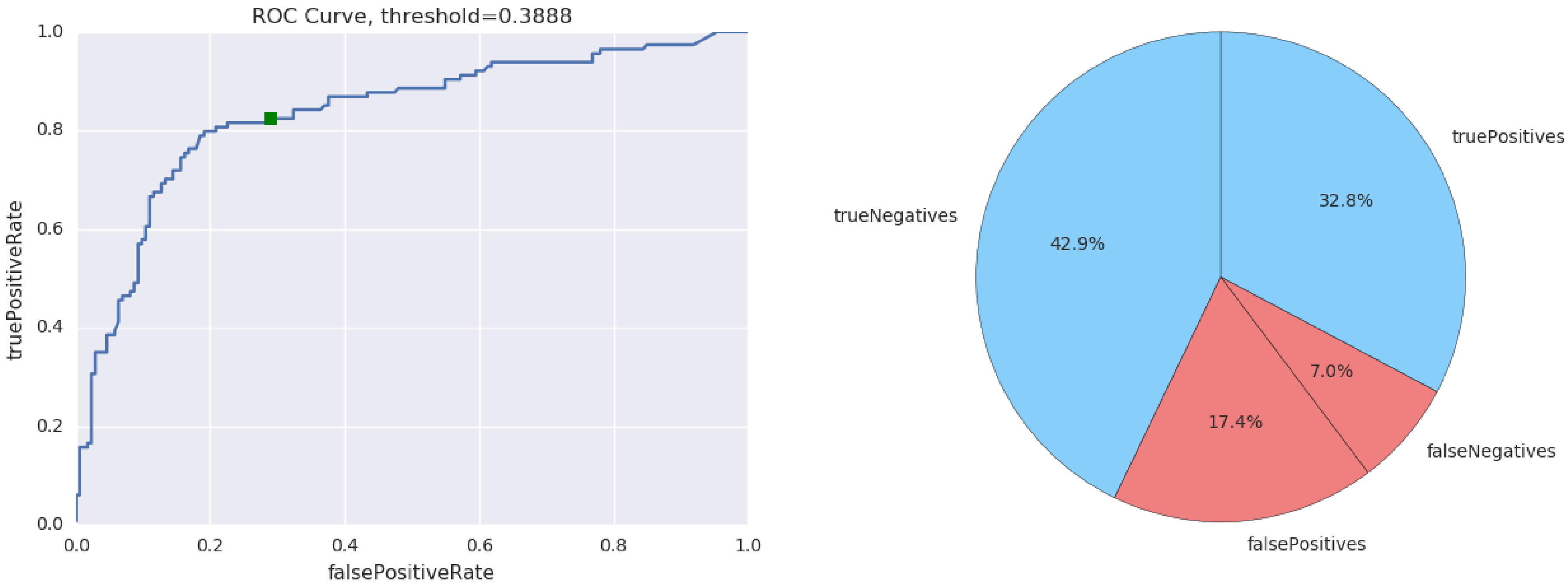
Out[8]:

	accuracy	falseNegatives	falsePositiveRate	falsePositives	index	label	precision	recall	score	trueNegatives	tru
_rowName											
583	0.606272	113	0	0	1	1	1	0.008772	0.817740	173	0.0
63	0.609756	112	0	0	2	1	1	0.017544	0.817460	173	0.0
488	0.613240	111	0	0	3	1	1	0.026316	0.816136	173	0.0
691	0.616725	110	0	0	4	1	1	0.035088	0.804640	173	0.0
220	0.620209	109	0	0	5	1	1	0.043860	0.798763	173	0.0

Here's an interactive way to graphically explore the tradeoffs between the True Positive Rate and the False Positive Rate, using what's called a ROC curve.

NOTE: the interactive part of this demo only works if you're running this Notebook live, not if you're looking at a static copy on http://docs.mldb.ai (http://docs.mldb.ai). See the documentation for Running MLDB (../../../../doc/#builtin/Running.md.html).

In [9]:
```
@ipywidgets.interact
def test_results_plot( threshold_index=[0,len(test_results)-1]):
    row = test_results.iloc[threshold_index]
    cols = ["trueNegatives","falsePositives","falseNegatives","truePositives",]
    f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    test_results.plot(ax=ax1, x="falsePositiveRate", y="truePositiveRate",
    legend=False, title="ROC Curve, threshold=%.4f" % row.score).set_ylabel('truePositiveRate')
    ax1.plot(row.falsePositiveRate, row.truePositiveRate, 'gs')

    ax2.pie(row[cols], labels=cols, autopct='%1.1f%%', startangle = 90,
          colors=['lightskyblue','lightcoral','lightcoral', 'lightskyblue'])
    ax2.axis('equal')
    f.subplots_adjust(hspace=.75)
    plt.show()
```



But what is the model doing under the hood?

Let's create a function of type classifier.explain (../../../../doc/#builtin/functions/ClassifierExplain.md.html) to help us understand what's happening here.

In [10]:
```
mldb.put('/v1/functions/titanic_explainer', {
    "id": "titanic_explainer",
    "type": "classifier.explain",
    "params": { "modelFileUrl": "file://models/titanic.cls" }
})
```

Out[10]: **PUT http://localhost/v1/functions/titanic_explainer**

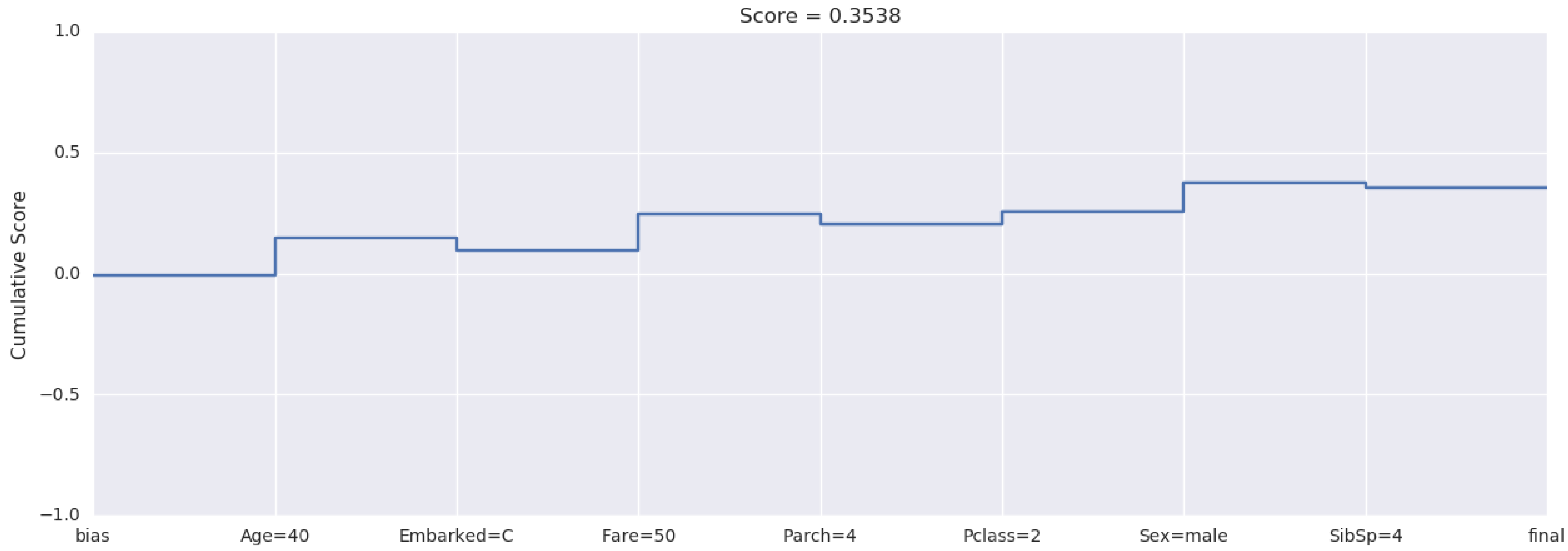
201 Created


```
{
  "status": {
    "mode": "regression",
    "summary": "COMMITTEE"
  },
  "config": {
    "params": {
      "modelFileUrl": "file://models/titanic.cls"
    },
    "type": "classifier.explain",
    "id": "titanic_explainer"
  },
  "state": "ok",
  "type": "classifier.explain",
  "id": "titanic_explainer"
}
```

Exploring the impact of features for a single example

NOTE: the interactive part of this demo only works if you're running this Notebook live, not if you're looking at a static copy on http://docs.mldb.ai (http://docs.mldb.ai). See the documentation for Running MLDB (../../../../doc/#builtin/Running.md.html).

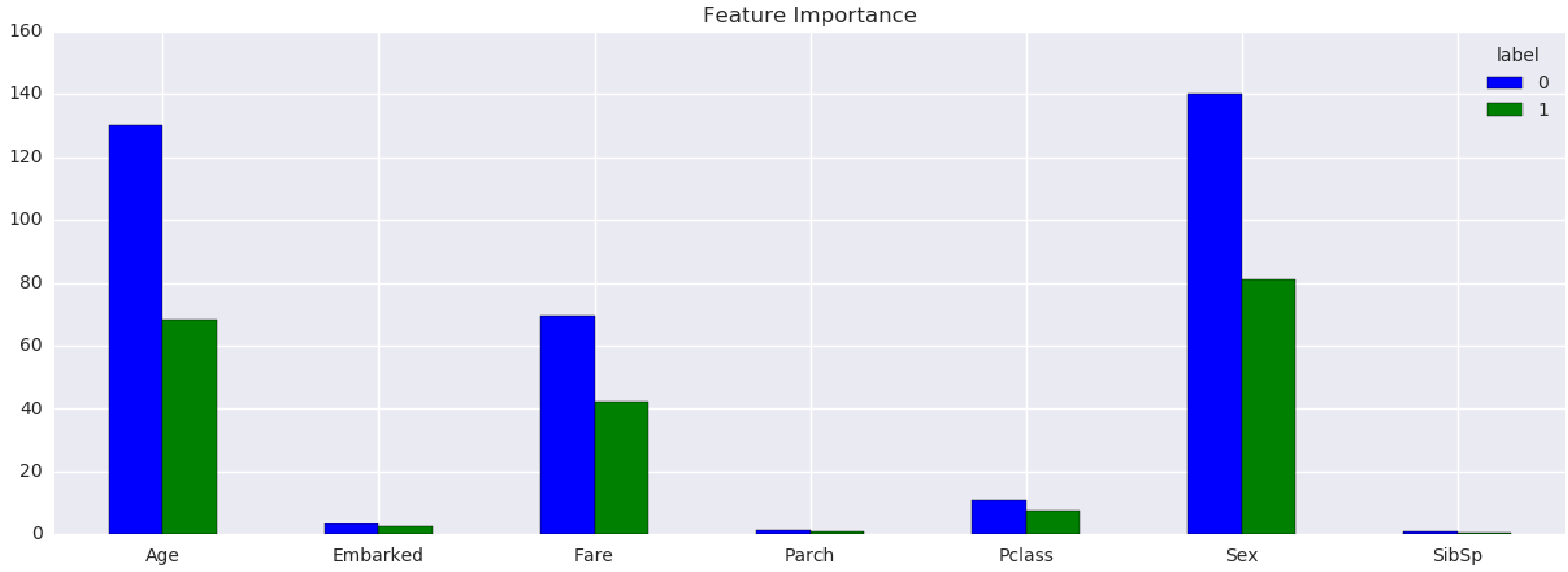
In [11]:
```
@ipywidgets.interact
def sliders( Age=[0,80],Embarked=["C", "Q", "S"], Fare=[1,100], Parch=[0,8], Pclass=[1,3],
            Sex=["male", "female"], SibSp=[0,8]):
    features = locals()
    x = mldb.get('/v1/functions/titanic_explainer/application', input={"features": features, "label": 1}).json()["output"]

    df = pd.DataFrame(
        {"%s=%s" % (feat, str(features[feat])): val for (feat, (val, ts)) in x["explanation"]},
        index=["val"]).transpose().cumsum()
    pd.DataFrame(
        {"cumulative score": [x["bias"]]+list(df.val)+[df.val[-1]]},
        index=['bias'] + list(df.index) + ['final']
    ).plot(kind='line', drawstyle='steps-post', legend=False, figsize=(15, 5),
          ylim=(-1, 1), title="Score = %.4f" % df.val[-1]).set_ylabel('Cumulative Score')

    plt.show()
```



Summing up explanation values to get overall feature importance

When we sum up the explanation values in the context of the correct label, we can get an indication of how important each feature was to making a correct classification.

In [12]:
```
df = mldb.query("""
select label, sum(
    titanic_explainer({
        label: label,
        features: {Sex, Age, Fare, Embarked, Parch, SibSp, Pclass}
    })[explanation]
) as *
from titanic_raw group by label
""")
df.set_index("label").transpose().plot(kind='bar', title="Feature Importance", figsize=(15, 5))
plt.xticks(rotation=0)
plt.show()
```



We can also load up a custom UI for this

In [13]:
```
mldb.put('/v1/plugins/pytanic', {
    "type":"python",
    "params": {"address": "git://github.com/datacratic/mldb-pytanic-plugin"}
})
```

Out[13]:

PUT http://localhost/v1/plugins/pytanic

201 Created


```
{
 "config": {
  "params": {
   "address": "git://github.com/datacratic/mldb-pytanic-plugin"
  },
  "type": "python",
  "id": "pytanic"
 },
 "state": "ok",
 "type": "python",
 "id": "pytanic"
}
```


Now you can browse to the plugin UI (../../../../v1/plugins/pytanic/routes/static/titanic.html).

NOTE: this only works if you're running this Notebook live, not if you're looking at a static copy on http://docs.mldb.ai (http://docs.mldb.ai). See the documentation for Running MLDB (../../../../doc/#builtin/Running.md.html).

Where to next?

Check out the other Tutorials and Demos (../../../../doc/#builtin/Demos.md.html).