Welcome to the Jungle

@__mharrison__

PyCon Colombia 2018

METASNAKE

About Me

- Corporate Trainer
- Author (*Illustrated Guide to Python 3, Guide to Learning the Pandas Library*)
- Python since 2000


OPEN

Outline

- Machine Learning in a Nutshell
- Which algorithm to use?
- The Titanic
- Decision Trees
- Random Forests
- Conclusion

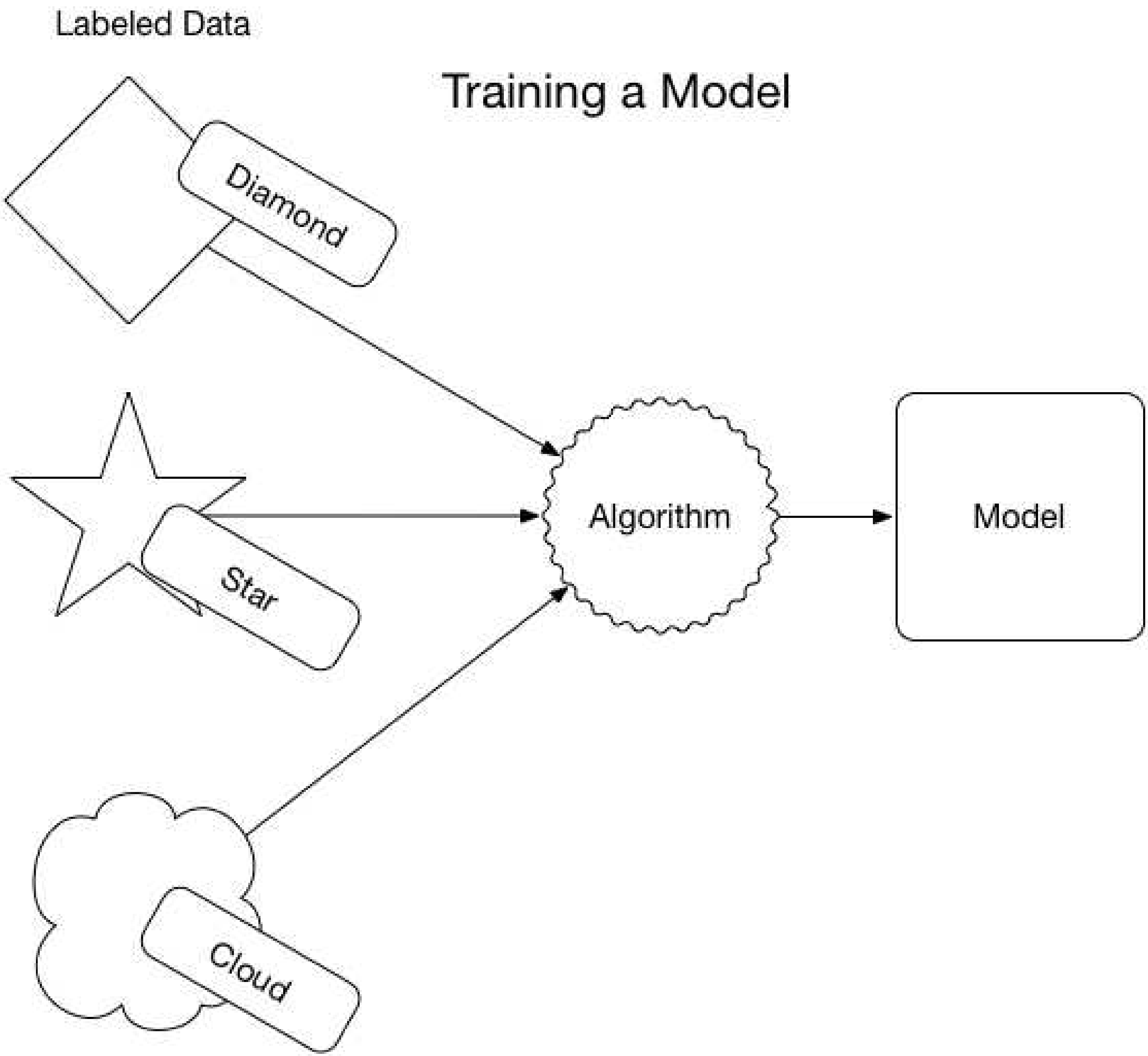
Machine Learning

Supervised Classification

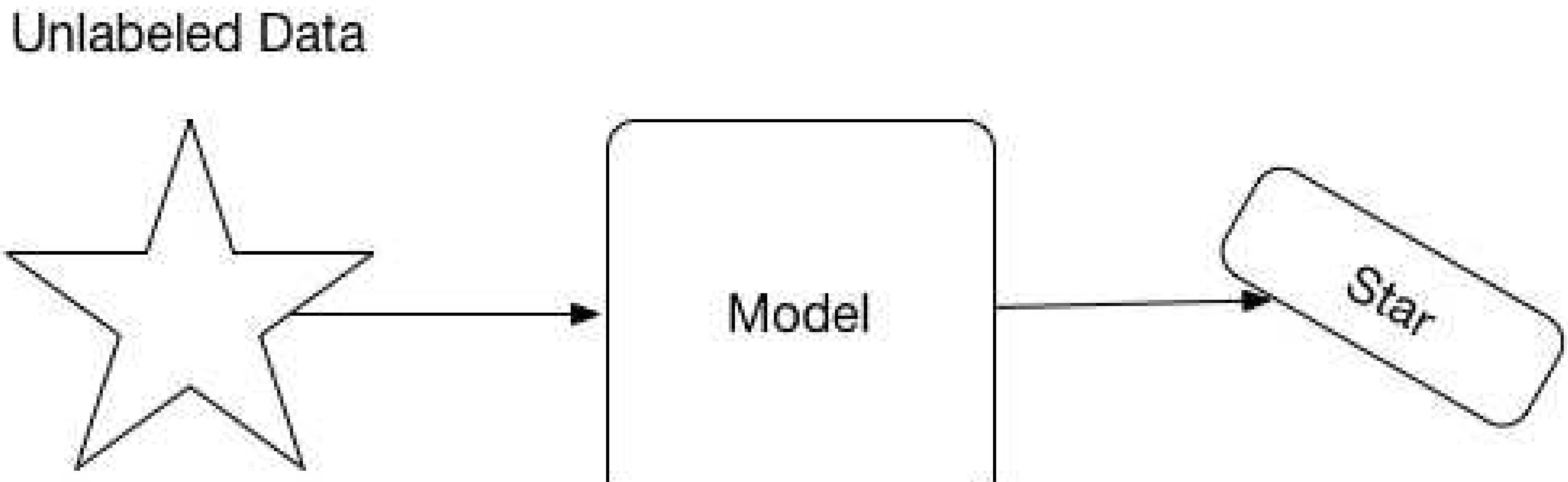
Sometimes called "Predictive Modeling."

1. Identify patterns from labeled examples. (Training Set => Model)
2. Based on those patterns, try to guess labels for other examples. (Predict)

Training a Model



Predicting a Label



METASNAKE

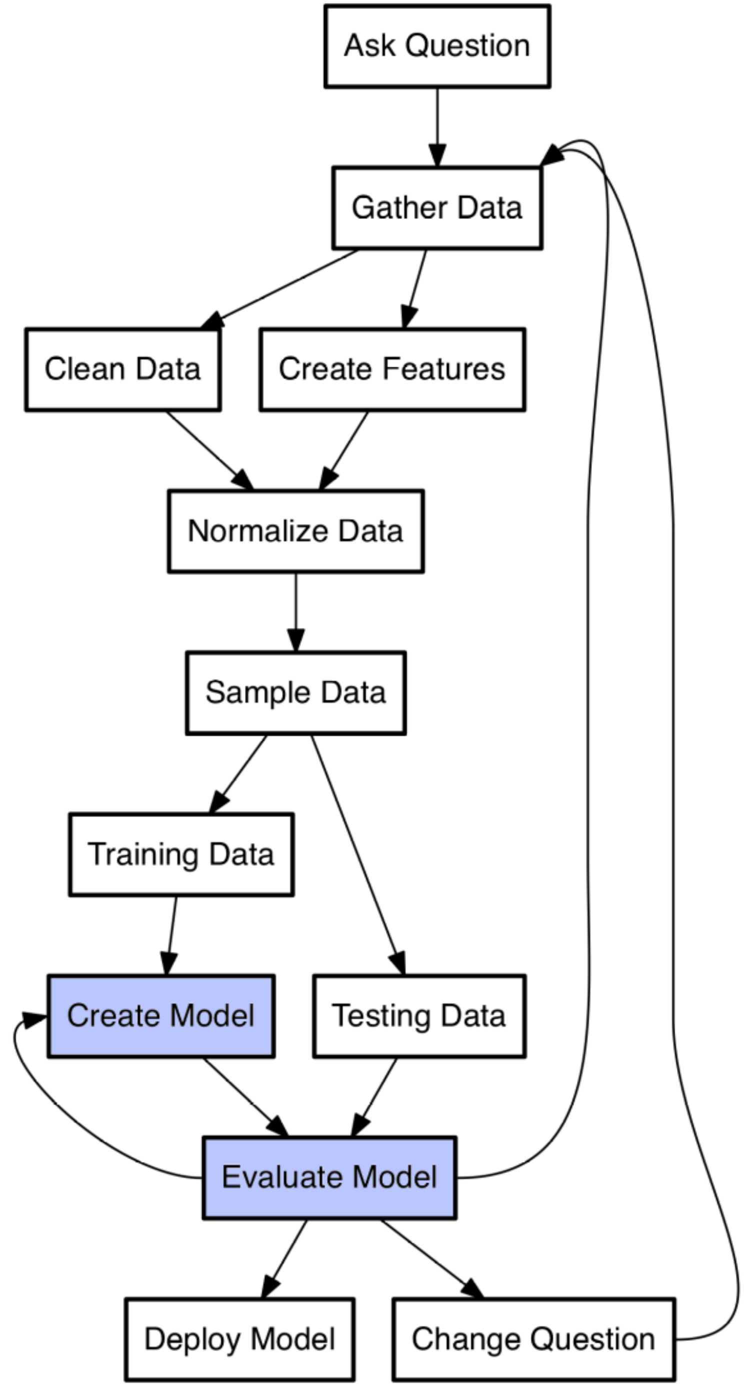
Examples

Binary Classification

- Is this student at high risk for dropping out?
- Is this individual at high risk for defaulting on a loan?
- Is this person at high risk for becoming infected with a certain disease?

Multi-class Classification

- What is the most likely disease given this individual's current symptoms?
- Which ad is the user most likely to click on?



Machine Learning Process

METASNAKE

So which algorithm should I use to build a model?

No Free Lunch

If an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems.

David Wolpert and William Macready. No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation, 1:67, 1997.

No Free Lunch

No single algorithm will build the best model on every dataset.

Which algorithm?

- What is your background discipline? (statistics, computer science, AI)
- Which classifiers are you aware of?
- Is there an implementation of this classifier in your project language?
- Do you understand the parameters well enough to tune them?
- Does it work with your data?

Premise

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research*, 15(Oct):3133–3181, 2014.

http://jmlr.csail.mit.edu/papers/v15/delgado14a.html

Premise

It turns out that Random Forests are usually a good place to start.

Premise

Thunderdome with:

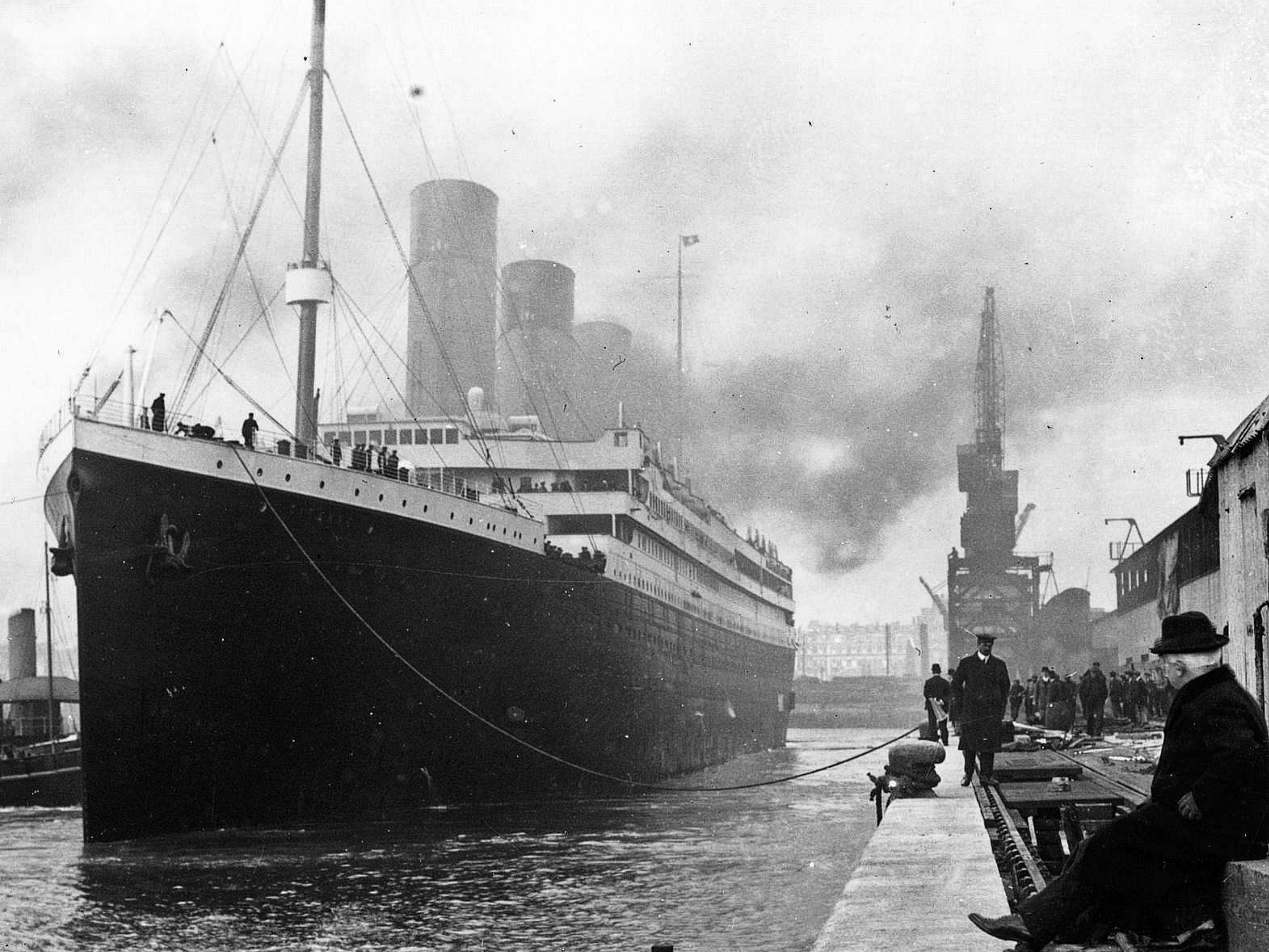
- 179 classifiers from 17 classifier families
- 121 datasets from the UCI repository

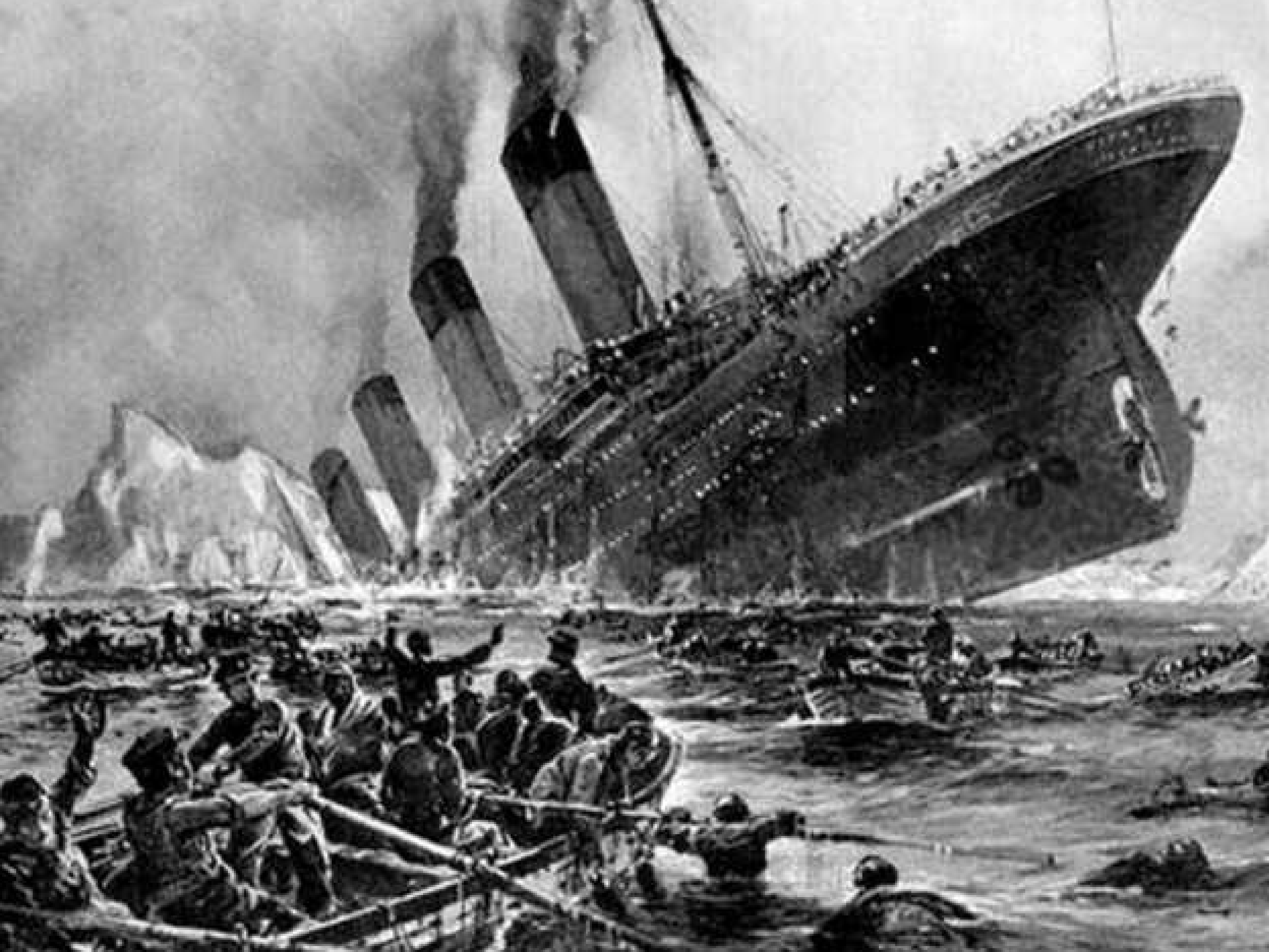
Premise

"The Random Forest is clearly the best family of classifiers, followed by SVM, neural networks, and boosting ensembles."

- On average, RF achieved 94.1% of the theoretical maximum accuracy for each dataset.
- RF achieved over 90% of maximum accuracy in 84.3% of datasets.

The RMS Titanic





Classification Task

Predict who did and did not survive the disaster on the Titanic

Classification Task

Predict who did and did not survive the disaster on the Titanic

(and see if we can get some idea of *why* it turned out that way)

Get Data

```
>>> import pandas as pd
>>> df = pd.read_excel('data/titanic3.xls')
```

http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls

Columns

- class: Passenger class (1 = first; 2 = second; 3 = third)
- name: Name
- sex: Sex
- age: Age
- sibsp: Number of siblings/spouses aboard
- parch: Number of parents/children aboard
- ticket: Ticket number
- fare: Passenger fare
- cabin: Cabin
- embarked: Port of embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
- boat: Lifeboat (if survived)
- body: Body number (if did not survive and body was recovered)

Label Column

The column giving the label for our classification task:

- survival: Survival (0 = no; 1 = yes)

Exploring

```
>>> df.shape
(1309, 14)
>>> df.embarked.value_counts()
S    914
C    270
Q    123
Name: embarked, dtype: int64
```

Exploring

```
>>> df.cabin.value_counts()
C23 C25 C27        6
B57 B59 B63 B66    5
G6                 5
B96 B98            4
F2                 4
F4                 4
C22 C26            4
F33                4
D                  4
C78                4
E101               3
B58 B60            3
...
Name: cabin, dtype: int64
```

Question

Can we build a model that will predict survival?

Assignment

Load Data

Decision Trees

Decision Trees

```
>>> from sklearn import tree
>>> model = tree.DecisionTreeClassifier(random_state=42)
>>> ignore =
set('boat,body,home.dest,name,ticket'.split(','))
>>> cols = [c for c in df.columns if c != 'survived' and c
not in ignore]
>>> X = df[cols]
>>> y = df.survived
>>> model.fit(X, y)
Traceback (most recent call last):
  ...
ValueError: could not convert string to float: 'S'
```

Create Dummy Variables

```
>>> dummy_cols = 'pclass,sex,cabin,embarked'.split(",")
>>> df2 = pd.get_dummies(df, columns=dummy_cols)

>>> model = tree.DecisionTreeClassifier(random_state=42)
>>> ignore =
set('boat,body,home.dest,name,ticket'.split(','))
>>> cols = [c for c in df2.columns if c != 'survived' and c
...     not in ignore and c not in dummy_cols]
>>> X = df2[cols]
>>> y = df2.survived
```

Try Again

```
>>> model.fit(X, y)
Traceback (most recent call last):
  ...
ValueError: Input contains NaN, infinity
or a value too large for
dtype('float32').
```

Imputing

Fancy term for filling in values. Mean is a good choice for decision trees as it doesn't bias the splitting, whereas 0 would

Try Again

```
>>> X = X.fillna(X.mean())
>>> X.dtypes
age                 float64
sibsp                 int64
parch                 int64
fare                float64
pclass_1            float64
pclass_2            float64
pclass_3            float64
sex_female          float64
sex_male            float64
cabin_A10           float64
cabin_A11           float64
cabin_A14           float64
```

METASNAKE

Assignment

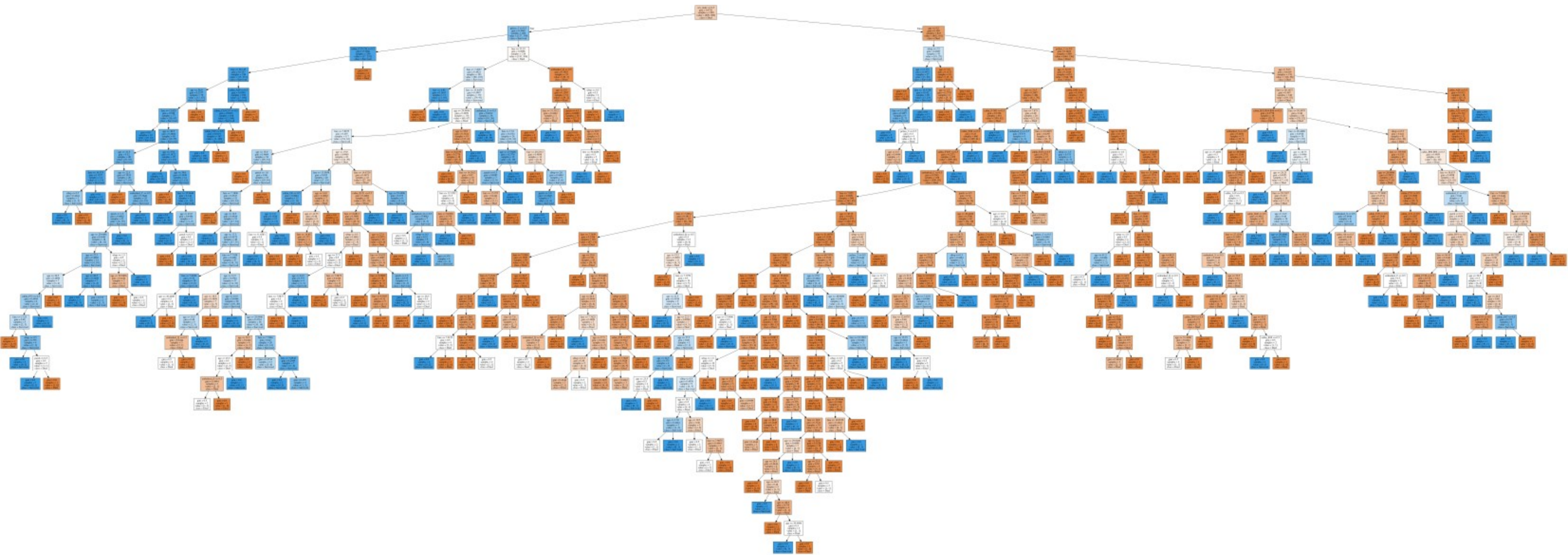
Convert to numeric

Try Again

```
>>> model.fit(X, y)
DecisionTreeClassifier(class_weight=None,
    criterion='gini', max_depth=None,
    max_features=None, max_leaf_nodes=None,
    min_samples_leaf=1, min_samples_split=2,
    min_weight_fraction_leaf=0.0,
    presort=False, random_state=42,
    splitter='best')
```

What Does the Tree Look Like?

```
>>> tree.export_graphviz(model,
...     out_file='/tmp/tree1.dot',
...   feature_names=X.columns,
...   class_names=['Died', 'Survived'],
...   filled=True)

>>> import subprocess
>>> _ = subprocess.check_output(
...     'dot -Tpng -oimg/tree1.png /tmp/tree1.dot'.split())
```



METASNAKE

Assignment

Decision Tree

Does it Generalize?

Need a Test Set

```
>>> from sklearn import model_selection
>>> X_train, X_test, y_train, y_test = \
...     model_selection.train_test_split(
...     X, y, test_size=.3, random_state=42)

>>> _ = model.fit(X_train, y_train)
>>> model.score(X_test, y_test)
0.7684478371501272
```

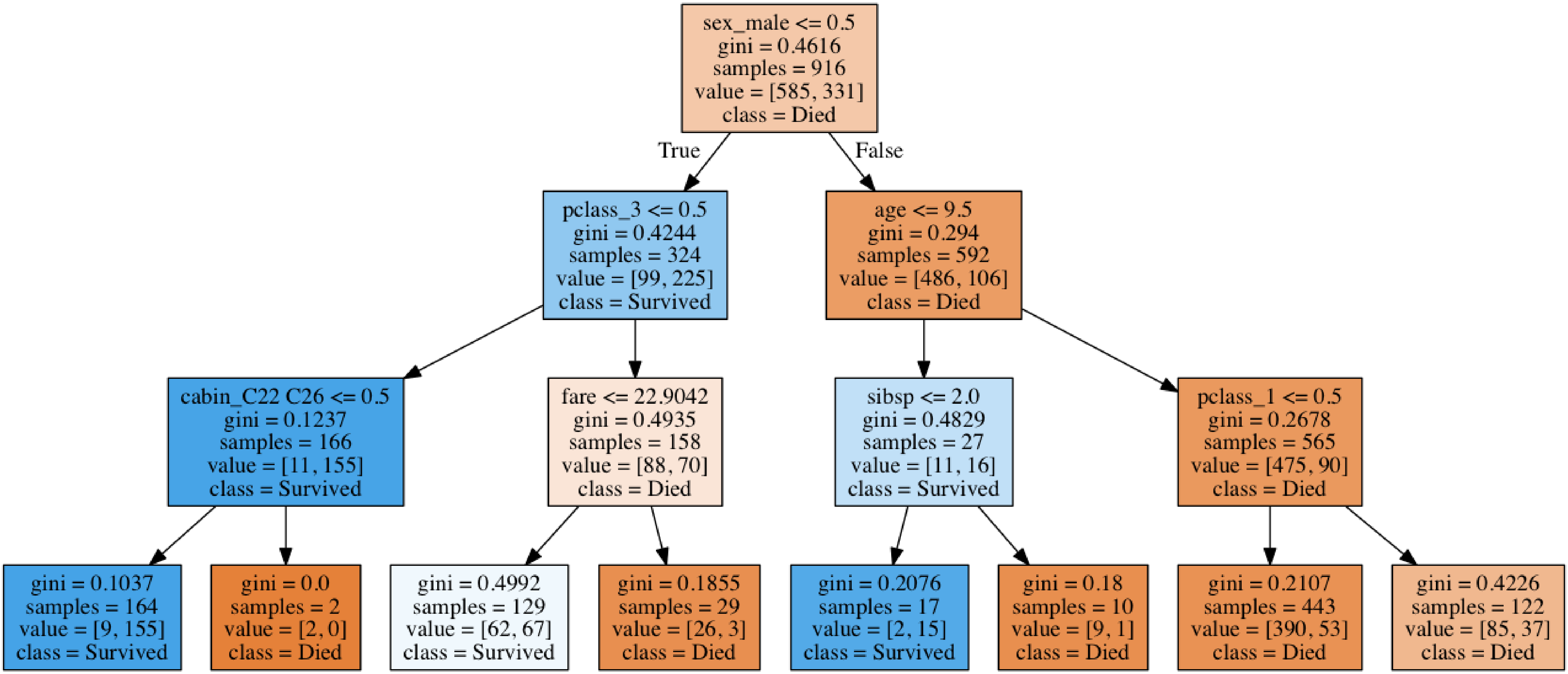
METASNAKE

Another Model

```
>>> model2 = tree.DecisionTreeClassifier(
...     random_state=42, max_depth=3)
>>> _ = model2.fit(X_train, y_train)
>>> model2.score(X_test, y_test)
0.8142493638676844
```

What Does the Tree Look Like?

```
>>> tree.export_graphviz(model2,
...     out_file='/tmp/tree2.dot',
...     feature_names=X.columns,
...     class_names=['Died', 'Survived'],
...     filled=True)

>>> import subprocess
>>> _ = subprocess.check_output(
...      'dot -Tpng -oimg/tree2.png /tmp/tree2.dot'.split())
```


sex_male <= 0.5
gini = 0.4616
samples = 916
value = [585, 331]
class = Died
True
False
pclass_3 <= 0.5
gini = 0.4244
samples = 324
value = [99, 225]
class = Survived
age <= 9.5
gini = 0.294
samples = 592
value = [486, 106]
class = Died
cabin_C22 C26 <= 0.5
gini = 0.1237
samples = 166
value = [11, 155]
class = Survived
fare <= 22.9042
gini = 0.4935
samples = 158
value = [88, 70]
class = Died
sibsp <= 2.0
gini = 0.4829
samples = 27
value = [11, 16]
class = Survived
pclass_1 <= 0.5
gini = 0.2678
samples = 565
value = [475, 90]
class = Died
gini = 0.1037
samples = 164
value = [9, 155]
class = Survived
gini = 0.0
samples = 2
value = [2, 0]
class = Died
gini = 0.4992
samples = 129
value = [62, 67]
class = Survived
gini = 0.1855
samples = 29
value = [26, 3]
class = Died
gini = 0.2076
samples = 17
value = [2, 15]
class = Survived
gini = 0.18
samples = 10
value = [9, 1]
class = Died
gini = 0.2107
samples = 443
value = [390, 53]
class = Died
gini = 0.4226
samples = 122
value = [85, 37]
class = Died

METASNAKE

Assignment

Training / Testing & Generalization

Another Performance Measure

ROC Curve

Receiver Operating Characteristic - area indicates performance

ROC

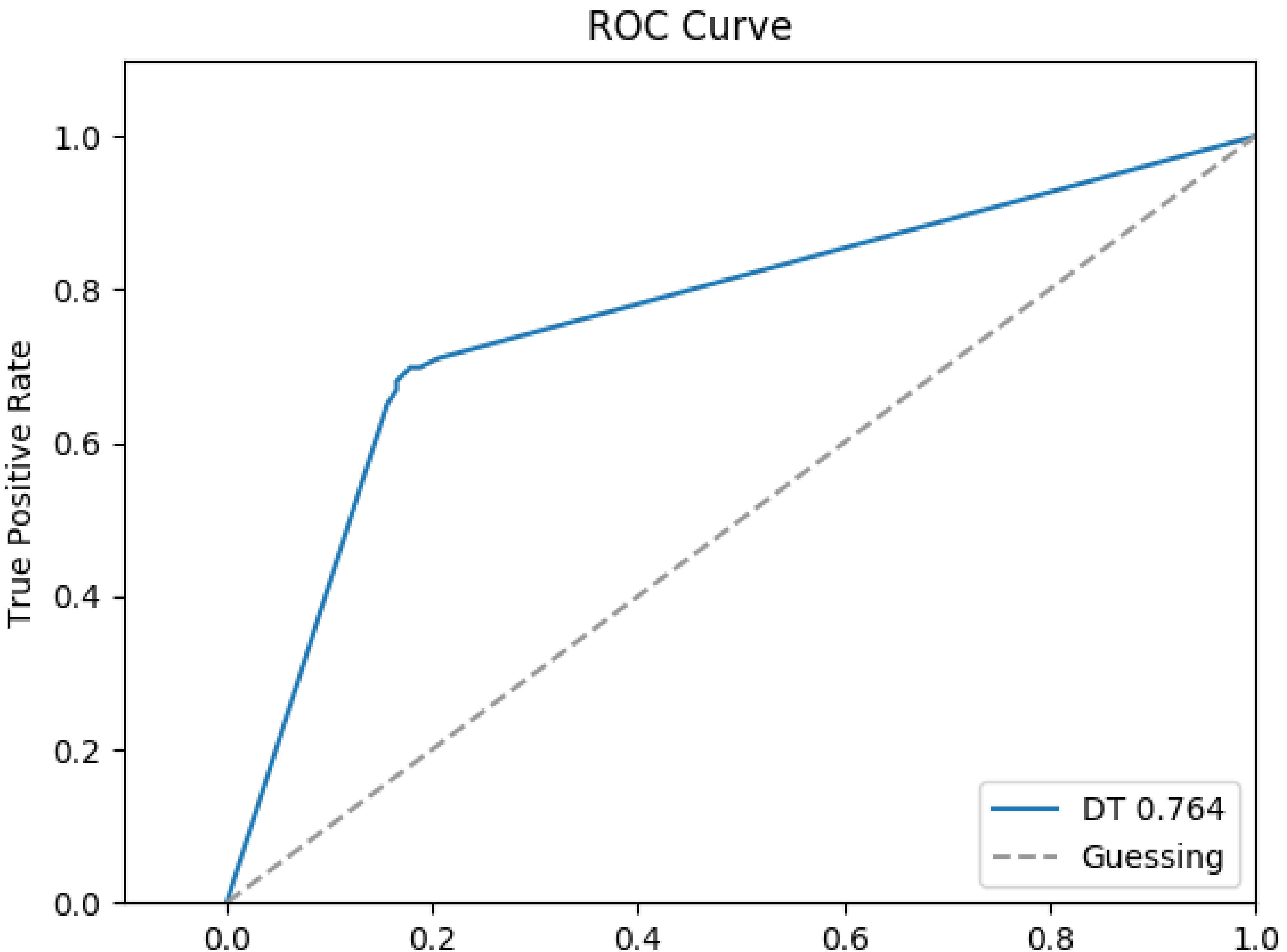
```
>>> from sklearn.metrics import auc, confusion_matrix, roc_curve

>>> def fig_with_title(ax, title, figkwargs):
...     if figkwargs is None:
...         figkwargs = {}
...     if not ax:
...         fig = plt.figure(**figkwargs)
...         ax = plt.subplot(111)
...     else:
...         fig = plt.gcf()
...     if title:
...         ax.set_title(title)
...     return fig, ax
```

ROC

```
>>> def plot_roc_curve_binary(clf, X, y, label='ROC Curve (area={area:.3})',
...                           title="ROC Curve", pos_label=None, sample_weight=None,
...                           ax=None, figkwargs=None, plot_guess=False):
...     ax = ax or plt.subplot(111)
...     ax.set_xlim([-.1, 1])
...     ax.set_ylim([0, 1.1])
...     y_score = clf.predict_proba(X)
...     if y_score.shape[1] != 2 and not pos_label:
...         warnings.warn("Shape is not binary {} and no pos_label".format(y_score.shape))
...         return
...     try:
...         fpr, tpr, thresholds = roc_curve(y, y_score[:,1], pos_label=pos_label,
...                                          sample_weight=sample_weight)
...     except ValueError as e:
...         if 'is not binary' in str(e):
...             warnings.warn("Check if y is numeric")
...             raise
...
...     roc_auc = auc(fpr, tpr)
...     fig, ax = fig_with_title(ax, title, figkwargs)
...
...     ax.plot(fpr, tpr, label=label.format(area=roc_auc))
...     if plot_guess:
...         ax.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Guessing')
...     ax.set_xlabel('False Positive Rate')
...     ax.set_ylabel('True Positive Rate')
...     ax.legend(loc="lower right")
...     return fig, ax
```

ROC

```
>>> from matplotlib import pyplot as plt
>>> plt.clf()
>>> fig, ax = plot_roc_curve_binary(
...     model, X_test, y_test,
...     'DT {area:.3}', plot_guess=1)
>>> fig.savefig('img/ml-roc.png')
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
DT 0.764
Guessing

Assignment

ROC Curve

Pros/Cons Decision Trees

Pros:

- Easy to explain

Cons:

- Tends to overfit

Random Forest

Random Forest

Created by Tin Kam Ho (1995), Leo Breiman, and Adele Cutler (2001).

Condorcet's Jury Theorem

From 1785 *Essay on the Application of Analysis to the Probabililty of Majority Decisions*. If each member of jury has p > .5 of predicting correct choice, adding more jury members increases probability of correct choice.

Random Forest

Algorithm:

- Sample from training set N (random WITH REPLACEMENT - lets us do OOB)
- Select m input variables (subset of M total input variables)
- Grow a tree
- Repeat above (create *ensemble*)
- Predict by aggregation predictions of forest (votes for classification, average for regression)

METASNAKE

Random Forest

```
>>> from sklearn import ensemble
>>> model3 = ensemble.RandomForestClassifier(random_state=42)
>>> model3.fit(X_train, y_train)
 RandomForestClassifier(bootstrap=True, class_weight=None,
     criterion='gini', max_depth=None, max_features='auto',
     max_leaf_nodes=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=None, verbose=0,
     warm_start=False)

>>> model3.score(X_test, y_test)
0.7557251908396946
```

METASNAKE

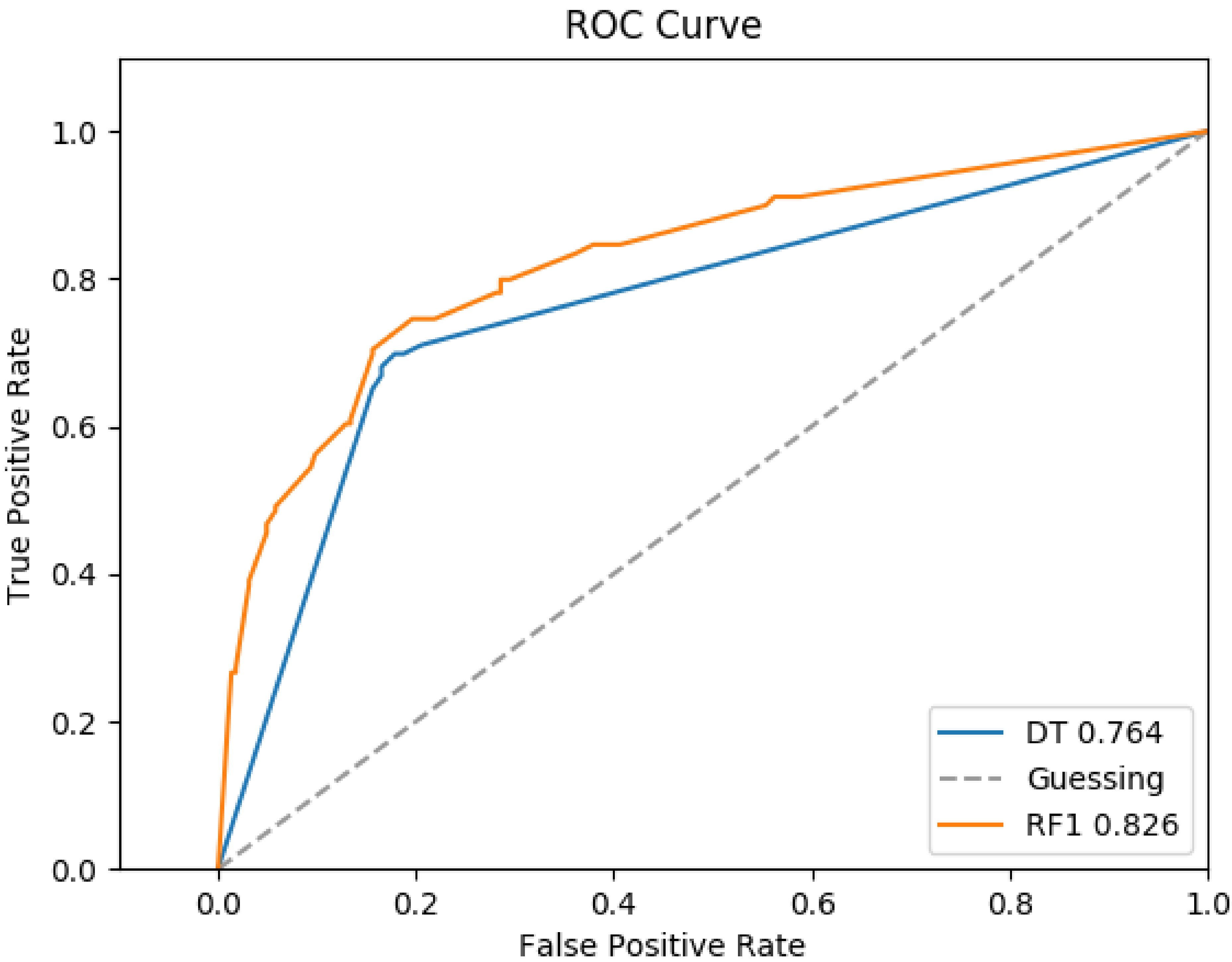
Feature Importance

Most important features at the top of the decision trees

```
>>> print(sorted(zip(X.columns, model3.feature_importances_),
...     key=lambda x: x[1], reverse=True))
 [('age', 0.22344483424840464), ('fare', 0.19018725802080991),
('sex_male', 0.12990057398621174), ('sex_female',
0.12860349870512569), ('pclass_3', 0.051127382589271984),
('parch', 0.042403381656923547), ('sibsp',
0.041437135835858306), ('pclass_1', 0.026146920495887703),
('embarked_S', 0.016952460872998475), ('pclass_2',
0.014536895778953276), ('embarked_C', 0.011974575978148253),
('embarked_Q', 0.0066746190486480592), ('cabin_D56',
0.0050674850086476347), ('cabin_C22 C26',
0.0038209715167321157), ('cabin_F E57',
```

ROC

```
>>> fig, ax = plot_roc_curve_binary(
...     model3, X_test, y_test,
...     'RF1 {area:.3}')
>>> fig.savefig('img/ml-roc3.png')
```



ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
DT 0.764
Guessing
RF1 0.826

Assignment

Random Forest

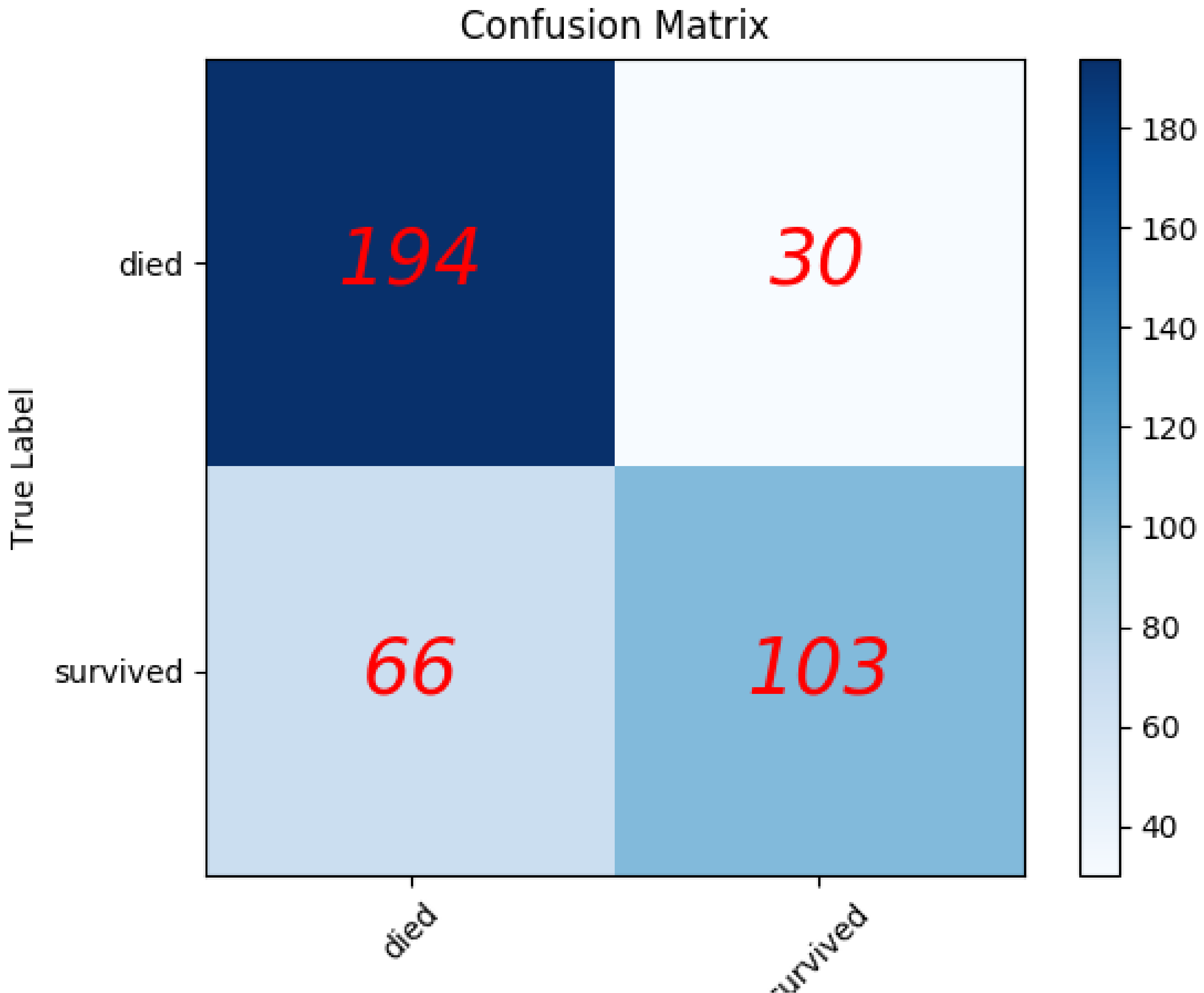
Confusion Matrix

Confusion Matrix

```
>>> def plot_confusion_matrix(clf, X, y, labels, random_state=42, annotate=True,
...                           cmap=plt.cm.Blues,
...                           title="Confusion Matrix", ax=None, figkwargs=None):
...     fig, ax = fig_with_title(ax, title, figkwargs)
...     #X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=random_state)
...     y_pred = clf.predict(X)
...     cm = confusion_matrix(y, y_pred)
...     im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
...     fig.colorbar(im)
...     ax.set_xticks(range(len(labels)))
...     ax.set_xticklabels(labels, rotation=45)
...     ax.set_yticks(range(len(labels)))
...     ax.set_yticklabels(labels)
...     ax.set_ylabel('True Label')
...     ax.set_xlabel('Predicted Label')
...     if annotate:
...         for x in range(len(labels)):
...             for y in range(len(labels)):
...                 plt.annotate(str(cm[x][y]),
...                              xy=(y,x),
...                              ha='center',va='center',color='red', fontsize=25, fontstyle='oblique')
...
...     return fig, ax
```

Confusion Matrix

```
>>> plt.clf()
>>> fig, ax = plot_confusion_matrix(
...     model3, X_test, y_test,
...     ['died', 'survived'])
>>> fig.savefig('img/ml-conf.png')
```


Confusion Matrix
True Label
died
survived
194
30
66
103
died
survived
180
160
140
120
100
80
60
40

Assignment

Plot a Confusion Matrix

Calibration Curves

Calibration Curves

Another mechanism to see how a classifier behaves

- https://jmetzen.github.io/2015-04-14/calibration.html

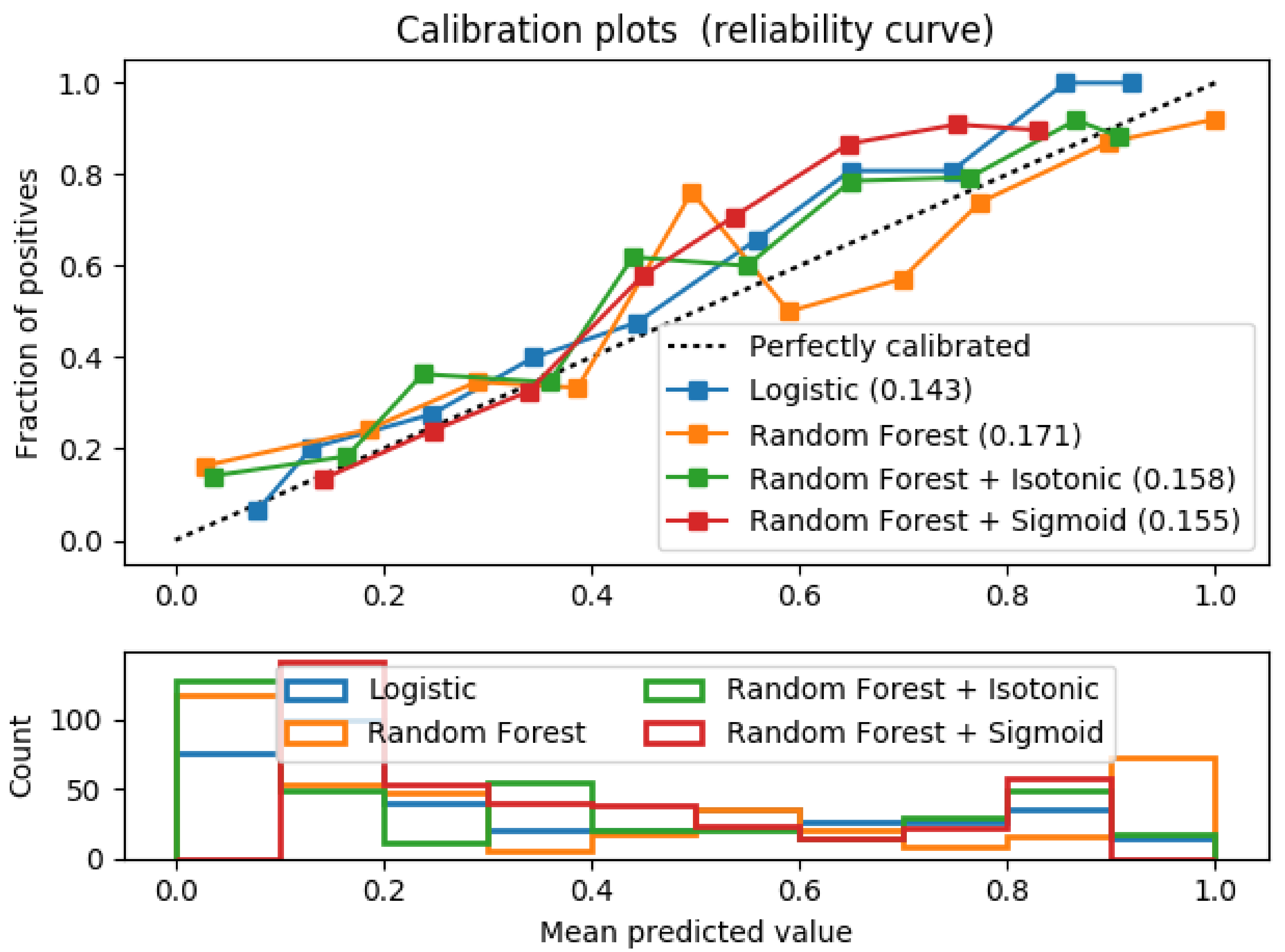
Calibration Curves

```
>>> from sklearn.calibration import CalibratedClassifierCV, calibration_curve
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
...                              f1_score)

>>> def plot_calibration_curve(est, name, fig_index,
...     X_train, X_test, y_train, y_test):
...     """Plot calibration curve for est w/o and with calibration. """
...     # Calibrated with isotonic calibration
...     isotonic = CalibratedClassifierCV(est, cv=2, method='isotonic')
...
...     # Calibrated with sigmoid calibration
...     sigmoid = CalibratedClassifierCV(est, cv=2, method='sigmoid')
...
...     # Logistic regression with no calibration as baseline
...     lr = LogisticRegression(C=1., solver='lbfgs')
...
...     fig = plt.figure(fig_index, figsize=(10, 10))
...     ax1 = plt.subplot2grid((3, 1), (0, 0), rowspan=2)
...     ax2 = plt.subplot2grid((3, 1), (2, 0))
...
...     ax1.plot([0, 1], [0, 1], "k:", label="Perfectly calibrated")
...     for clf, name in [(lr, 'Logistic'),
...                       (est, name),
...                       (isotonic, name + ' + Isotonic'),
...                       (sigmoid, name + ' + Sigmoid')]:
...         clf.fit(X_train, y_train)
...         y_pred = clf.predict(X_test)
...         if hasattr(clf, "predict_proba"):
...             prob_pos = clf.predict_proba(X_test)[:, 1]
...         else:  # use decision function
...             prob_pos = clf.decision_function(X_test)
...             prob_pos = \
...                 (prob_pos - prob_pos.min()) / (prob_pos.max() - prob_pos.min())
...
...         clf_score = brier_score_loss(y_test, prob_pos, pos_label=y.max())
...         print("%s:" % name)
...         print("\tBrier: %1.3f" % (clf_score))
...         print("\tPrecision: %1.3f" % precision_score(y_test, y_pred))
...         print("\tRecall: %1.3f" % recall_score(y_test, y_pred))
...         print("\tF1: %1.3f" % f1_score(y_test, y_pred))
...         print("\tScore: %1.3f\n" % clf.score(X_test, y_test))
...
...         fraction_of_positives, mean_predicted_value = \
...             calibration_curve(y_test, prob_pos, n_bins=10)
...
...         ax1.plot(mean_predicted_value, fraction_of_positives, "s-",
...                  label="%s (%1.3f)" % (name, clf_score))
...
...         ax2.hist(prob_pos, range=(0, 1), bins=10, label=name,
...                  histtype="step", lw=2)
...
...     ax1.set_ylabel("Fraction of positives")
...     ax1.set_ylim([-0.05, 1.05])
...     ax1.legend(loc="lower right")
...     ax1.set_title('Calibration plots  (reliability curve)')
...
...     ax2.set_xlabel("Mean predicted value")
...     ax2.set_ylabel("Count")
...     ax2.legend(loc="upper center", ncol=2)
...
...     plt.tight_layout()
...     return fig, [ax1, ax2]
```

METASNAKE

Calibration Curves

```
>>> fig, axs = plot_calibration_curve(
...     model3, 'Random Forest', 1,
...     X_train, X_test, y_train, y_test)
>>> fig.savefig('img/ml-cc.png')
```



Calibration plots (reliability curve)
Fraction of positives
1.0
0.8
0.6
0.4
0.2
0.0
Perfectly calibrated
Logistic (0.143)
Random Forest (0.171)
Random Forest + Isotonic (0.158)
Random Forest + Sigmoid (0.155)
0.0
0.2
0.4
0.6
0.8
1.0
Count
100
50
0
Logistic
Random Forest
Random Forest + Isotonic
Random Forest + Sigmoid
0.0
0.2
0.4
0.6
0.8
1.0
Mean predicted value

Tuning

Overfitting & Underfitting

- Overfitting - memorizing data
- Underfitting - not flexible enough (cannot capture trend)

Tuning

Fancy term *regularization* - attempt to prevent *overfitting*

Tuning

- `max_features` - Don't want to use all of the features (all tree will look the same). By taking samples of features, you reduce bias and correlation amoung trees
- `n_estimators` - More is better, but diminishing returns (don't need too many jurors, takes a longer time to train, lots of memory)
- `max_depth` - too tall, overfitting. Can't know ahead of time what good size could be. Can use these parameters to constrain depth as well:
- `min_samples_leaf` - smaller is more prone to overfitting (capturing noise)
- `max_leaf_nodes` - Can't be more than this many leaves
- `min_weight_fraction_leaf` - The minimum weighted fraction of the input samples required to be at a leaf node. Note: this parameter is tree-specific.

Adjusting Parameters

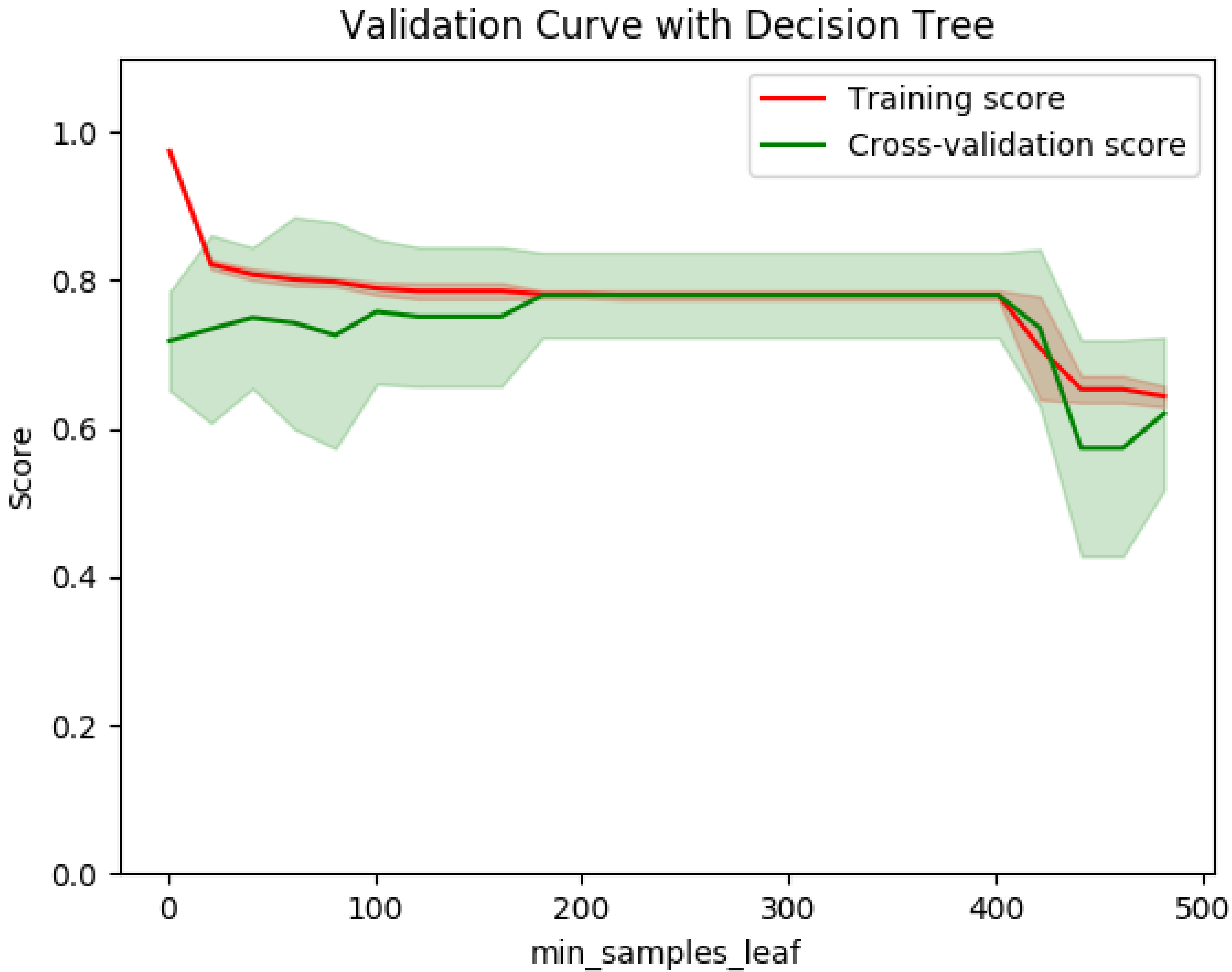
Adjust Parameters

Adjust `min_samples_leaf`:

```
>>> import numpy as np
>>> from sklearn.model_selection import validation_curve
>>> model3 = tree.DecisionTreeClassifier(random_state=42)
>>> param_range = np.arange(1, 500, 20)
>>> param_name = 'min_samples_leaf'
>>> train_scores, test_scores = validation_curve(
...     model3, X, y, param_name=param_name,
param_range=param_range,
...     cv=10, scoring="accuracy", n_jobs=1)
>>> train_scores_mean = np.mean(train_scores, axis=1)
>>> train_scores_std = np.std(train_scores, axis=1)
>>> test_scores_mean = np.mean(test_scores, axis=1)
>>> test_scores_std = np.std(test_scores, axis=1)
```

Plot Validation Curve

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> plt.title("Validation Curve with Decision Tree")
>>> plt.xlabel(param_name)
>>> plt.ylabel("Score")
>>> plt.ylim(0.0, 1.1)
>>> plt.plot(param_range, train_scores_mean, label="Training score", color="r")
>>> plt.fill_between(param_range, train_scores_mean - train_scores_std,
...    train_scores_mean + train_scores_std, alpha=0.2, color="r")
>>> plt.plot(param_range, test_scores_mean, label="Cross-validation score",
...    color="g")
>>> plt.fill_between(param_range, test_scores_mean - test_scores_std,
...   test_scores_mean + test_scores_std, alpha=0.2, color="g")
>>> plt.legend(loc="best")

>>> fig.savefig('img/ml-dt-param-features.png')
>>> #plt.clf()
```


Validation Curve with Decision Tree
Training score
Cross-validation score
Score
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
min_samples_leaf

Grid Search

Grid Search

```
>>> from sklearn.model_selection import GridSearchCV
>>> model5 = ensemble.RandomForestClassifier()
>>> params = {'max_features': [.1, .3, .5, 1],
...           'n_estimators': [10, 20, 50],
...           'min_samples_leaf': [3, 5, 9],
...           'random_state': [42]}
>>> cv = GridSearchCV(model5, params).fit(X, y)
>>> cv.best_params_
{'max_features': 0.1, 'min_samples_leaf': 3,
'n_estimators': 20, 'random_state': 42}
```

METASNAKE

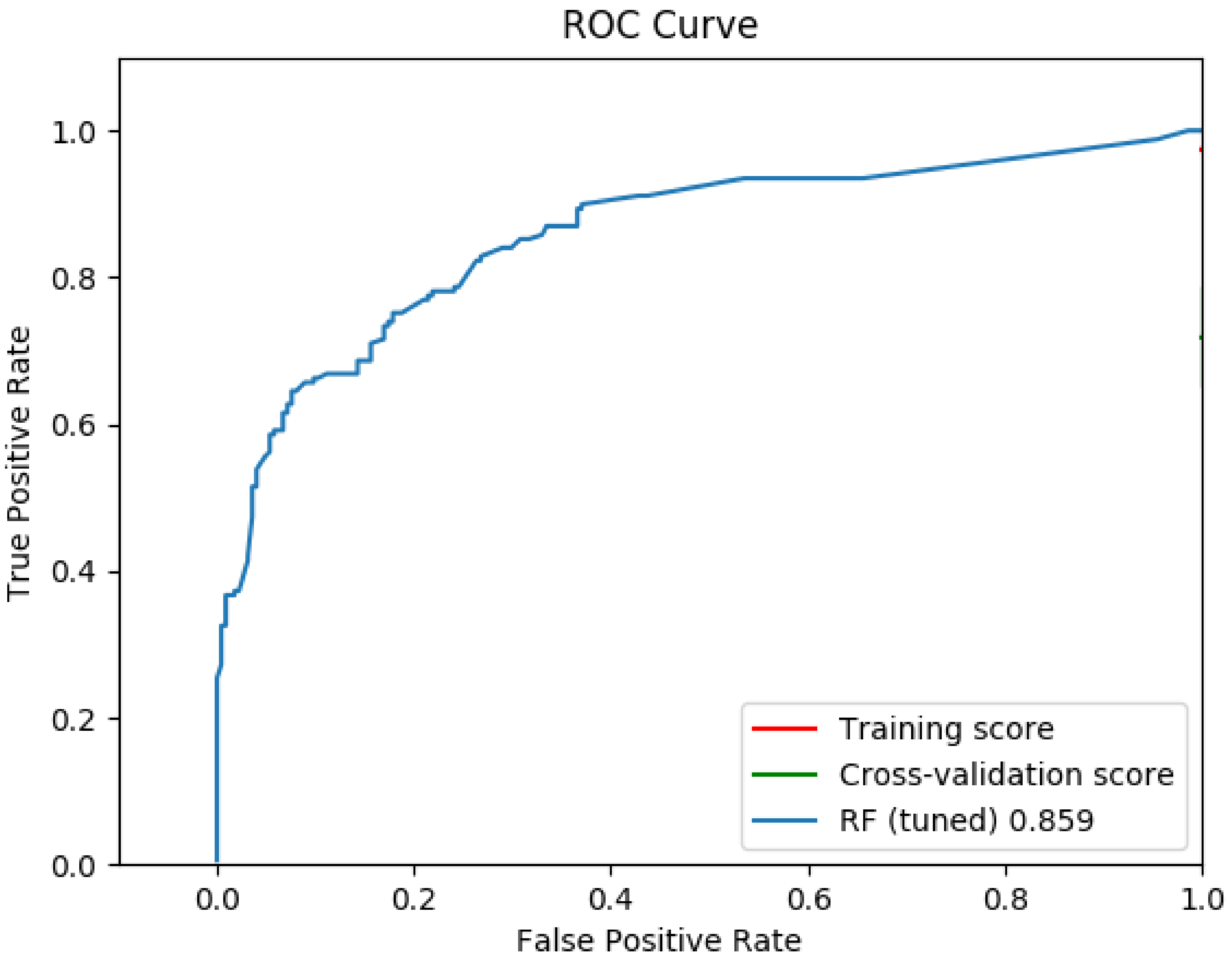
Grid Search

```
>>> model6 = ensemble.RandomForestClassifier(
...     **cv.best_params_)
>>> model6.fit(X_train, y_train)
>>> model6.score(X_test, y_test)
0.7760814249363868
```

METASNAKE

ROC

```
>>> fig, ax = plot_roc_curve_binary(
...     model6, X_test, y_test,
...     'RF (tuned) {area:.3}')
>>> fig.savefig('img/ml-roc6.png')
```


ROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
Training score
Cross-validation score
RF (tuned) 0.859

Assignment

Optimizing Models

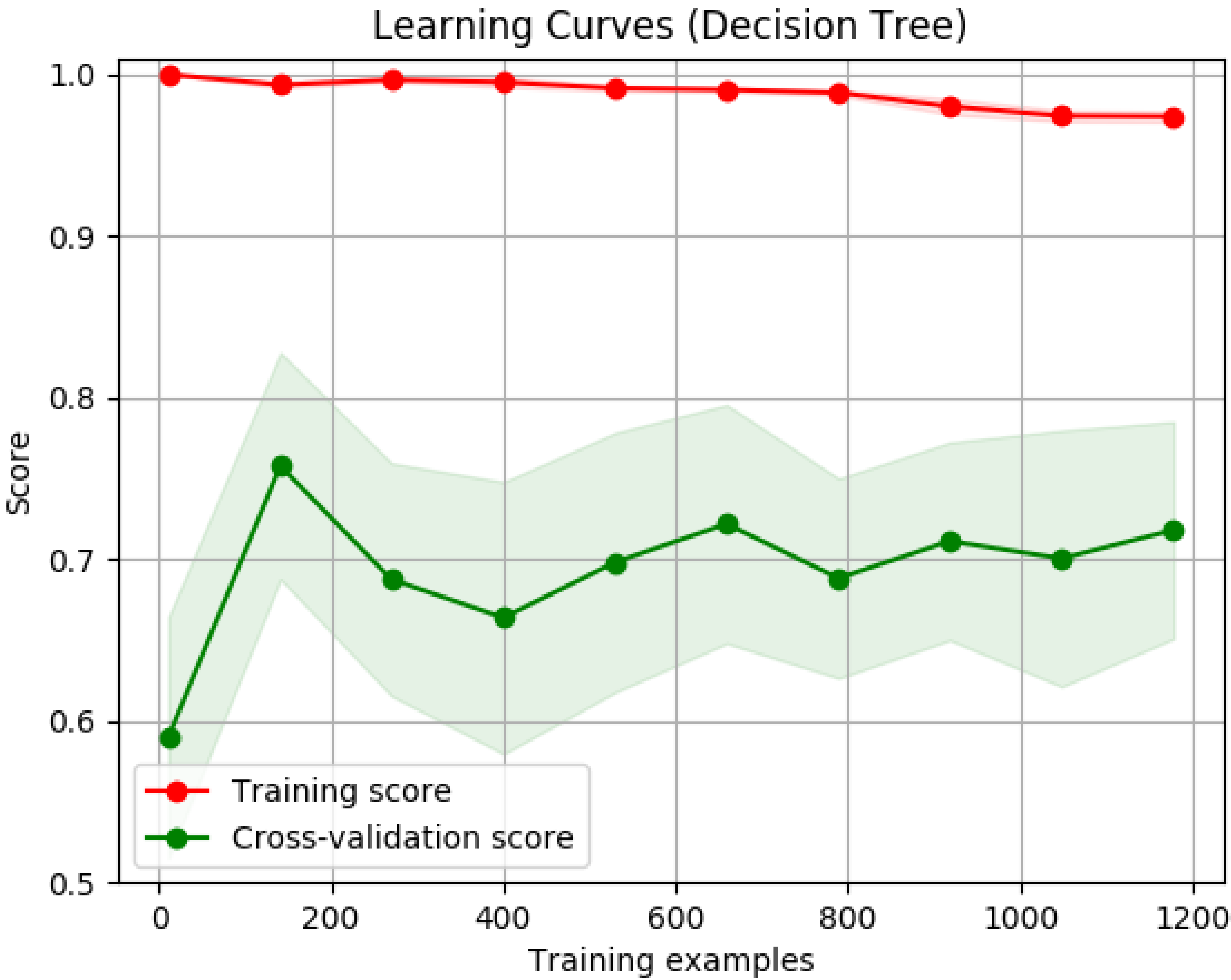
How Much Data Do We Need?

Learning Curve

```
>>> from sklearn.model_selection import learning_curve
>>> def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
...                         n_jobs=1, train_sizes=np.linspace(.01, 1.0, 10)):
...     fig = plt.figure()
...     plt.title(title)
...     if ylim is not None:
...         plt.ylim(*ylim)
...     plt.xlabel("Training examples")
...     plt.ylabel("Score")
...     train_sizes, train_scores, test_scores = learning_curve(
...         estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
...     train_scores_mean = np.mean(train_scores, axis=1)
...     train_scores_std = np.std(train_scores, axis=1)
...     test_scores_mean = np.mean(test_scores, axis=1)
...     test_scores_std = np.std(test_scores, axis=1)
...     plt.grid()
...
...     plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
...                      train_scores_mean + train_scores_std, alpha=0.1,
...                      color="r")
...     plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
...                      test_scores_mean + test_scores_std, alpha=0.1, color="g")
...     plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
...              label="Training score")
...     plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
...              label="Cross-validation score")
...
...     plt.legend(loc="best")
...     return fig, plt
```

Plot it

```
>>> title = "Learning Curves (Decision Tree)"
>>> fig, plt = plot_learning_curve(model,
...     title, X, y, ylim=(0.5, 1.01), cv=10, n_jobs=4)
>>> fig.savefig('img/ml-lc.png')
```


Learning Curves (Decision Tree)
Score
Training examples
1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
1200
Training score
Cross-validation score

Assignment

Learning Curves

Summary

Summary

- Scalable (can build trees per CPU)
- Reduces variance in decision tree
- No normalization of data (don't want money range ($0 - $10,000,000) overidding age (0-100)
- Feature importance (look at "mean decrease of impurity" where this node appears)
- Helps with missing data, outliers, and dimension reduction
- Works with both regression and classification
- Sampling allows "out of bag" estimate, removing need for test set

Why Python?

- Efficient algorithm
- Close to metal, 3000+ lines of Cython
- Faster than OpenCV (C++), Weka (Java), RandomForest (R/Fortran)

Attribution

- https://www.flickr.com/photos/eekim/2375990831
- Leornardo from internet

Thanks

`@__mharrison__`