

Mastering Applied Data Science

Day 4: Machine Learning - Classification



Agenda – Day 4

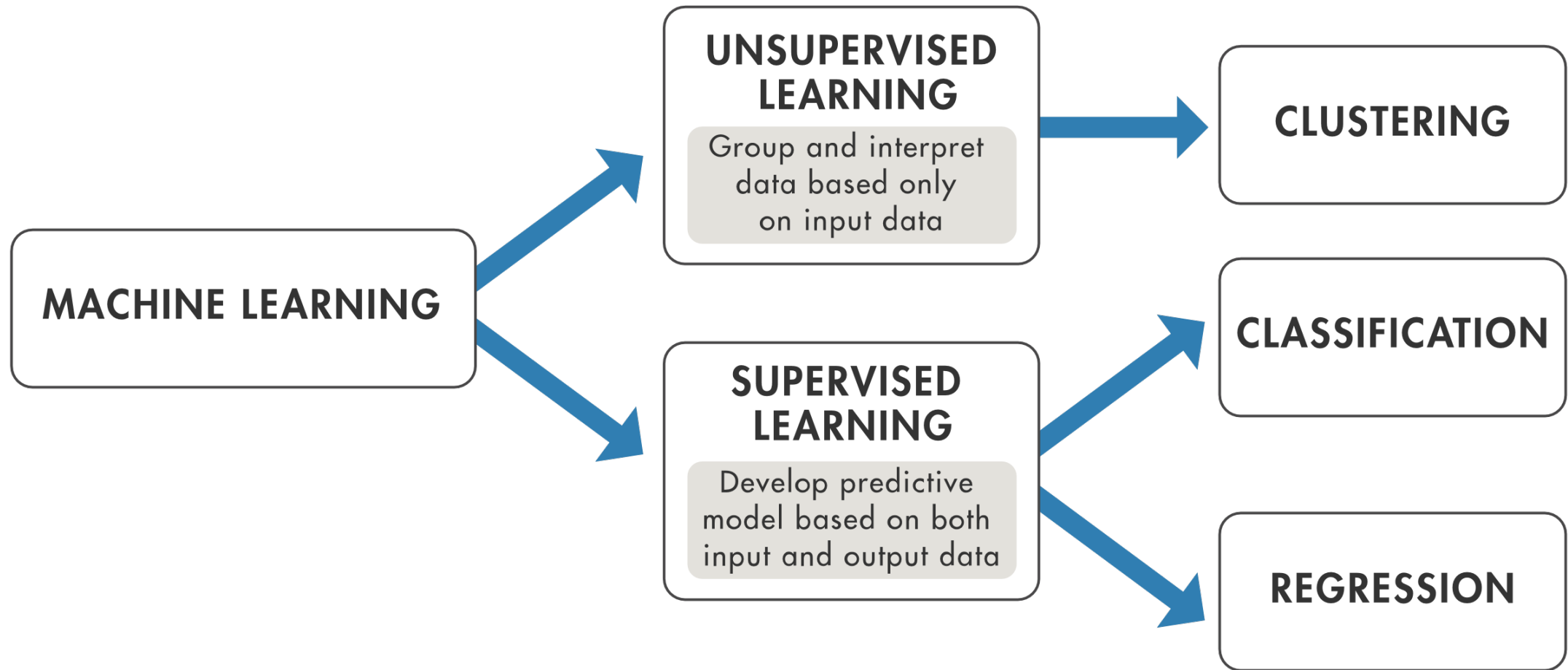
Classification

- Naive Bayes Classifier
- K-Nearest Neighbors Classifier
- Logistic Regression
- Decision Tree Classifier
- Ensemble models
 - Random Forest Classifier
 - Gradient Boost Classifier

Model Evaluation for Classification

- Accuracy score, Precision score, Recall score, F-1 score
- Confusion Matrix
- ROC curves and AUC

Machine Learning



Classification

Classification is about predicting a qualitative output from p inputs of arbitrary types (quantitative and/or qualitative).

Today's Use Case : HR Employee Turnover Prediction

- Data: hr_data.csv
- Business objective: Predict which employees are likely to leave next
- Data Setup:
 - Read in data
 - Target variable
 - Split training set
 - Data prep



Classification Algorithms

- **Probabilistic-based:**
 - Naïve Bayes Classifier
- **Group-Based:**
 - K-Nearest Neighbors Classifier
- **Maximum Entropy-Based:**
 - Logistic Regression
- **Tree-based:**
 - Decision Tree Classifier
- **Ensemble models:**
 - Random Forest Classifier
 - Gradient Boost Classifier

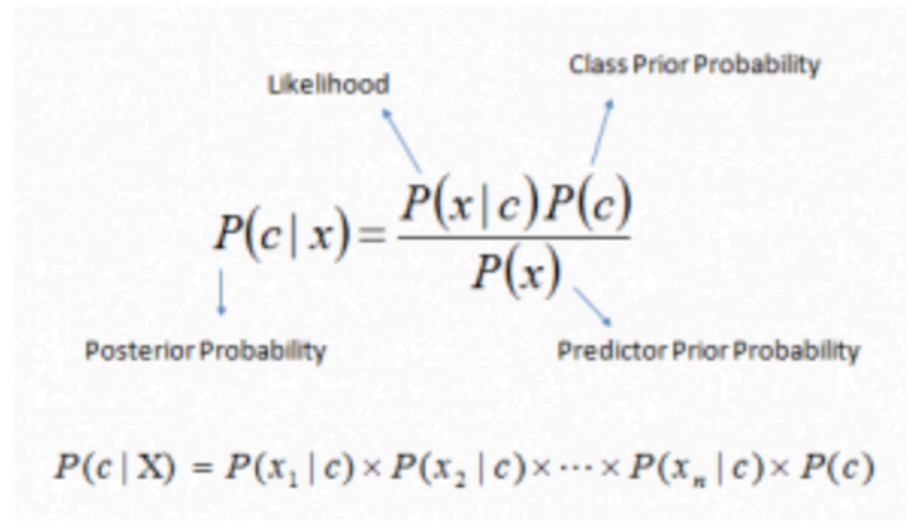
Theoretical Comparisons of Algorithms

	Naïve Bayes	K-Nearest Neighbors	Logistic Regression	Decision Tree
When	Features are assumed / proven to not be related: word frequencies.	'Birds of a feather, flock together.' fits with your data.	Groups can be considered: 'clear cut' between.	Groups are not distinct, but based on values within features.
Pros	Small bias, usually better than guessing. Trains linearly rather than approx., thus fast.	Non-parametric. Highly interpretable. Small bias.	Highly interpretable. Smaller bias.	Highly interpretable & mappable. Non-parametric.
Cons	Horrible when features are related. Works best as baseline for comparisons.	Slow as a baseline: trains slowly, predicts slowly.	Parametric: groups split by line. Breaks with a lot of groups.	Less data = high bias. More data = high variance.

Naïve Bayes

The **Naive Bayes algorithm** is a classification technique based on Bayes' Theorem with an assumption of independence among predictors.

- It perform well in case of categorical input variables compared to numerical variable(s).
- Gaussian Naive Bayes is used in classification and it assumes that features follow a normal distribution.
- Bernoullie Naive Bayes is useful if your feature vectors are binary (i.e. zeros and ones).



The diagram illustrates the Naive Bayes classification formula. It shows the equation $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$ with arrows pointing from descriptive labels to the corresponding parts of the formula. 'Likelihood' points to $P(x|c)$, 'Class Prior Probability' points to $P(c)$, 'Posterior Probability' points to $P(c|x)$, and 'Predictor Prior Probability' points to $P(x)$. Below the main equation, the expanded formula is given: $P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Labels in the diagram:

- Likelihood (points to $P(x|c)$)
- Class Prior Probability (points to $P(c)$)
- Posterior Probability (points to $P(c|x)$)
- Predictor Prior Probability (points to $P(x)$)

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Naïve Bayes

`sklearn.naive_bayes` : Naive Bayes

The `sklearn.naive_bayes` module implements Naive Bayes algorithms. These are supervised learning methods based on applying Bayes' theorem with strong (naive) feature independence assumptions.

User guide: See the [Naive Bayes](#) section for further details.

<code>naive_bayes.BernoulliNB ([alpha, binarize, ...])</code>	Naive Bayes classifier for multivariate Bernoulli models.
<code>naive_bayes.GaussianNB ([priors])</code>	Gaussian Naive Bayes (GaussianNB)
<code>naive_bayes.MultinomialNB ([alpha, ...])</code>	Naive Bayes classifier for multinomial models

Naïve Bayes

1. Import Algorithms
2. Set Parameters
3. Fit the data

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
g = GaussianNB()
b = BernoulliNB()
```

```
g.fit(X.drop('Attrition', axis = 1), X['Attrition'])
b.fit(X.drop('Attrition', axis = 1), X['Attrition'])
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

K-Nearest Neighbors Classifier

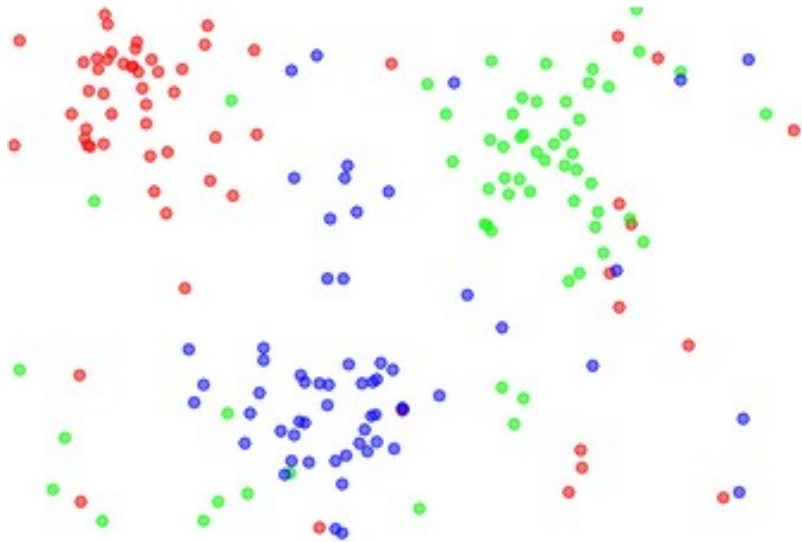
The **K-Nearest Neighbors (KNN) algorithm** is non-parametric, instance-based and used in a supervised learning setting.

It is simple to understand and easy to implement, and can be used to solve both classification and regression problems. However, the algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

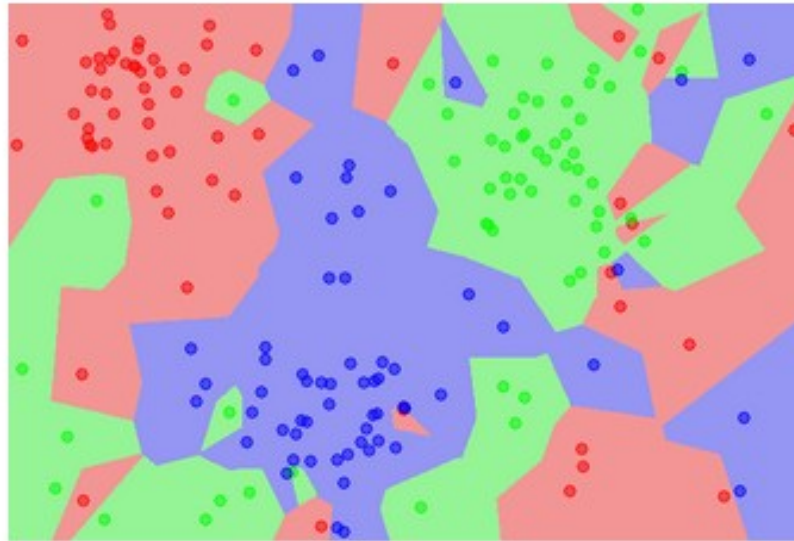
KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

K-Nearest Neighbors Classifier

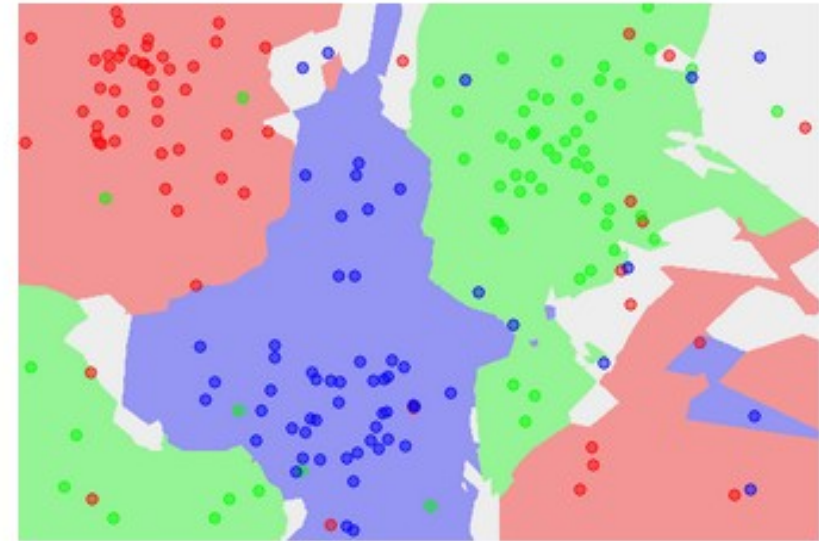
the data



NN classifier



5-NN classifier



K-Nearest Neighbors

1. Import Algorithms
2. Set Parameters
3. Fit the data

- <http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

```
from sklearn.neighbors import KNeighborsClassifier  
k = KNeighborsClassifier()
```

```
k.fit(X.drop('Attrition', axis = 1), X['Attrition'])
```

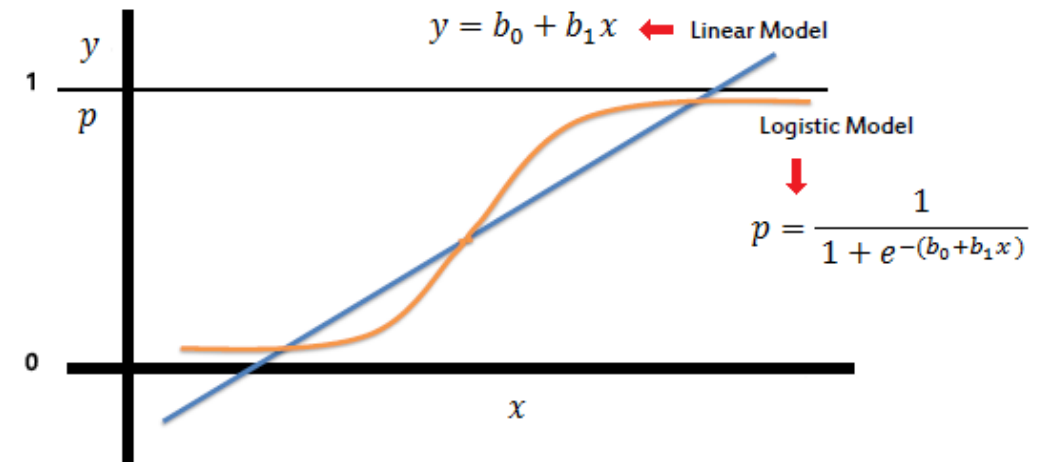
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,  
                    weights='uniform')
```

Logistic Regression

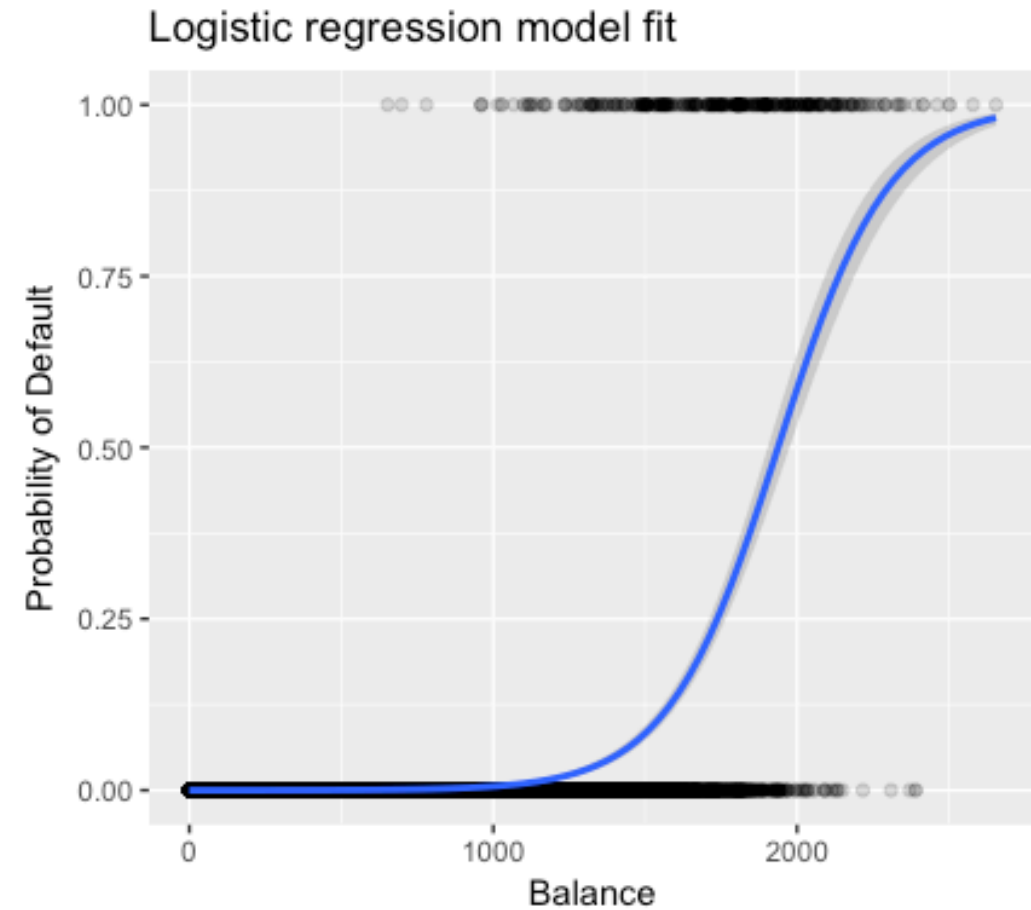
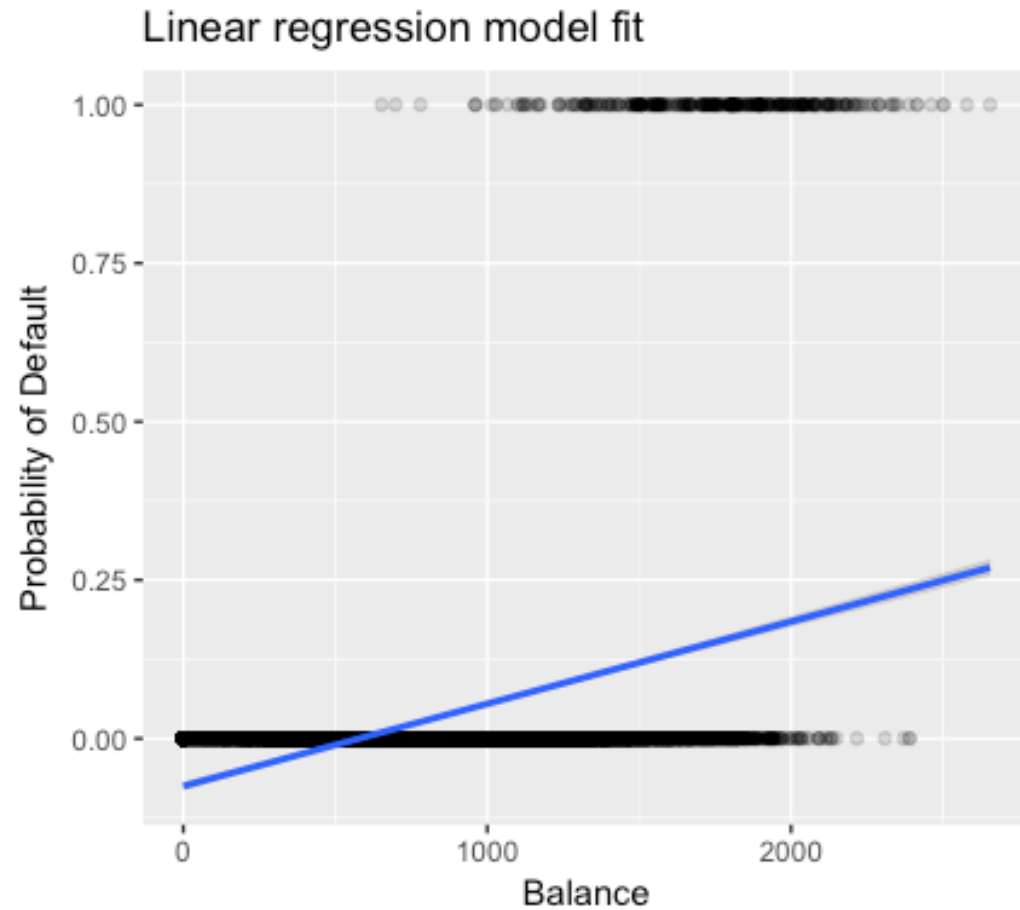
Logistic regression (aka logit regression or logit model) is a classification algorithm where the response variable Y is categorical.

It transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$



Logistic Regression



Logistic Regression

1. Import Algorithms
2. Set Parameters
3. Fit the data

- http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

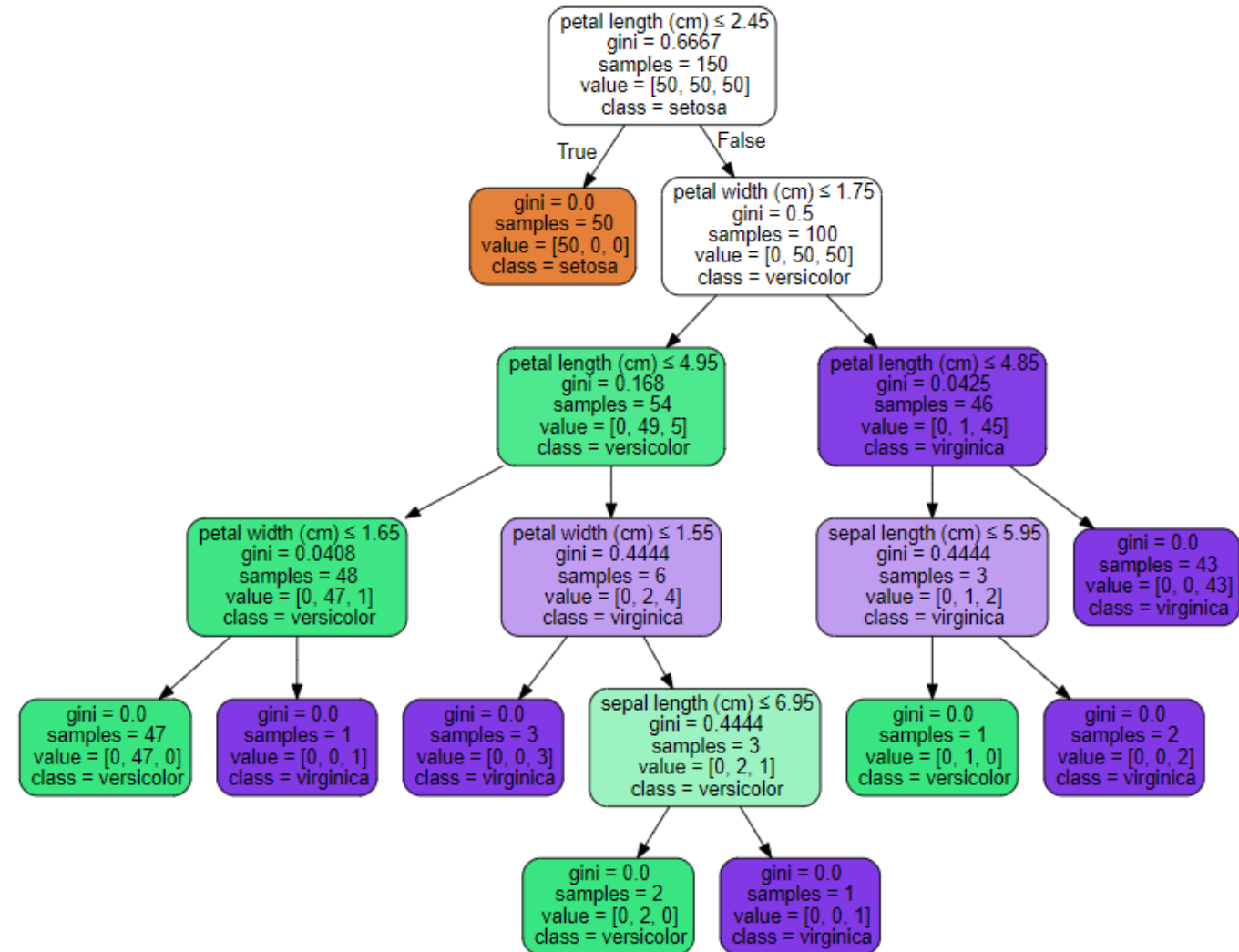
```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
```

```
log.fit(X.drop('Attrition', axis = 1), X['Attrition'])
```

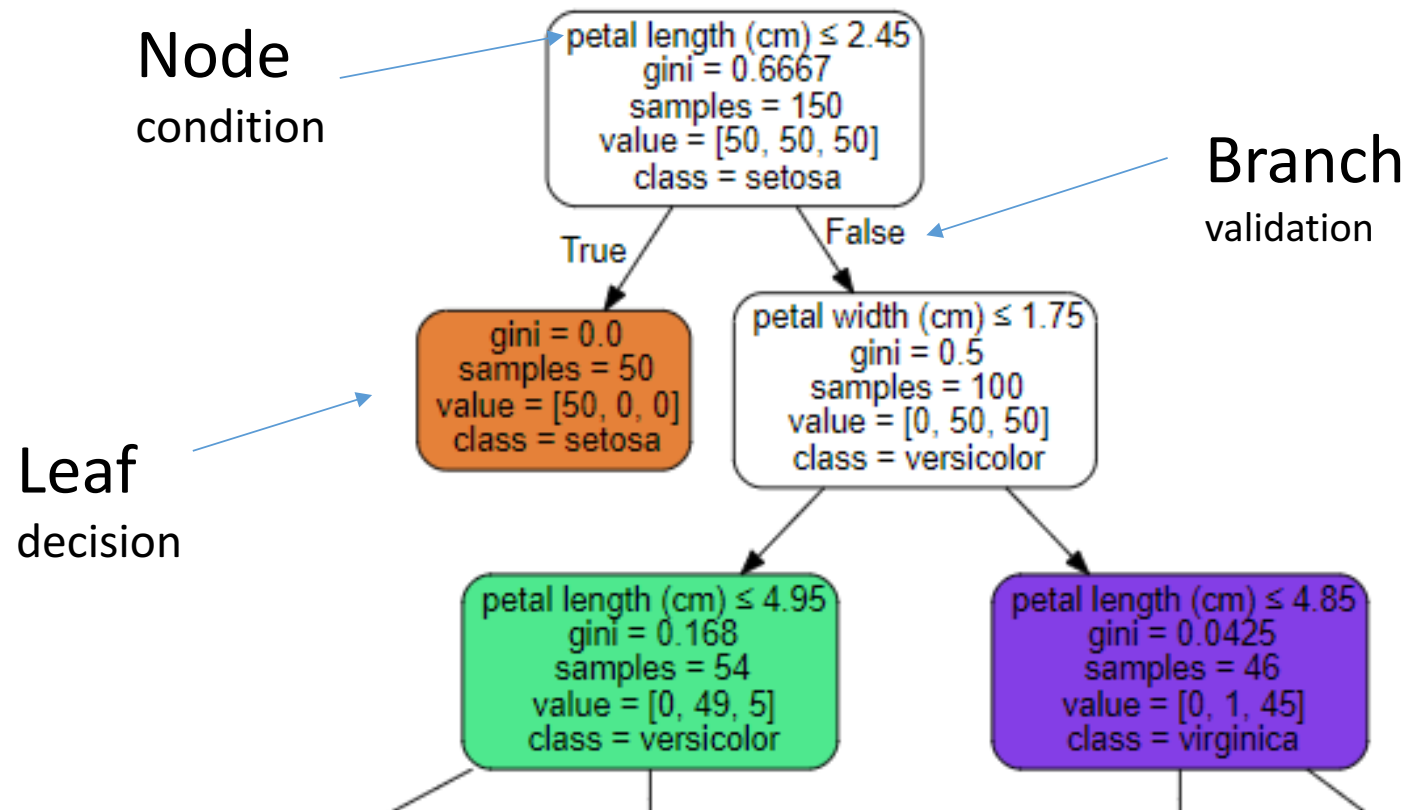
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='warn',
                    n_jobs=None, penalty='l2', random_state=None, solver='warn',
                    tol=0.0001, verbose=0, warm_start=False)
```


Introduction to Tree-based Algorithms

Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation.



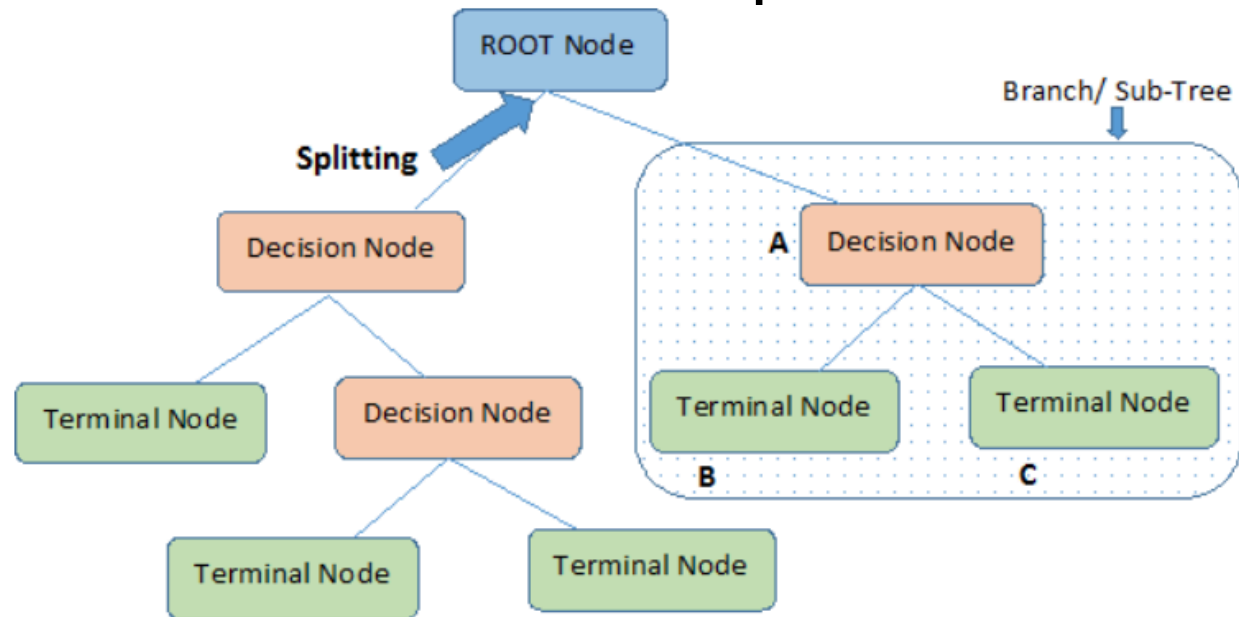
Nodes go through Branches to make Leaves.



Decision Tree

Decision tree works for both categorical and continuous input and output variables.

In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.



Note:- A is parent node of B and C.

Decision Tree

1. Import Algorithms
2. Set Parameters
3. Fit the data

- <http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

```
from sklearn.tree import DecisionTreeClassifier  
d = DecisionTreeClassifier()
```

```
d.fit(X.drop('Attrition', axis = 1), X['Attrition'])
```

```
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')
```

Introduction to Ensemble Models

Ensemble methods, which combines several decision trees to produce better predictive performance than utilizing a single decision tree. The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner.

Ensemble methods involve group of predictive models to achieve a better accuracy and model stability.

Introduction to Ensemble Models

Bagging

Can reduce variance in large data impacted algorithms, like decision tree.

Process: bootstrapping

- Subsamples from train data with replacement.
- Trains on all subsamples.
- Combines all training by majority-class.
- Creates a model based on most predicted.

Boosting

Can reduce variance & bias in large data impacted algorithms.

Process: weight-adjustment.

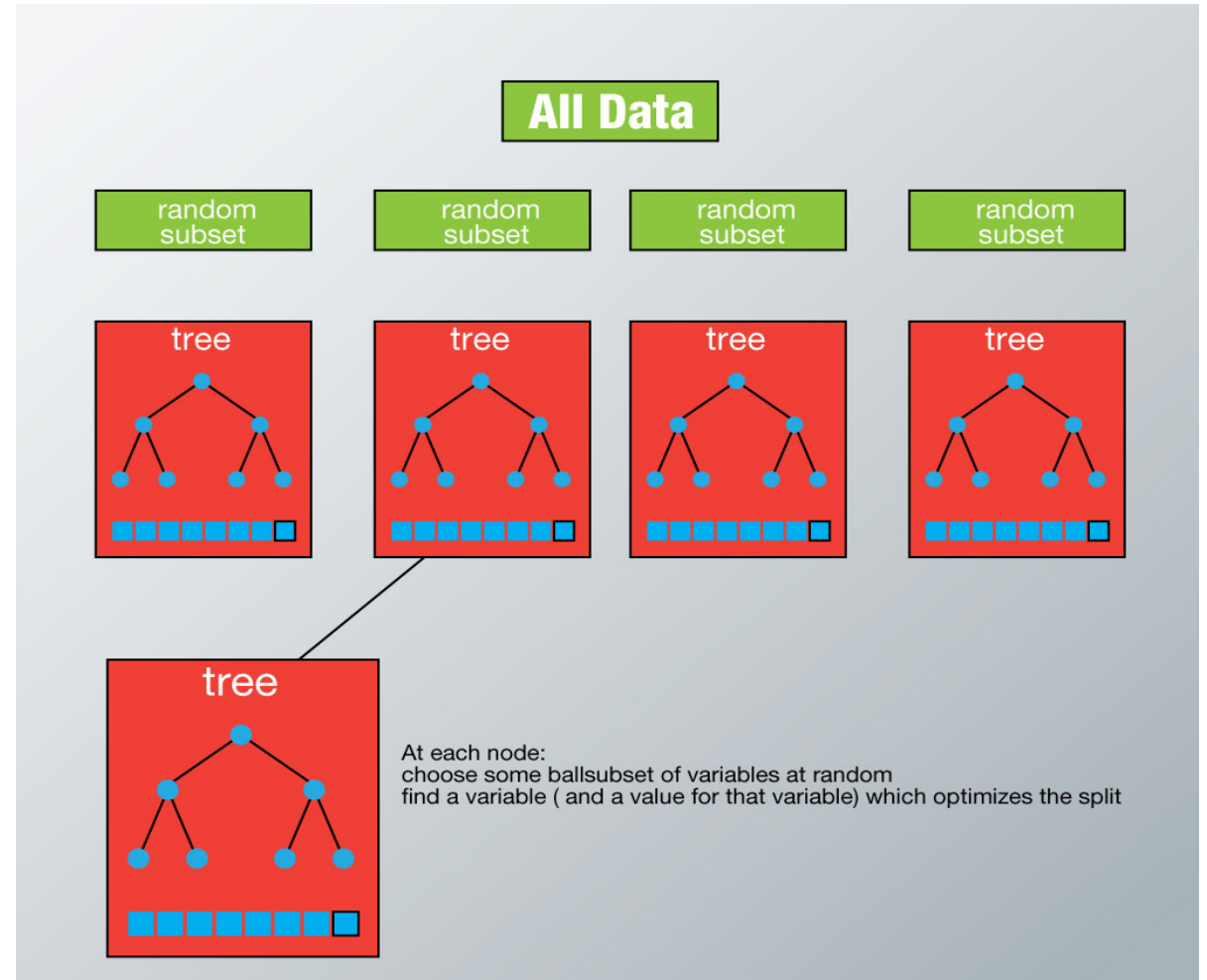
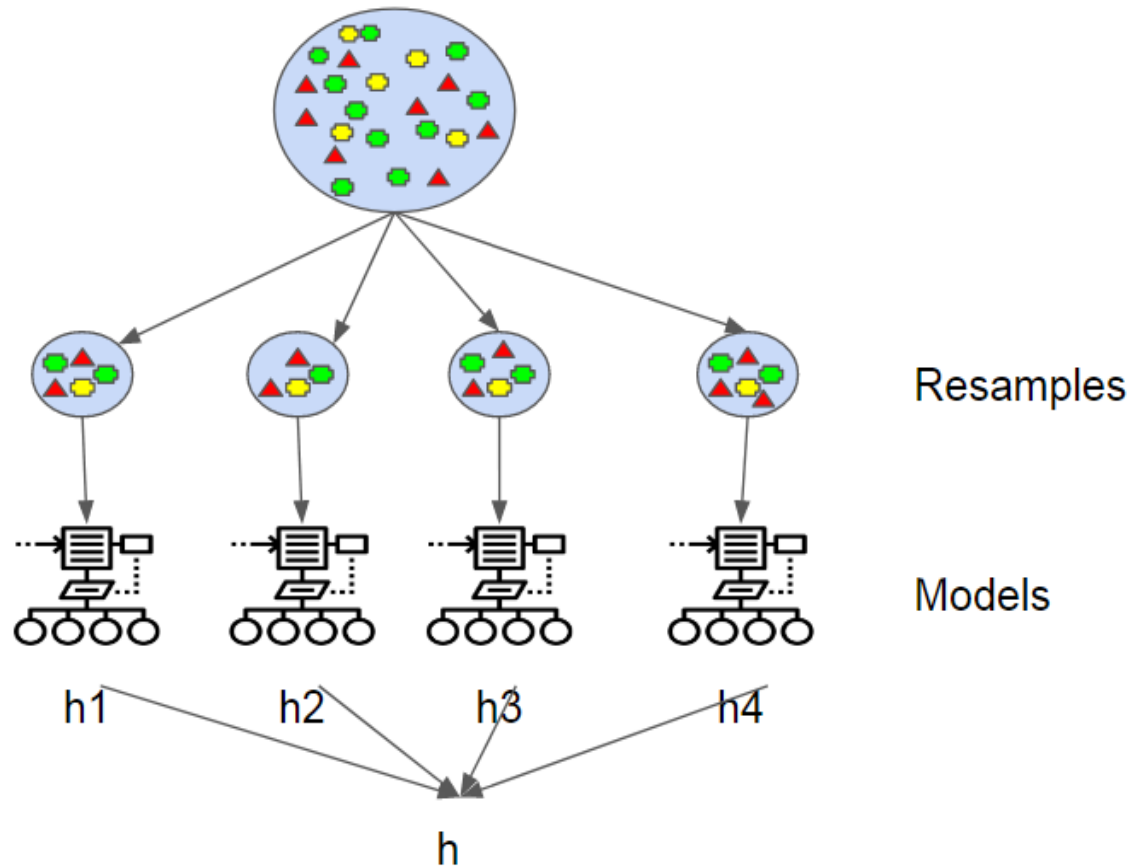
- Set training data subsample, with incorrect weights.
- Trains on subsample to adjust weights.
- Iterates until weights are optimized.
- Creates a model based on most optimized.

Bagging, Random Forest Classifier

Bagging (Bootstrap Aggregation) is used when our goal is to reduce the variance of a decision tree. *Here idea is to create several subsets of data from training sample chosen randomly with replacement. Now, each collection of subset data is used to train their decision trees. As a result, we end up with an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree.*

Random Forest is an extension over bagging. It takes one extra step where in addition to taking the random subset of data, it also takes the random selection of features rather than using all features to grow trees. When you have many random trees. It's called Random Forest

Bagging, Random Forest Classifier



Bagging, Random Forest Classifier

1. Import Algorithms
 2. Set Parameters
 3. Fit the data
- <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

```
from sklearn.ensemble import RandomForestClassifier  
r = RandomForestClassifier()
```

```
r.fit(X.drop('Attrition', axis = 1), X['Attrition'])
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',  
                        max_depth=None, max_features='auto', 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, n_estimators=10, n_jobs=None,  
                        oob_score=False, random_state=None, verbose=0,  
                        warm_start=False)
```

Boosting, Gradient Boosting Classifier

- **Boosting** is another ensemble technique to create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analyzing data for errors. In other words, we fit consecutive trees (random sample) and at every step, the goal is to solve for net error from the prior tree.
- **Gradient Boosting** is an extension over boosting method. ***Gradient Boosting = Gradient Descent + Boosting***. It uses gradient descent algorithm which can optimize any differentiable loss function. An ensemble of trees are built one by one and individual trees are summed sequentially. Next tree tries to recover the loss (difference between actual and predicted values).

Boosting, Gradient Boosting Classifier

1. Import Algorithms
2. Set Parameters
3. Fit the data

- <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>

```
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
```

```
gbc.fit(X.drop('Attrition', axis = 1), X['Attrition'])
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           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, n_estimators=100,
                           n_iter_no_change=None, presort='auto', random_state=None,
                           subsample=1.0, tol=0.0001, validation_fraction=0.1,
                           verbose=0, warm_start=False)
```

Model Evaluation

Classification Algorithms

When evaluating classification models there are many possible metrics to assess performance.

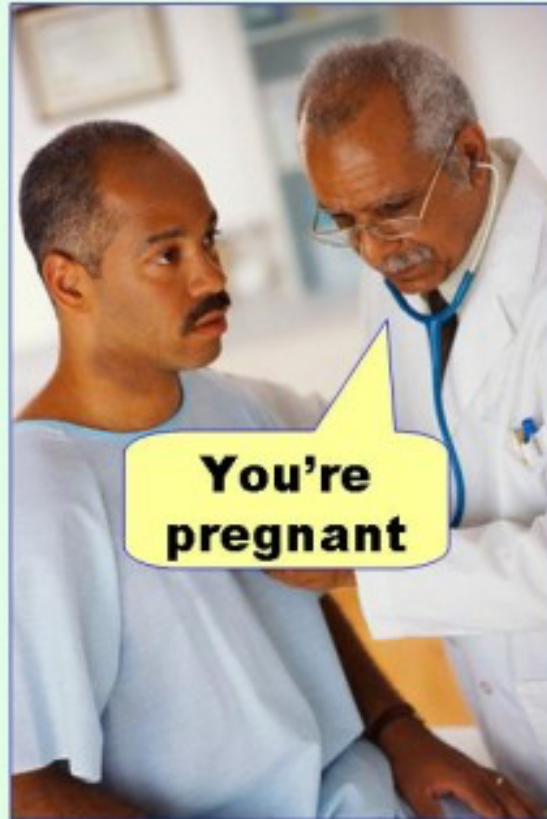
- Accuracy, Precision, Recall, F1 scores
- Confusion Matrix
- ROC curve and AUC

Confusion Matrix

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

CONFUSION MATRIX

Type I error
(false positive)



Type II error
(false negative)



Confusion Matrix

n=165	Predicted: NO	Predicted: YES	
	Actual: NO	Actual: YES	
	TN = 50	FP = 10	60
	FN = 5	TP = 100	105
	55	110	

Basic Terminology:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)

Accuracy:

- Overall, how often is it **correct**?
- $(TP + TN) / \text{total} = 150/165 = 0.91$

Misclassification Rate (Error Rate):

- Overall, how often is it **wrong**?
- $(FP + FN) / \text{total} = 15/165 = 0.09$

Precision, Recall, and F-Score

Precision

- $P = \frac{TP}{TP+FP}$
- Positive Predicted Value

Recall

- $R = \frac{TP}{TP+FN}$
- Also known as Sensitivity

F-Score (F₁-Score)

- $F_1 \text{ Score} = 2 * \frac{P * R}{P + R}$
- A measure that combines P and R

Recall

Precision

n=165		Predicted: NO	Predicted: YES	
Actual: NO		TN = 50	FP = 10	60
Actual: YES		FN = 5	TP = 100	105
		55	110	

Confusion matrix

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

False Positive Rate:

- When actual value is **negative**, how often is prediction **wrong**?
- $FP / \text{actual no} = 10/60 = 0.17$

Sensitivity:

- When actual value is **positive**, how often is prediction **correct**?
- $TP / \text{actual yes} = 100/105 = 0.95$
- “True Positive Rate” or “Recall”

Specificity:

- When actual value is **negative**, how often is prediction **correct**?
- $TN / \text{actual no} = 50/60 = 0.83$

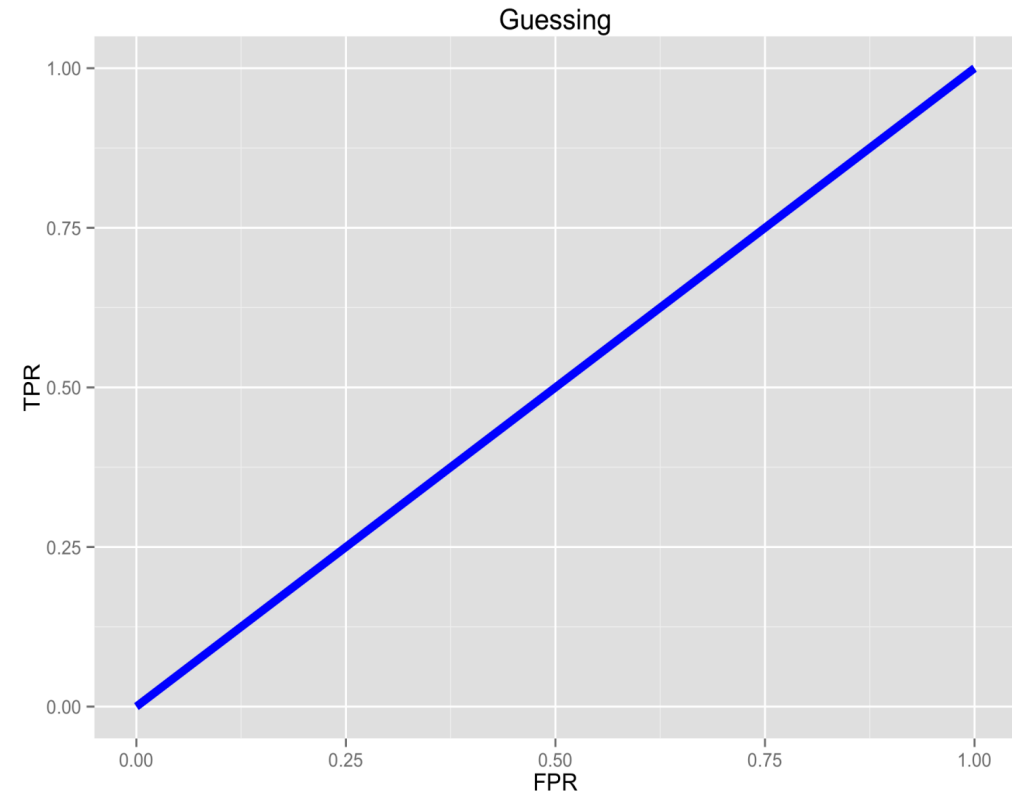
RECEIVING OPERATING CHARACTERISTIC - ROC

- Receiving Operating Characteristic, or ROC, is a visual way for inspecting the performance of a binary classifier (0/1). In particular, it's comparing the rate at which your classifier is making correct predictions (True Positives or TP) and the rate at which your classifier is making false alarms (False Positives or FP).
- $TPR = \text{TruePositives} / (\text{TruePositives} + \text{FalseNegatives})$
- $FPR = \text{FalsePositives} / (\text{FalsePositives} + \text{TrueNegatives})$

RECEIVING OPERATING CHARACTERISTIC - ROC

A diagonal line indicates that the classifier is just making completely random guesses. Since your classifier is only going to be correct 50% of the time

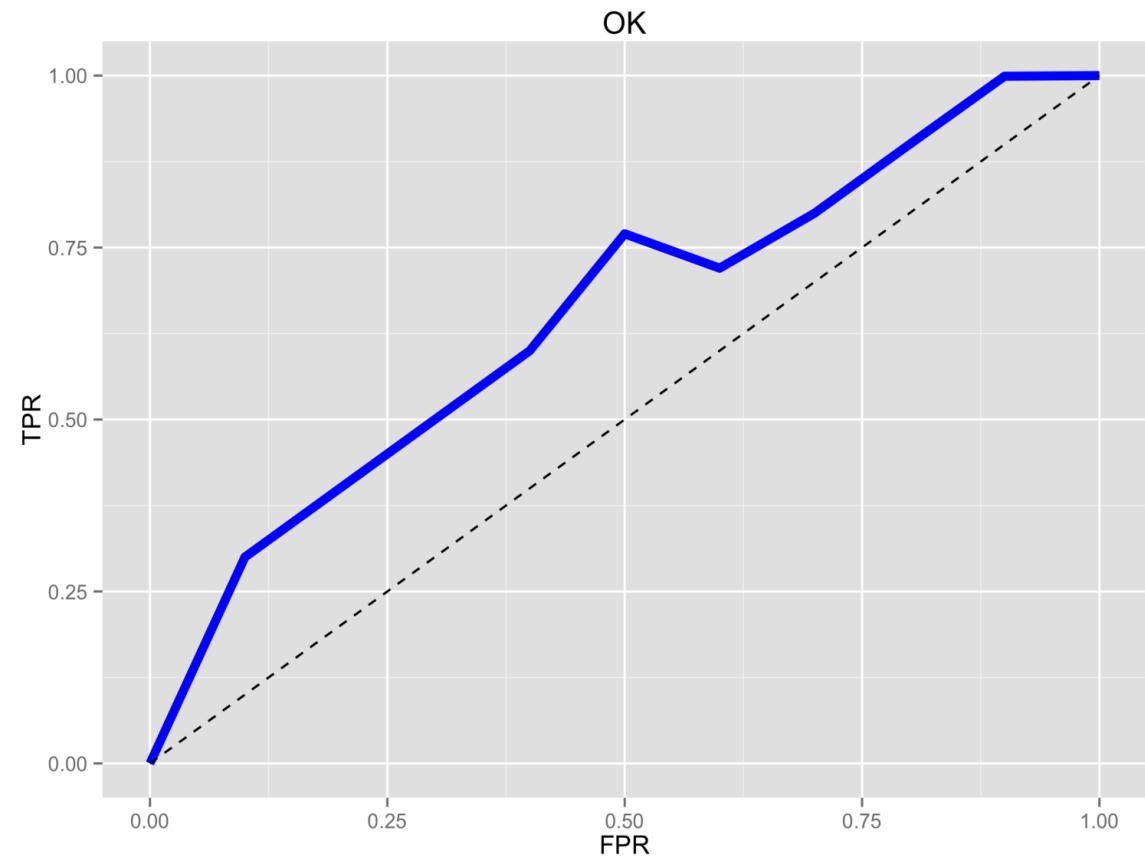
ROC CURVE



RECEIVING OPERATING CHARACTERISTIC - ROC

An okay classifier is better than random however fairly close to it

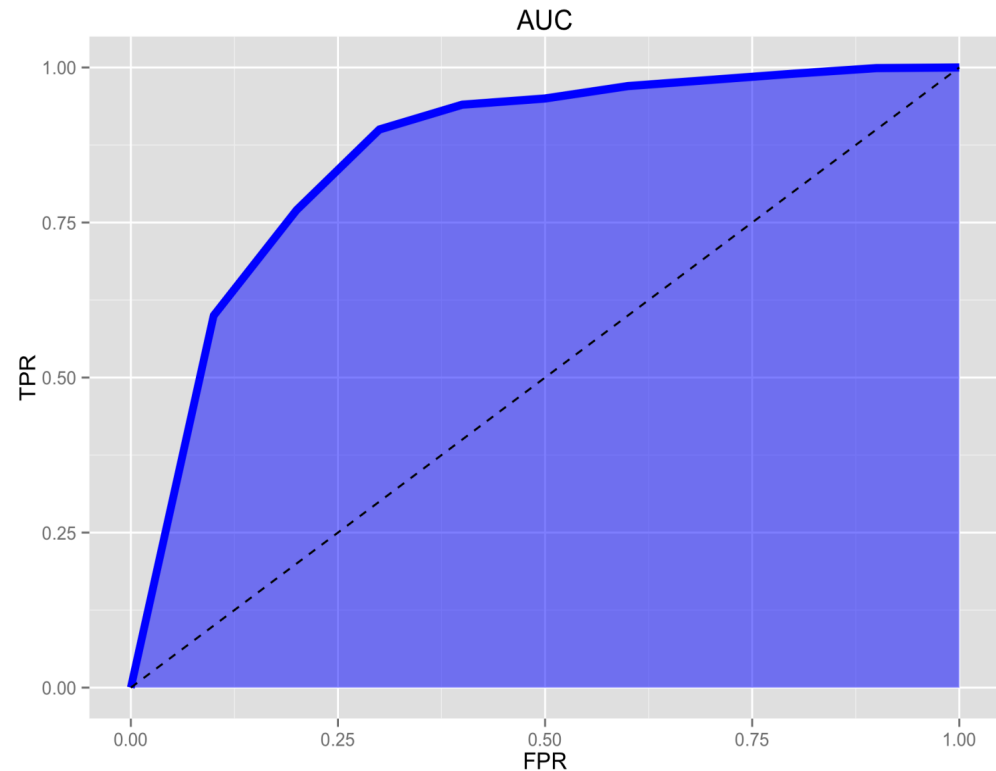
ROC CURVE



RECEIVING OPERATING CHARACTERISTIC - ROC

Area under the curve is the amount of space underneath the ROC curve

AREA UNDER THE CURVE - AUC

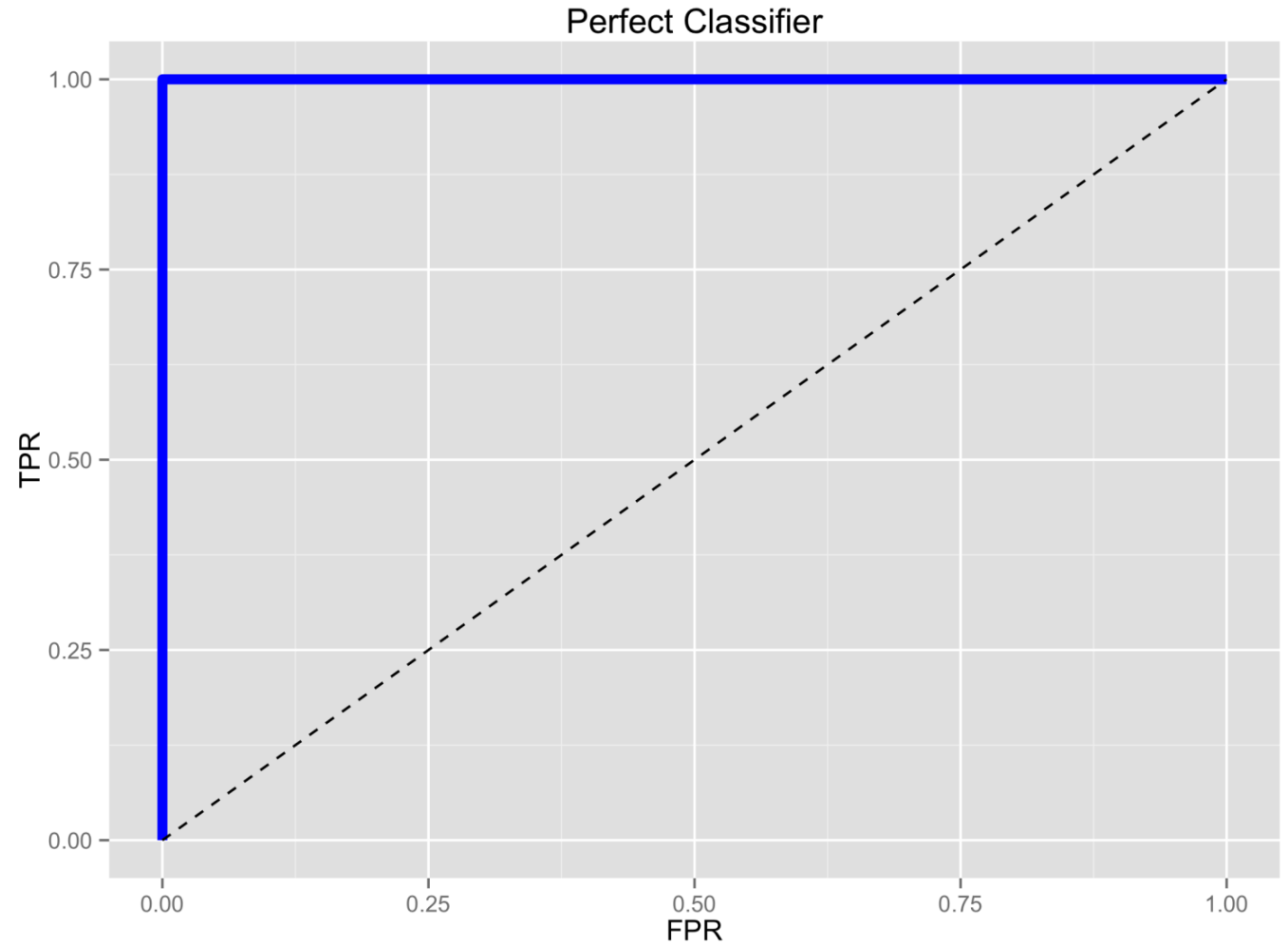


CLASSIFICATION METRICS

- PCC. Percent Correct Classification
- Equivalence:
 - Sensitivity = Recall
 - False Dismissal Rate (FD) = $1 - \text{Sensitivity}$
 - False Alarm Rate (FA) = $1 - \text{Specificity}$
 - Specificity = True Negative Rate
 - Type I Error = FA Rate
 - Type II Error = FD Rate

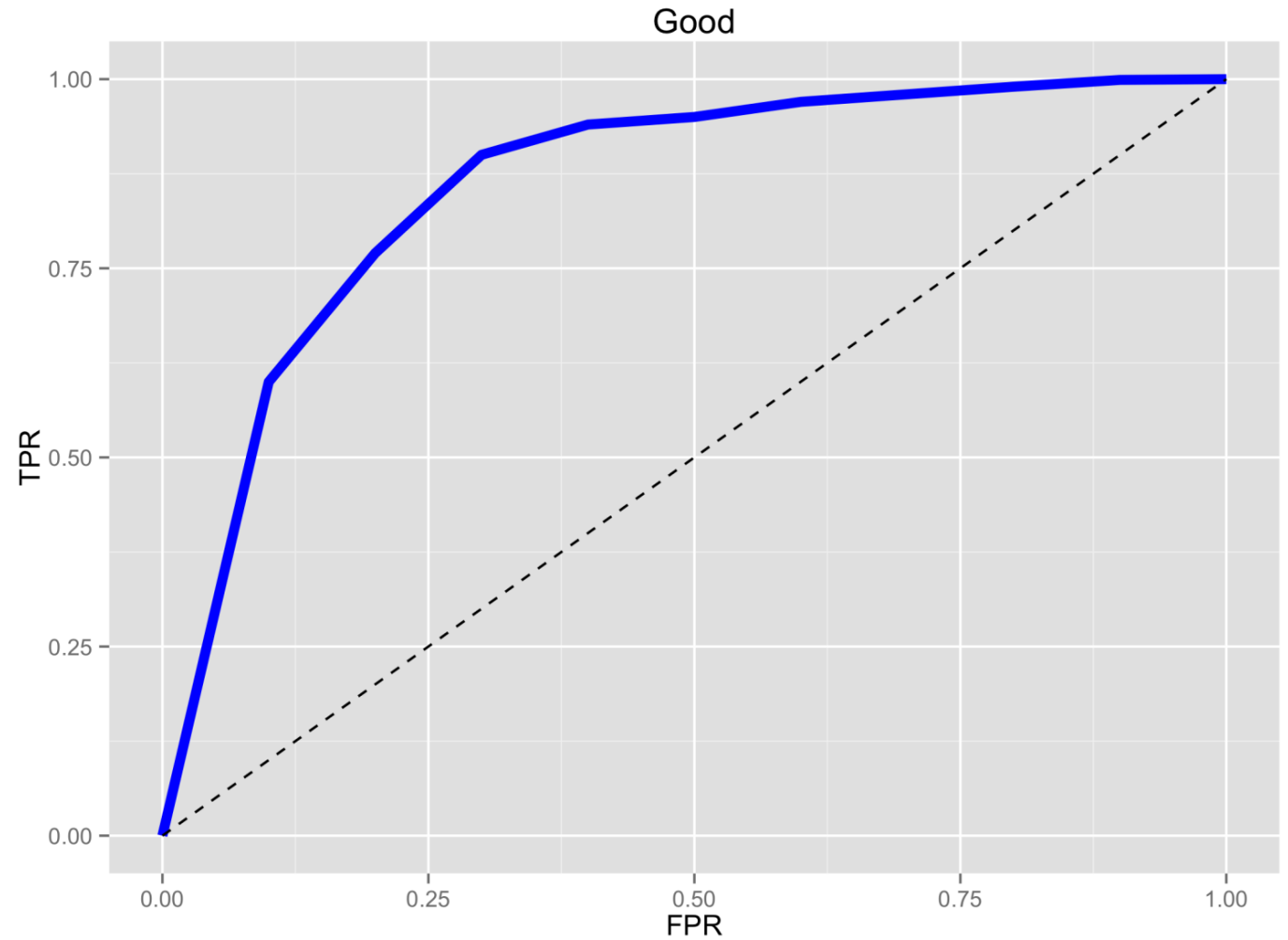
ROC CURVE

- A PERFECT classifier—
is something that makes
every prediction correctly.



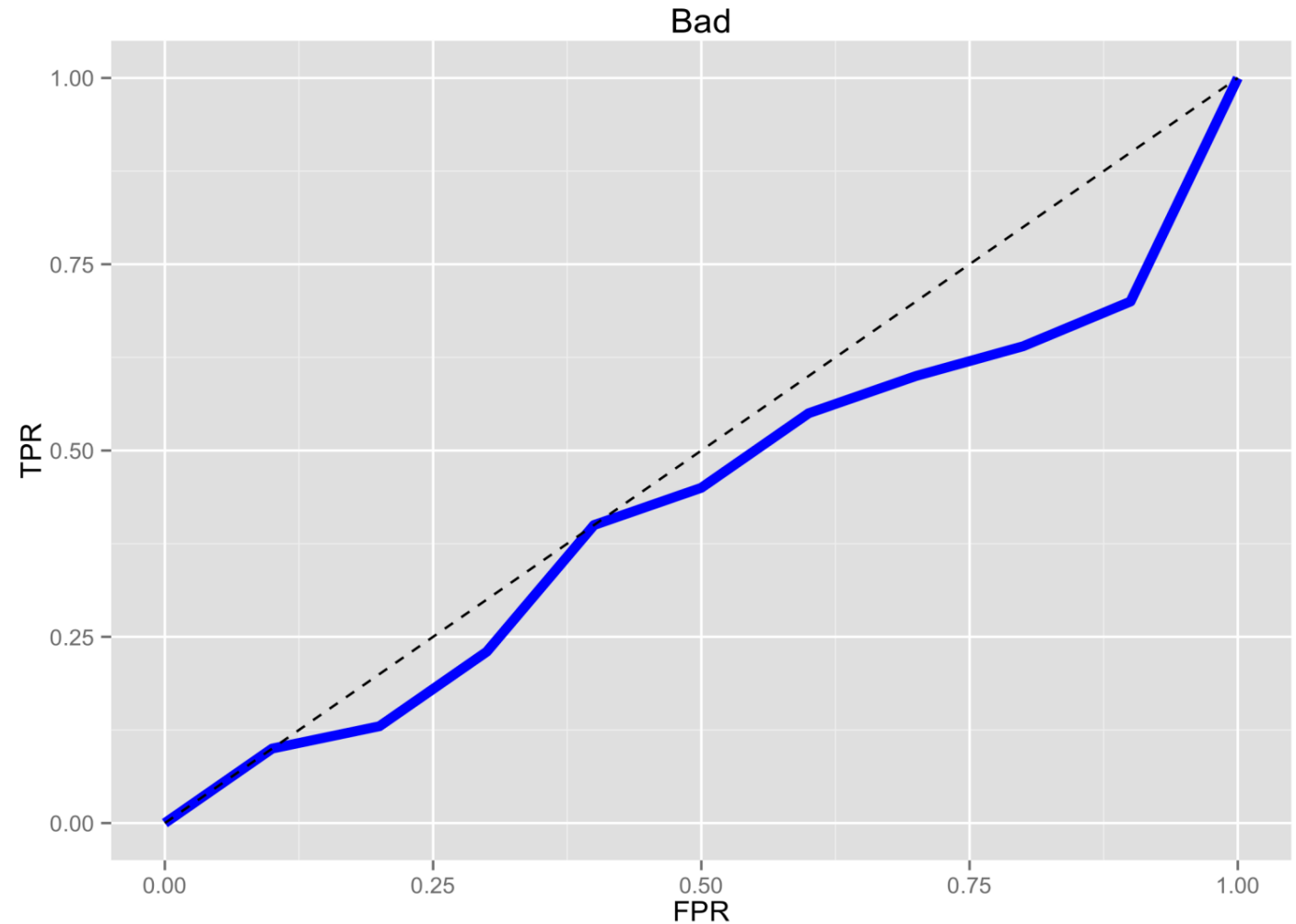
ROC CURVE

- A Good classifier is will look like this – It is closer to a perfect classifier



ROC CURVE

- A bad classifier is something that's *worse than guessing* and will appear below the random line.



Use Case: Model Evaluation

- Define target variable and feature variables

```
X = pd.get_dummies(filled, drop_first = True)
target = X['Attrition']
features = X.drop('Attrition', axis = 1)
```

- Define the algorithms to be evaluated

```
algorithms = [g,b,k,log,d,r,gb]
names = ['GaussianNB', 'BernoulliNB', 'K Nearest', 'Logistic', 'Single Tree', 'Random Forest', 'Gradient Boost']
```

Use Case: Model Evaluation

- Define a function to evaluate the models:

```
evaluate_model()
```

	Accuracy	Precision	Recall	F1
Random Forest	0.980083	0.000000	0.980083	0.980083
Gradient Boost	0.960996	0.960996	0.960996	0.960996
Logistic	0.887967	0.887967	0.887967	0.887967
K Nearest	0.855602	0.855602	0.855602	0.855602
BernoulliNB	0.844813	0.844813	0.844813	0.844813
GaussianNB	0.808299	0.808299	0.808299	0.808299
Single Tree	0.000000	0.000000	0.000000	0.000000

Use Case: Model Evaluation

- We can also check the confusion matrix and classification report for the predictions

```
confusion_matrix(target, pred_gbc)
```

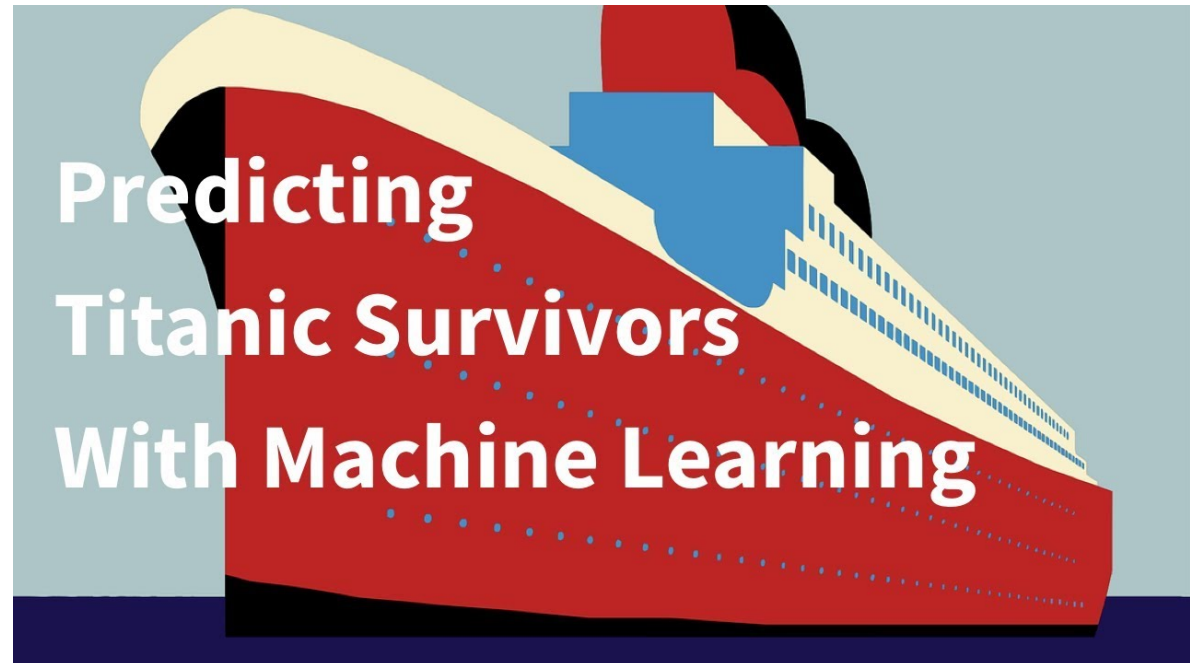
```
array([[1019,    1],  
       [  46,   139]])
```

```
print(classification_report(target, pred_gbc))
```

	precision	recall	f1-score	support
0.0	0.96	1.00	0.98	1020
1.0	0.99	0.75	0.86	185
micro avg	0.96	0.96	0.96	1205
macro avg	0.97	0.88	0.92	1205
weighted avg	0.96	0.96	0.96	1205

Take-home Project

- Data: train.csv and test.csv
- Use the Titanic data to predict which passengers will survive



ANY QUESTIONS?

Zia Khan

Senior Data Scientist

theDevMasters

zia@thedevmasters.com

www.brandman.edu/DataScience

