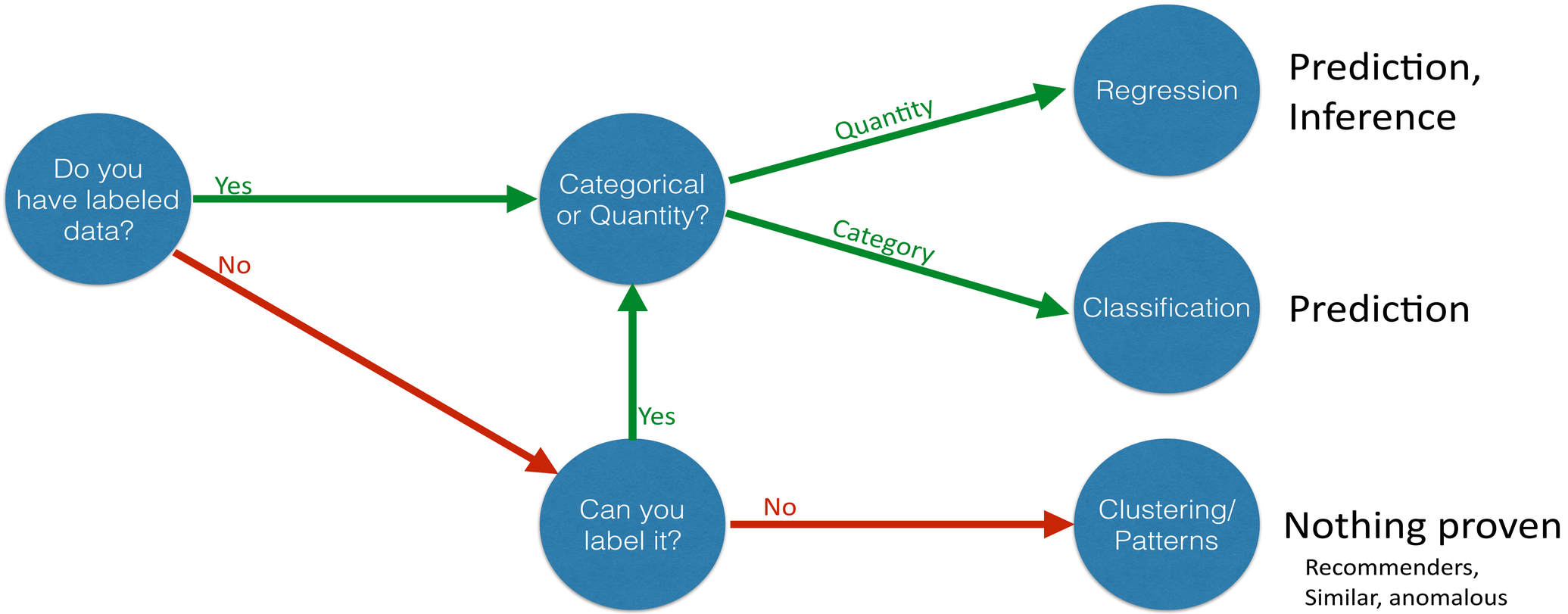

GTK CYBER
GET ON THE FAST TRACK

Machine Learning for Security Professionals - Day 2

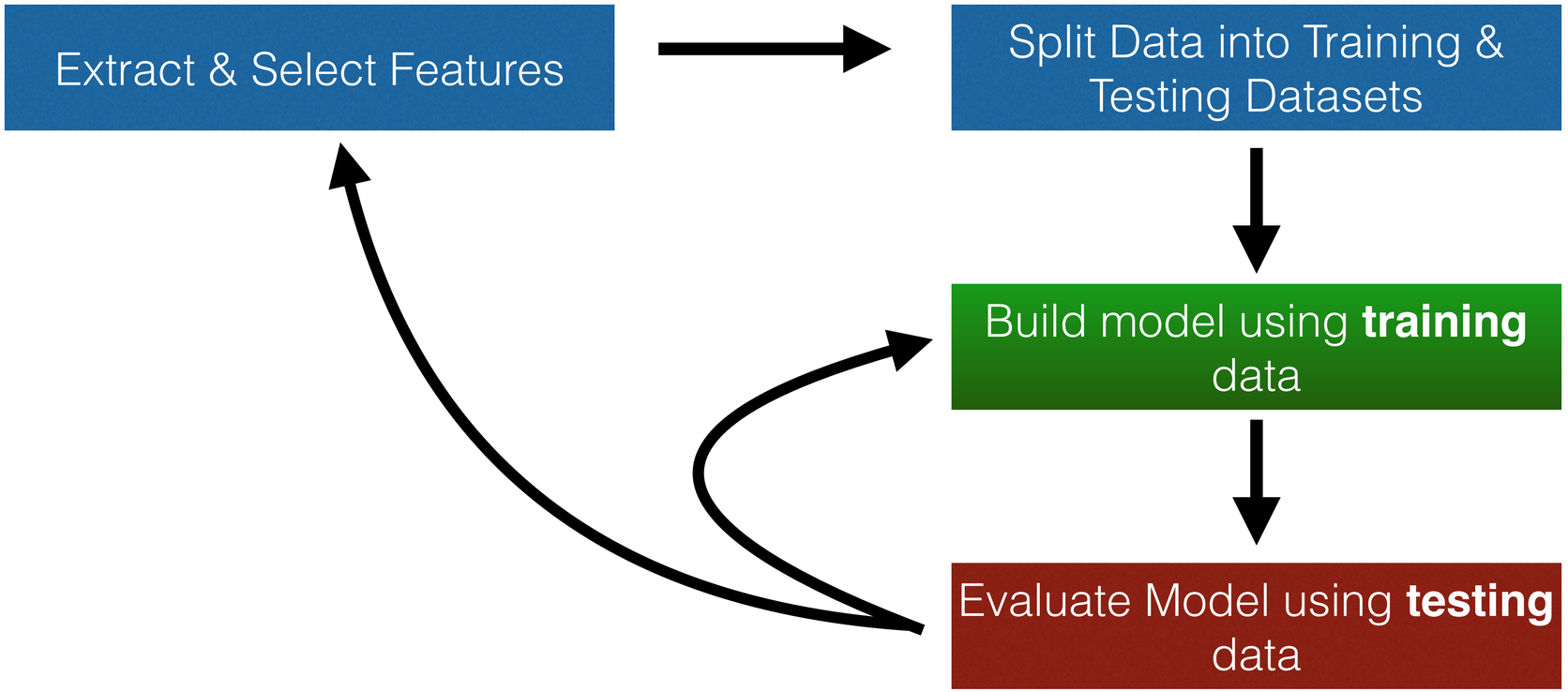
Supervised Learning: Classification

Agenda for Today

- Feature Selection & Engineering
- Math free overview of classification models
- Evaluating Model Performance
- Improving model performance


Do you have labeled data?
Yes
No
Categorical or Quantity?
Quantity
Category
Regression
Classification
Yes
Can you label it?
No
Clustering/ Patterns
Prediction, Inference
Prediction
Nothing proven
Recommenders,
Similar, anomalous

Supervised ML Process



Machine Learning: Feature Engineering

Use Case: From URL strings to “Features”

Features

http://www.google.com

URL Definition

https://www.google.com/search?q=URL&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjcl6ut-IDUAhVEPCYKHdJGDsYQ_AUIDCgD&biw=1215&bih=652

https://	protocol
www	subdomain
google.com	zone apex
google	domain
.com	top-level-domain (tld)
/search?q=URL…	path

DNS 101

- Domain Name Service (DNS) resolves domain names to IP addresses (like a phone book)
- Domain Registrars: authority that signs unique domain names
- State of Authority (SOA): Contains for example name of server for zone, administrator of zone, default time-to-live (ttl = time a DNS record is cached), seconds of secondary name server should wait before checking for updates
- Root Zone controlled by Internet Assigned Numbers Authority (IANA)
- Name Servers (NS Records): used by tld servers to direct traffic to DNS server (which contains authoritative DNS records)
- A records (part of DNS record): “A” stands for IP Address
- CNAME (part of DNS record): resolves one domain name to another
- Autonomous System (AS) and Border gateway Protocol (BGP) info

Python libraries: **`python-whois, dnspython, tldextract, ipaddress`**

Representation of URL Knowledge

- Come up with a representation/set of knowledge that has enough complexity to accurately describe the problem for the computer
- Knowledge here does not mean hard-coded knowledge or formal set of rules
- The computer rather uses the knowledge we provide to extract patterns and acquire own knowledge
- We should provide knowledge about reality that has high variance about the problem it describes (e.g. a feature that is high when it rains and low when it’s sunshine)

What makes them different?

amazon-sicherheit.kunden-ueberpruefung.xyz

eclipsehotels.com/language/en-GB/eng.exe

bohicacapital.com/page

summerweb.net

ad.getfond.info

vdula.czystykod.pl/rxdjna2.html

svision-online.de/mgfi/administrator/
components/com_babackup/classes/fx29id1.txt

URL WhiteList

gurufocus.com/stock/PNC

dvdtalk.ru/review

333cn.com/zx/zhxw.html

made-in-china.com/special/led-lighting

google.com/u/0/112261544981697332354/posts

youtube.com/watch?v=Qp8MQ4shN6U

unesco.org/themes/education-sustainable-developm

thisisthefirst.com/page/5

Malicious URL Detection Features (Literature)

1. **BlackList Features**: BlackLists suffer from a high false negative rate, but can still be useful as machine learning feature.
2. **Lexical Features**: Capture the property that malicious URLs tend to "look different" from benign URLs. Contextual information such as the length of the URL, number of digits, lengths of different parts, entropy of domain name.
3. **Host-based Features**: Properties of web site host. "Where" the site is hosted, "who" owns it and "how" it is managed. API queries are needed (WHOIS, DNS records). Examples: Date of registration, the geolocations, autonomous system (AS) number, connection speed or time-to-live (TTL).
4. **Content-based Features**: Less commonly used feature as it requires execution of web-page. Can be not only be not safe, but also increases the computational cost. Examples: HTML or JavaScript based.

Sahoo et al. 2017: Malicious URL Detection using Machine Learning: A Survey

Preparation In Class Exercise
ML Feature Engineering

Lexical Features

1. Length of URL
2. Length of domain
3. Count of digits
4. Entropy of domain
5. Position (or index) of the first digit
6. **Bag-of-words** for tld, domain and path parts of the URL

Host-based Features

1. Time delta between today's date and creation date
2. Check if it is an IP address

Bag-of-Words

- Bag-of-words model: (Frequency of) occurrence of each word is used as a feature
- Sklearn's **`CountVectorizer`**: Convert a collection of text documents to a matrix of token counts

Simple Example: Imposing a vocabulary of `top_tlds=['.com', '.de', '.uk']`

```
CountVectorizer_tlds = CountVectorizer(analyzer='word', vocabulary=top_tlds)
CountVectorizer_tlds = CountVectorizer_tlds.fit(tlds)
matrix_tlds = CountVectorizer_tlds.transform(tlds)
```

Bag-of-words model fitting

URL string
…google.ru…
…facebook.com…
…google.de…

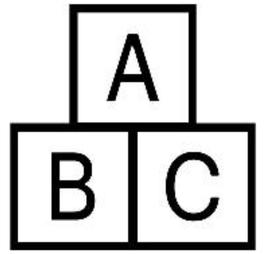
'.com'	'.de'	'.uk'
0	0	0
1	0	0
0	1	0

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

Preprocessing - an Art Work!

- Imputing missing values
- Scaling/Normalization
- One-Hot Encoding (Encoding categorical features)
- Embedding (e.g. word2vec)
- Binarizing (e.g. needed for Deep Learning multi-class target vector encoding)
- Encoding strings as int
- Dimensionality Reduction (e.g. PCA)
- Augmentation (e.g. tild/zoom images)
- Feature selection based on classifier
- Variance threshold

Data Types



Primary Python libraries: **`pandas, sklearn, scipy`**

http://scikit-learn.org/stable/modules/preprocessing.html

Imputing Missing Values

```
# using the most_frequent value
df.src_bytes = df.src_bytes.fillna
(df.src_bytes.value_counts().index[0])

# using the mean value
df.dst_bytes = df.dst_bytes.fillna(df.dst_bytes.mean())
```

Categorical Data

One Hot Encoding

Color
Red
Red
Blue
Green
Yellow
Red

One Hot Encoding

4 Categories

4 Columns with 1 when Category is True and delete original column!

Color
Red
Red
Blue
Green
Yellow
Red

Color_Red	Color_Blue	Color_Yellow	Color_Green
1	0	0	0
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0
1	0	0	0

One Hot Encoding

```
colors = ['Red', 'Red', 'Blue', 'Green', 'Yellow', ‘Red']
series_data = pd.Series( colors )
pd.get_dummies( series_data )

# df scenario
df=pd.get_dummies(df, prefix=None, prefix_sep='_',
dummy_na=False, columns=['protocol_type','flag'],
sparse=False)
```

Encoding strings as int

```
PROBE = ['portsweep.', 'satan.', 'nmap.', 'ipsweep.']
df = df.replace(to_replace = PROBE, value=1)
```

Feature Scaling

When you're creating a scaling object, you should first "**fit**" it to the **training data**, then **transform** both the **training and testing data** using the "fit" scaler.

If you try to fit the training and testing data separately, you will get inaccurate results.

Standard Scaling

$$zscore(x_i) = \frac{x_i - \mu}{\sigma}$$

```
scaled_feature = (feature - column_mean) / standard_deviation
```

Standard Scaling

$$zscore(x_i) = \frac{x_i - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

or
X_scaled = scaler.fit_transform(X)
```

Standard Scaling

```
original values:
 [[  0.9   0.1  40. ]
 [  0.3   0.2  50. ]
 [  0.6   0.8  60. ]]

Mean of each column:
 [  0.6  0.3667  50.]

SD of each column:
 [ 0.2449  0.3091  8.165 ]
```

```
scaled values:
 [[ 1.2247 -0.8627 -1.2247]
 [-1.2247 -0.5392  0.    ]
 [ 0.      1.4018  1.2247]]

Means of scaled data, per column:
 [ 0. -0.  0.]

SD's of scaled data, per column:
 [ 1.  1.  1.]
```

Min/Max Scaling

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
normed_feature = (feature - col_min) / (col_max - col_min)
```

Min/Max Scaling

```
from sklearn.preprocessing import MinMaxScaler
minmax= MinMaxScaler()

minmax.fit(X)
X_scaled_minmax = minmax.transform(X)

or

X_scaled_minmax = minmax.fit_transform(X)
```

MinMax Scaling

```
original values:
 [[  0.9   0.1  40. ]
 [  0.3   0.2  50. ]
 [  0.6   0.8  60. ]]

Mean of each column:
 [  0.6  0.3667  50.]

SD of each column:
 [ 0.2449  0.3091  8.165 ]
```

```
scaled values:
 [[ 1.      0.      0.    ]
 [ 0.      0.1429  0.5   ]
 [ 0.5     1.      1.    ]]
Means of scaled data, per column:
[ 0.5    0.381  0.5  ]

SD's of scaled data, per column:
[ 0.4082  0.4416  0.4082]
```

Selecting Features

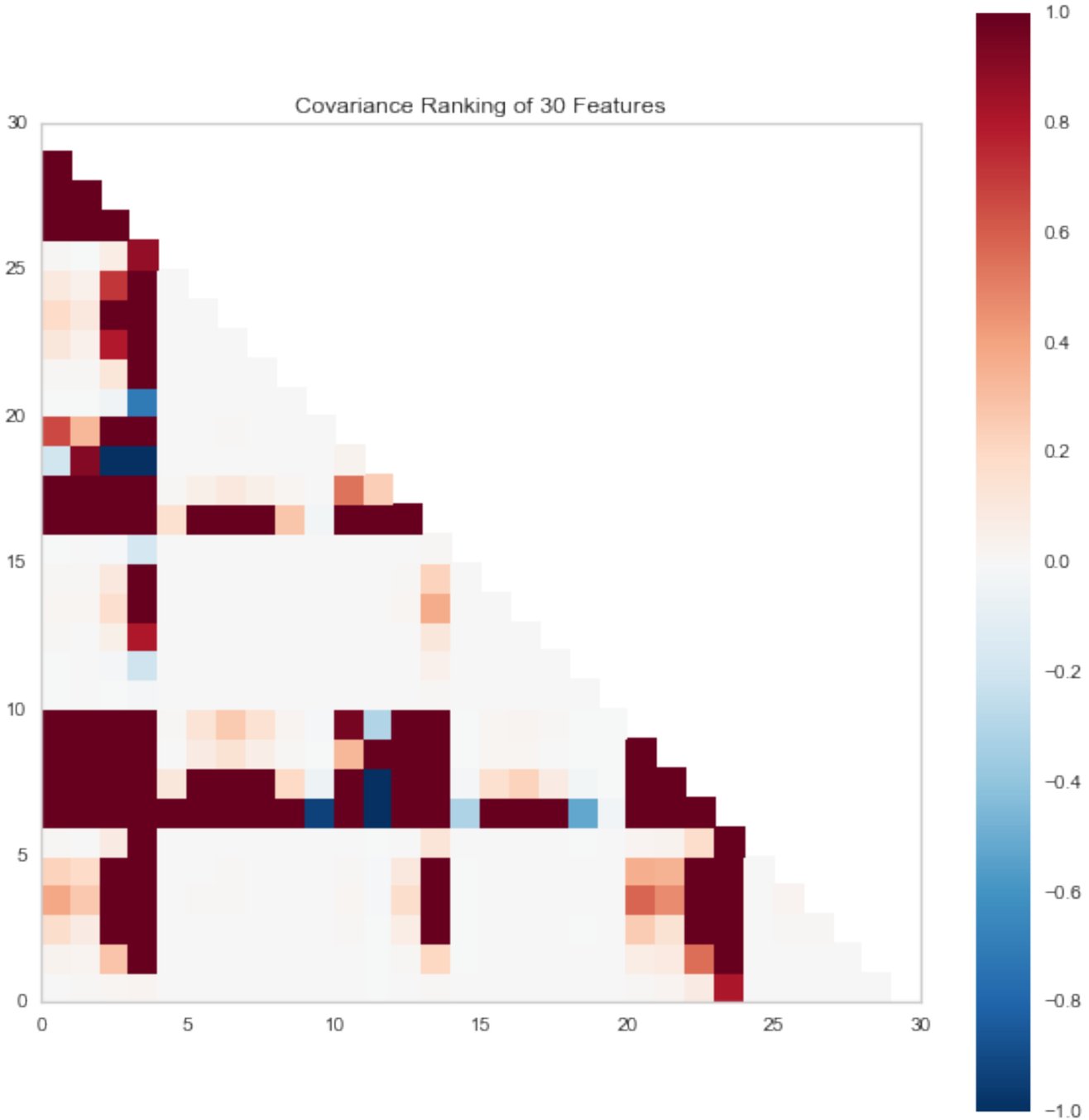
Should we use all of them?

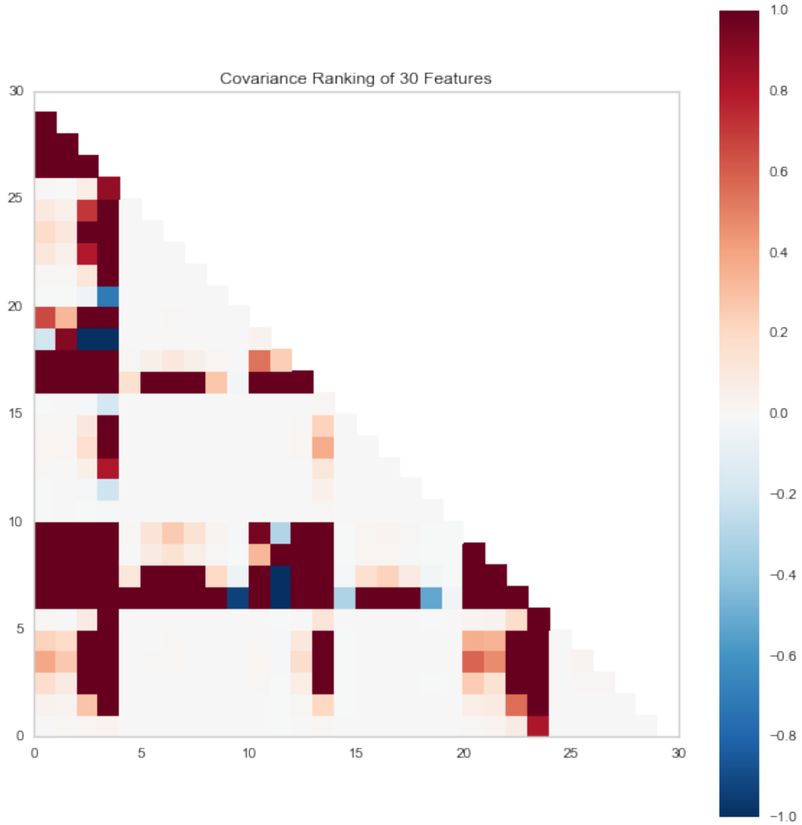
How do we know which features to use and which to discard?

How do we know which features to use and which to discard?

Visualize them!!

Introducing Yellowbrick


Covariance Ranking of 30 Features
30
25
20
15
10
5
0
0
5
10
15
20
25
30
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0




```python
import pandas as pd
import seaborn as sns
from yellowbrick.features.rankd import Rank2D

# Extract the numpy arrays from the data frame
X = df[features]
y = df.diagnosis

# Instantiate the visualizer with the Covariance ranking algorithm
visualizer = Rank2D(features=features, algorithm='covariance')

visualizer.fit(X, y)
visualizer.transform(X)
visualizer.poof()
```

Selects *k* features according to the highest score

```
best_features = SelectKBest( score_func=chi2,k=3).fit_transform(X,y)
```

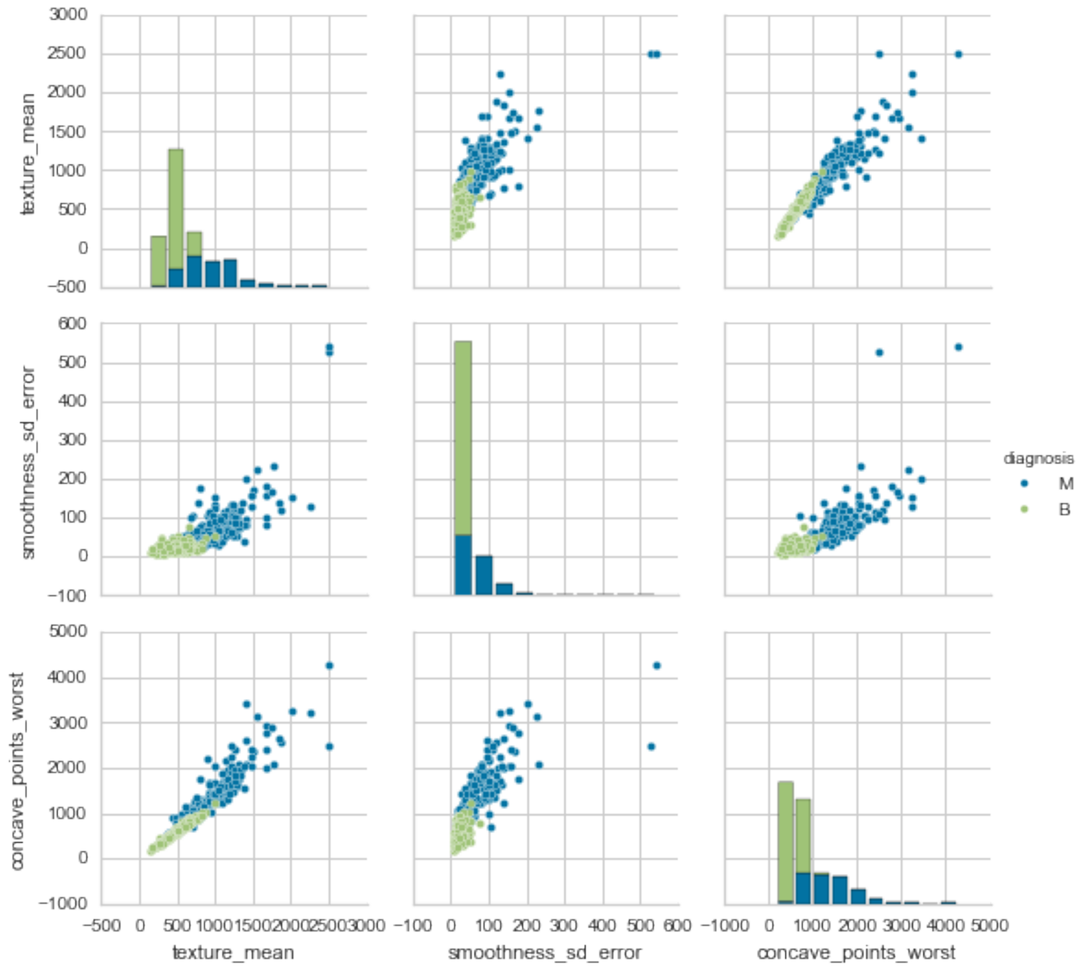
Selects all features above a given threshold in the scoring function

```
best_features = SelectPercentile( score_func=chi2,percentile=3).fit_transform(X,y)
```

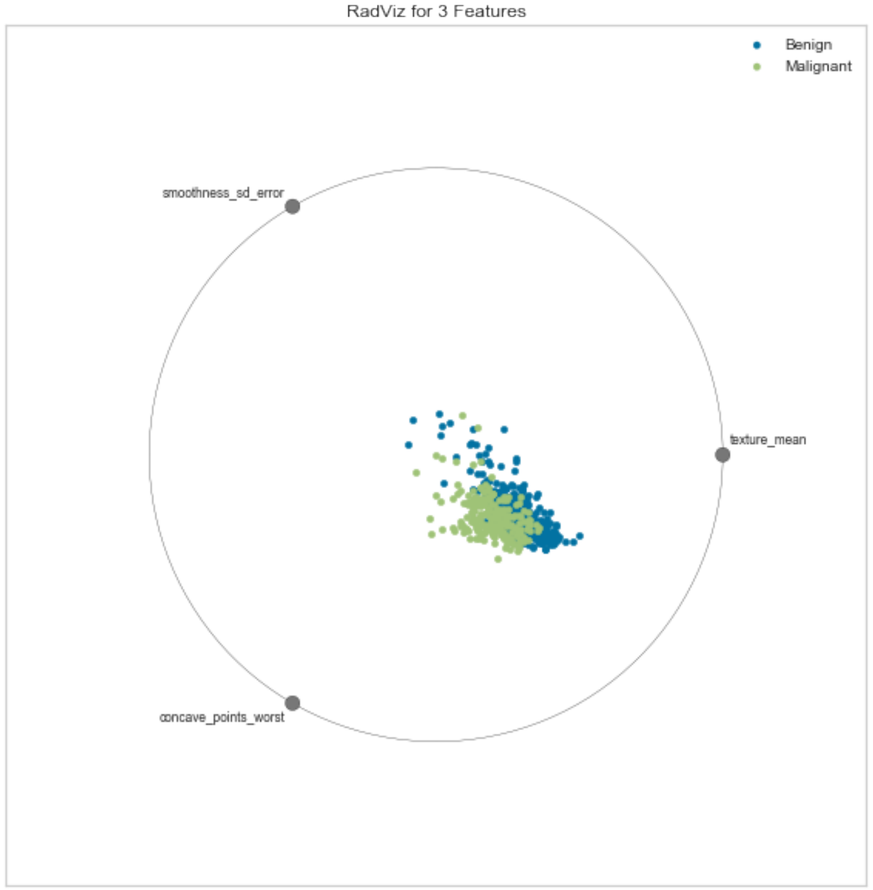
Available Scoring Functions:

- **For regression:** f_regression, mutual_info_regression
- **For classification:** chi2, f_classif, mutual_info_classif

References:
http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
http://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection




```
import seaborn as sns
sns.pairplot(<features>, hue='<target>' )
```




```
from yellowbrick.features.radviz import RadViz
…
visualizer = RadViz(classes=<target classes>, features = <features>)
visualizer.fit(X, y)
visualizer.transform(X)
visualizer.poof()
```

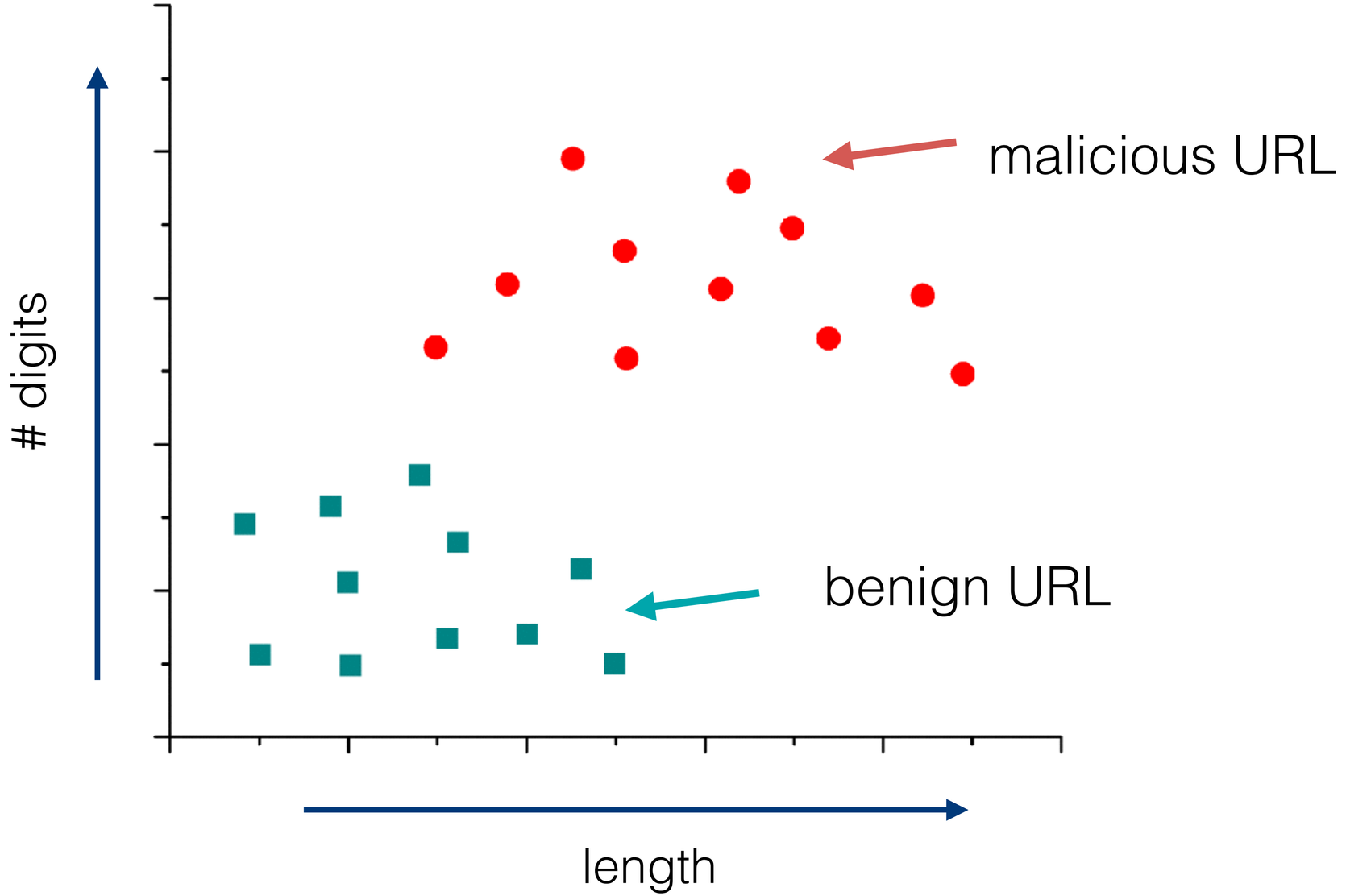
In Class Exercise

Please take 30-40 minutes and complete

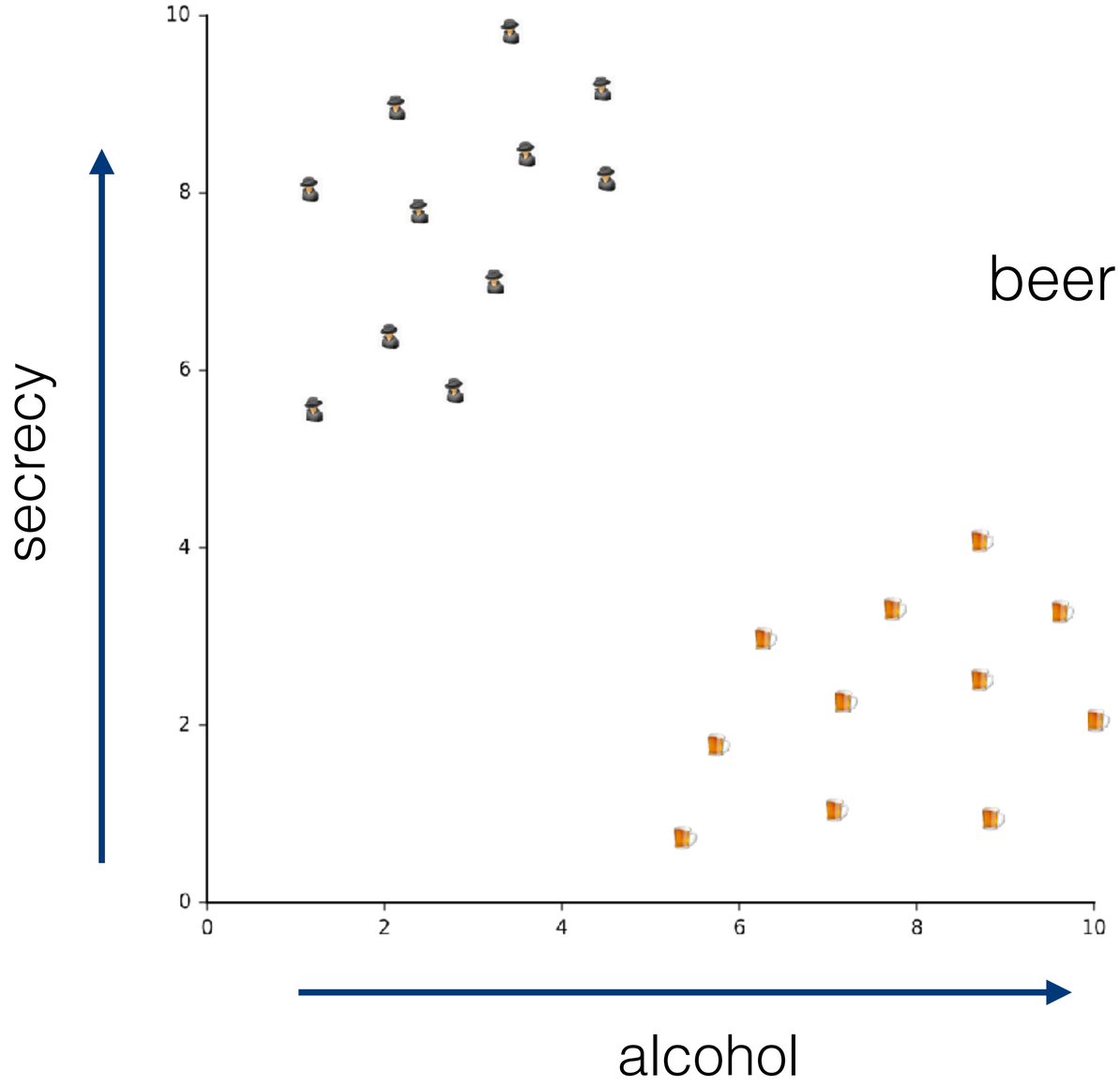
Day 2 - Feature Engineering

Math Free Overview of Supervised Machine Learning Algorithms

Features and Labels

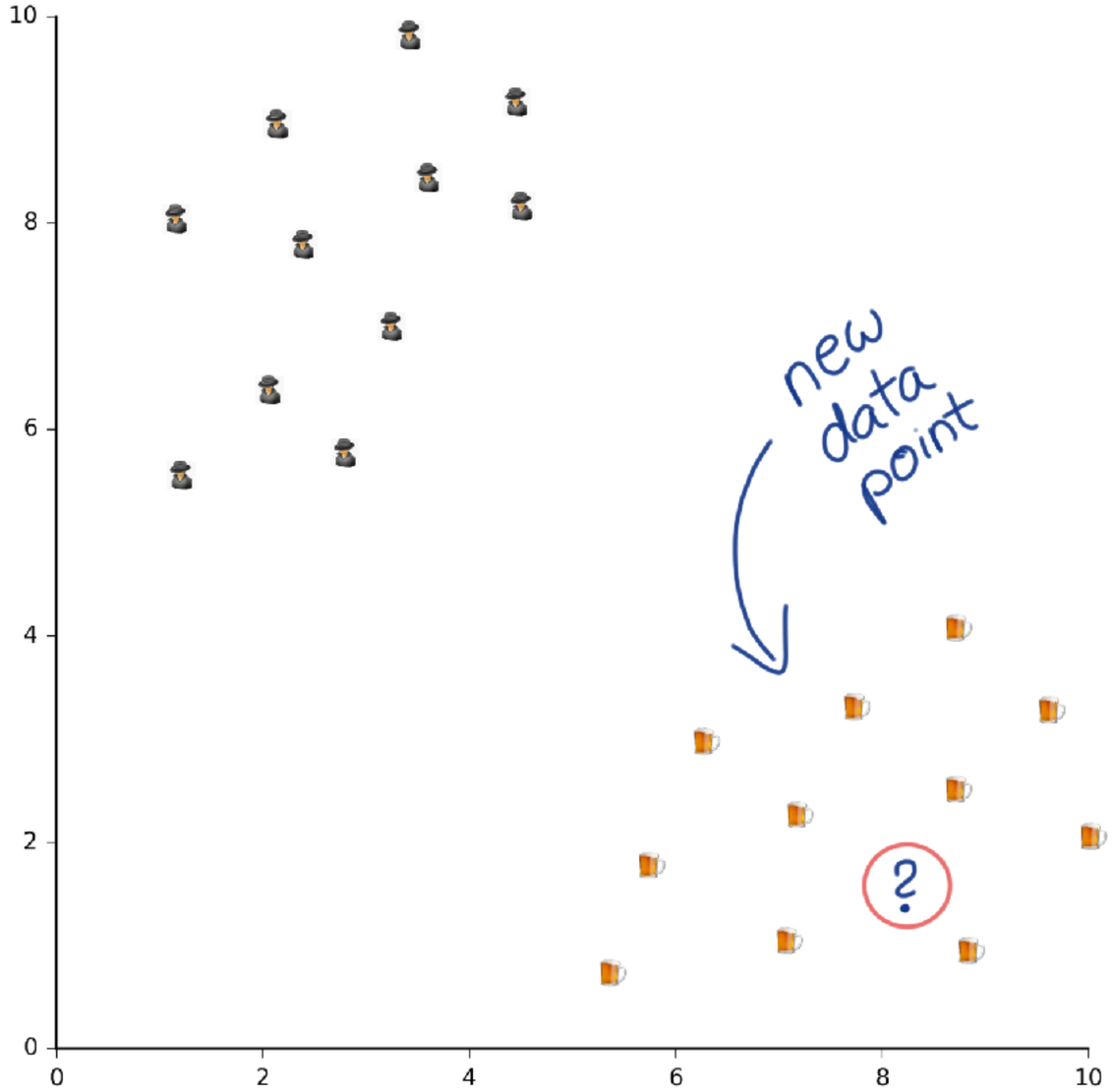


FUN Features and Labels

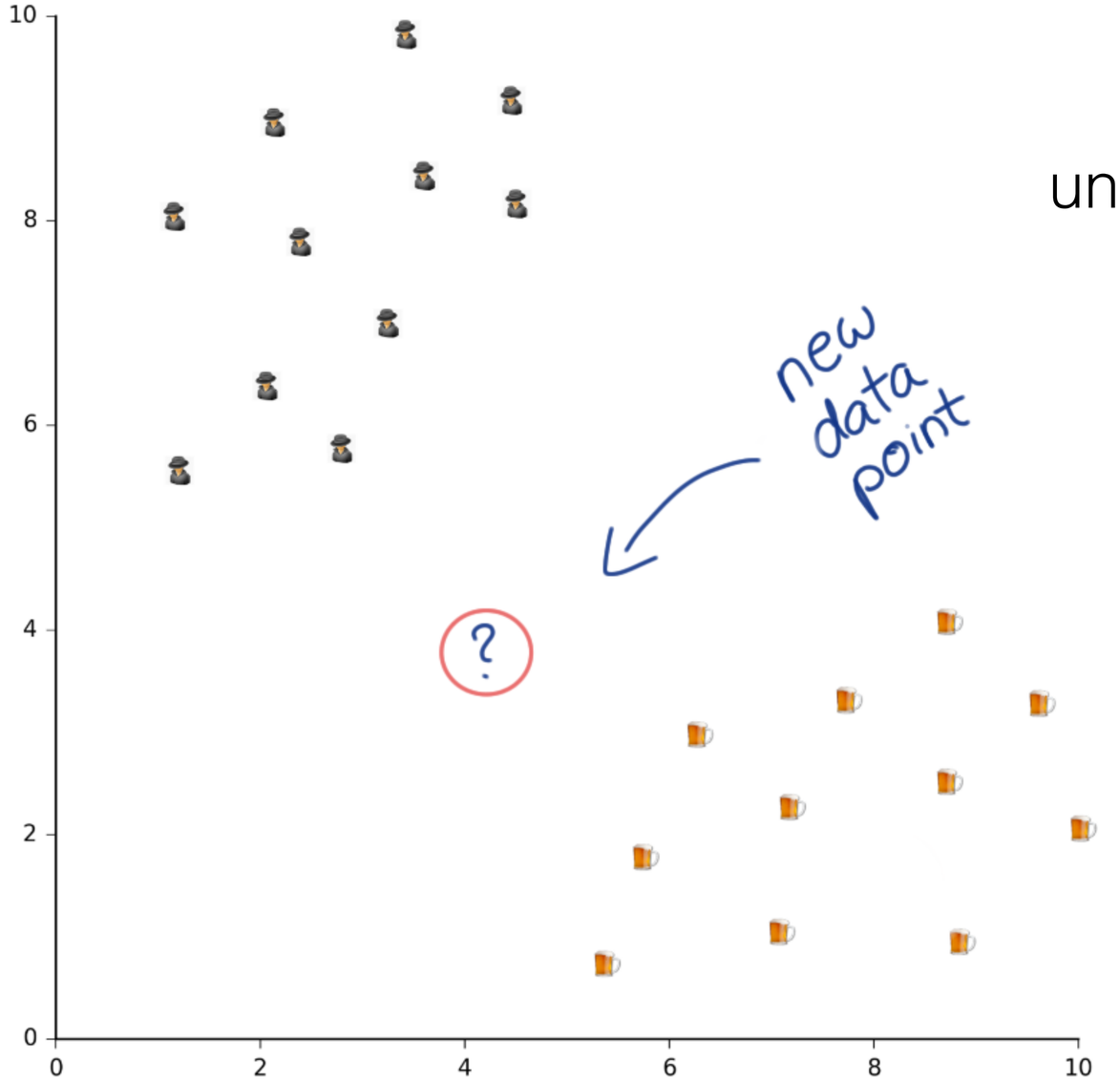


beer and cyber !!!

Making a new decision

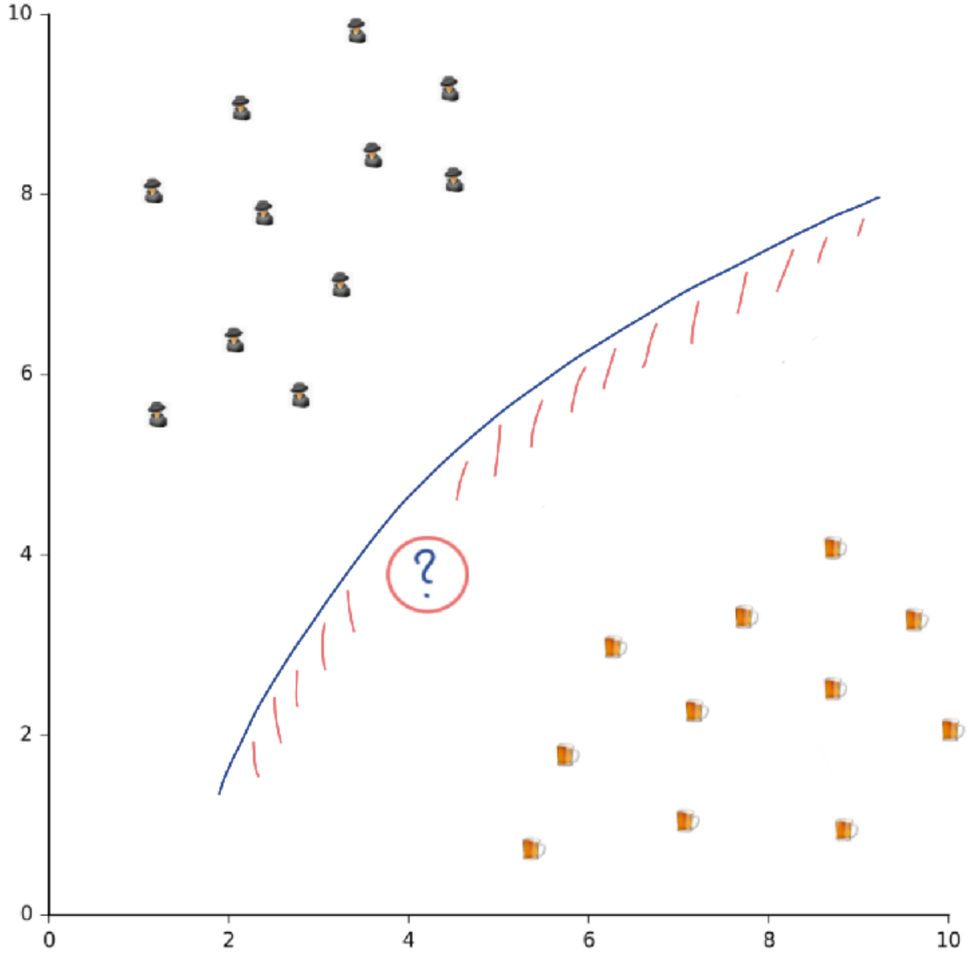


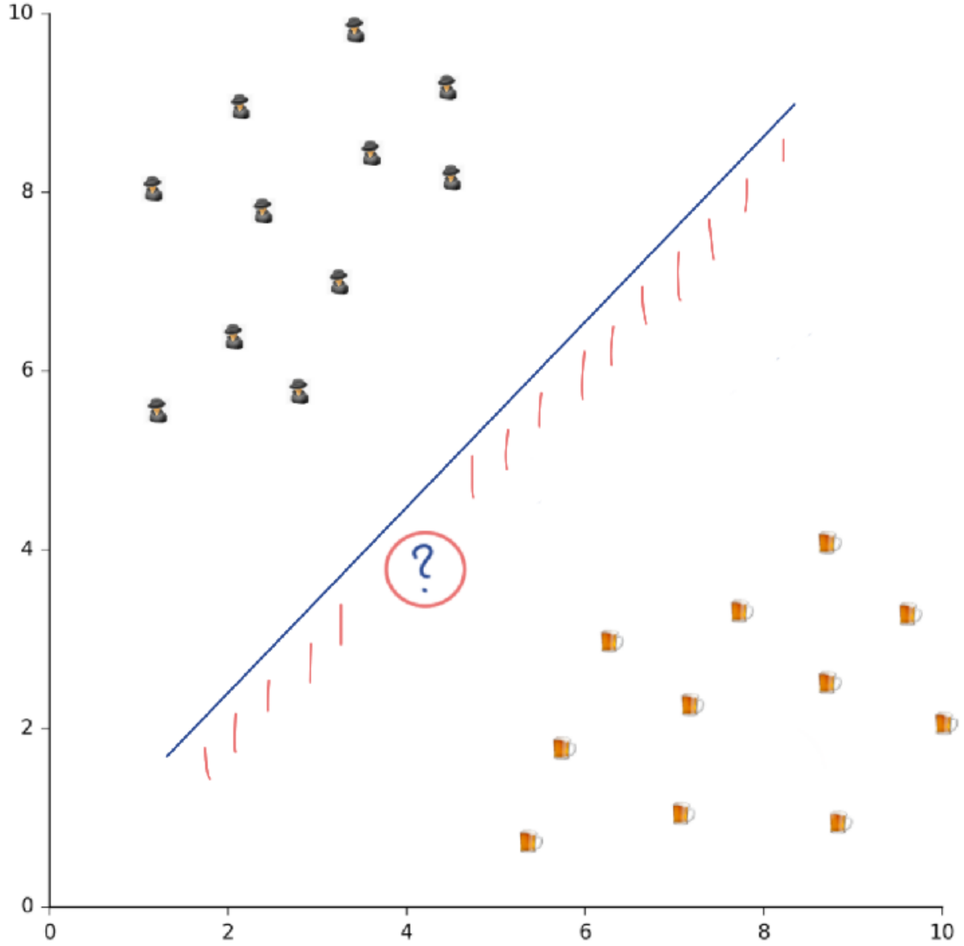
Making a new decision



unclear?

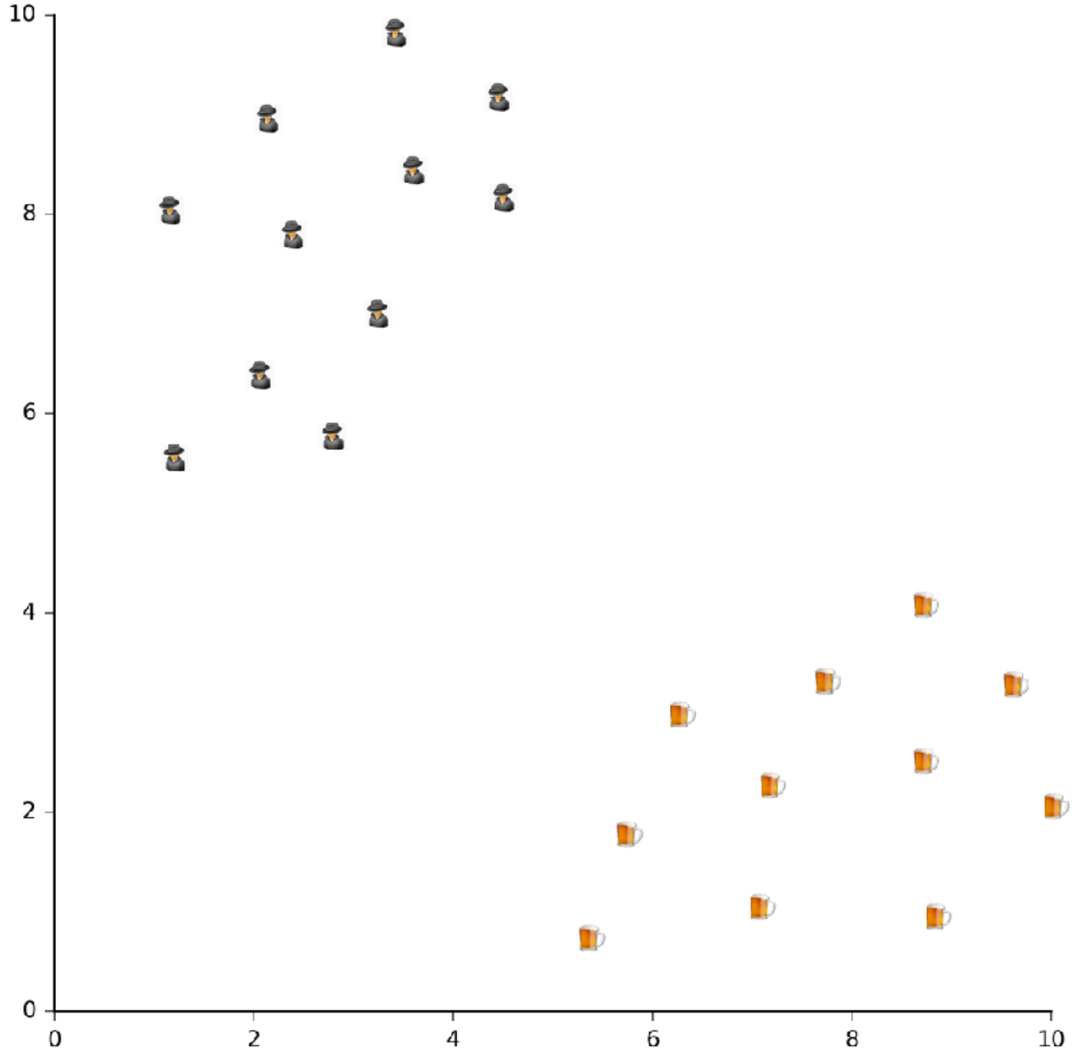
Decision surface concept



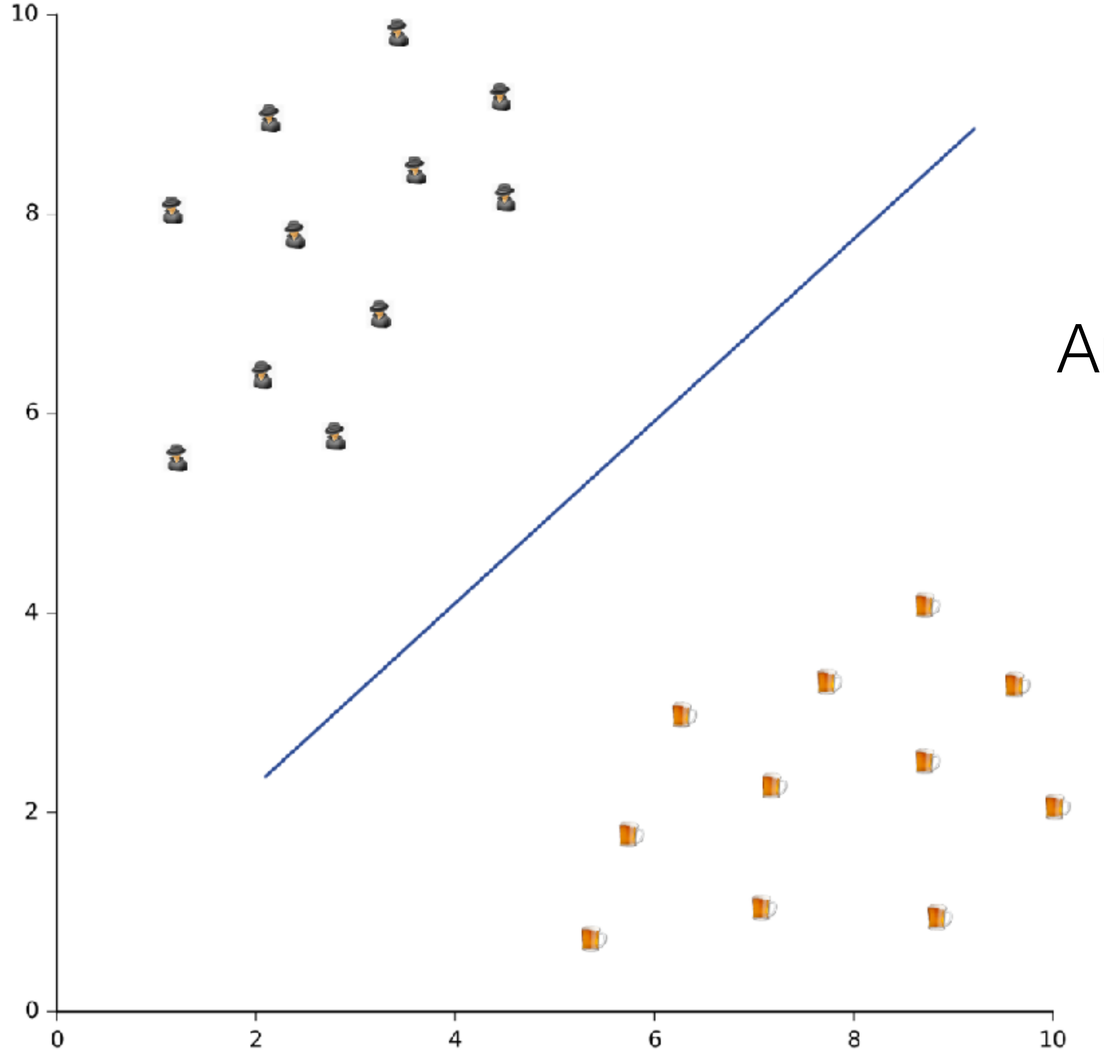


Support Vector Machines (SVM)

How can we **linearly** separate beer and cyber?

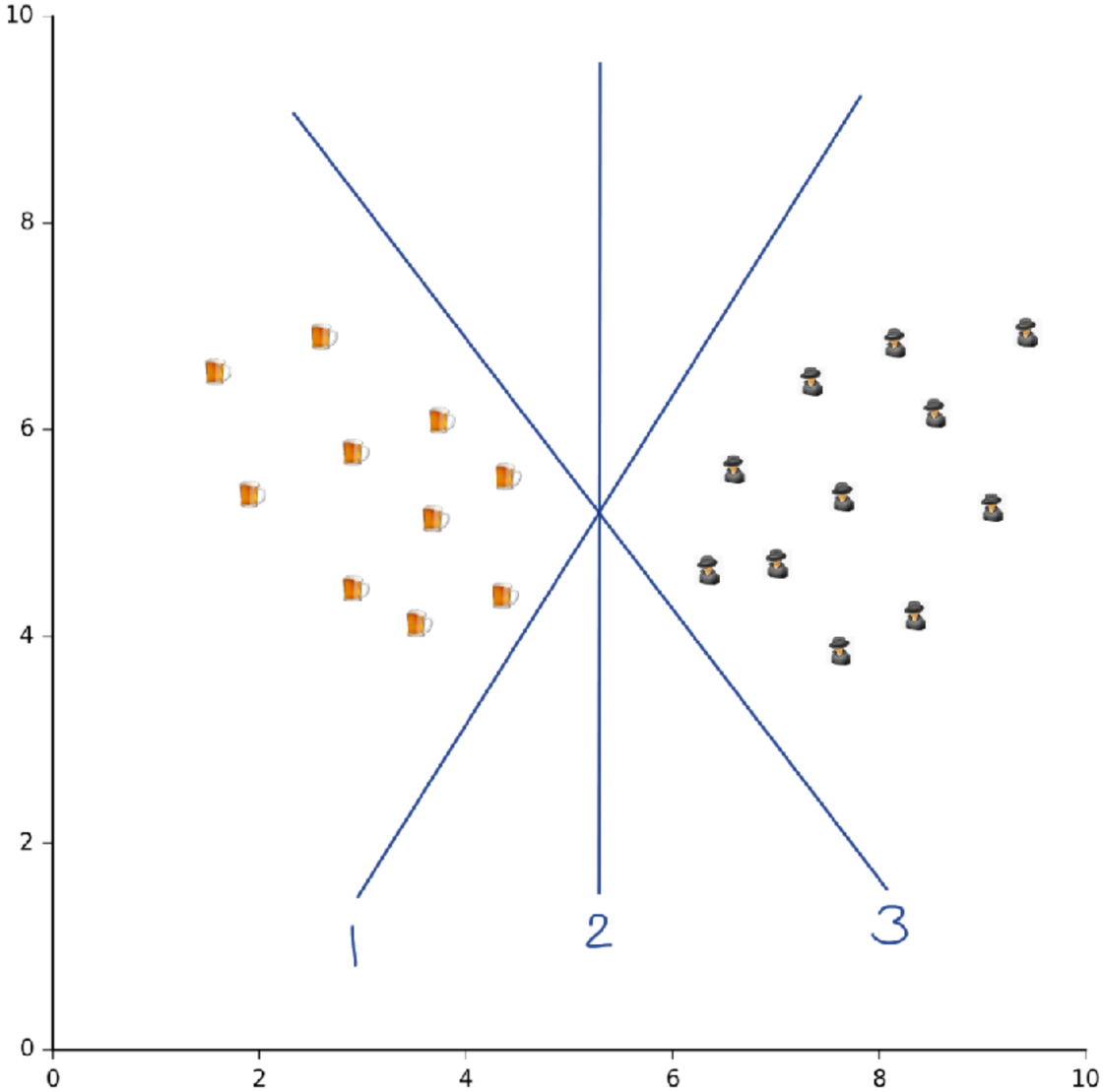


How can we **linearly** separate beer and cyber?

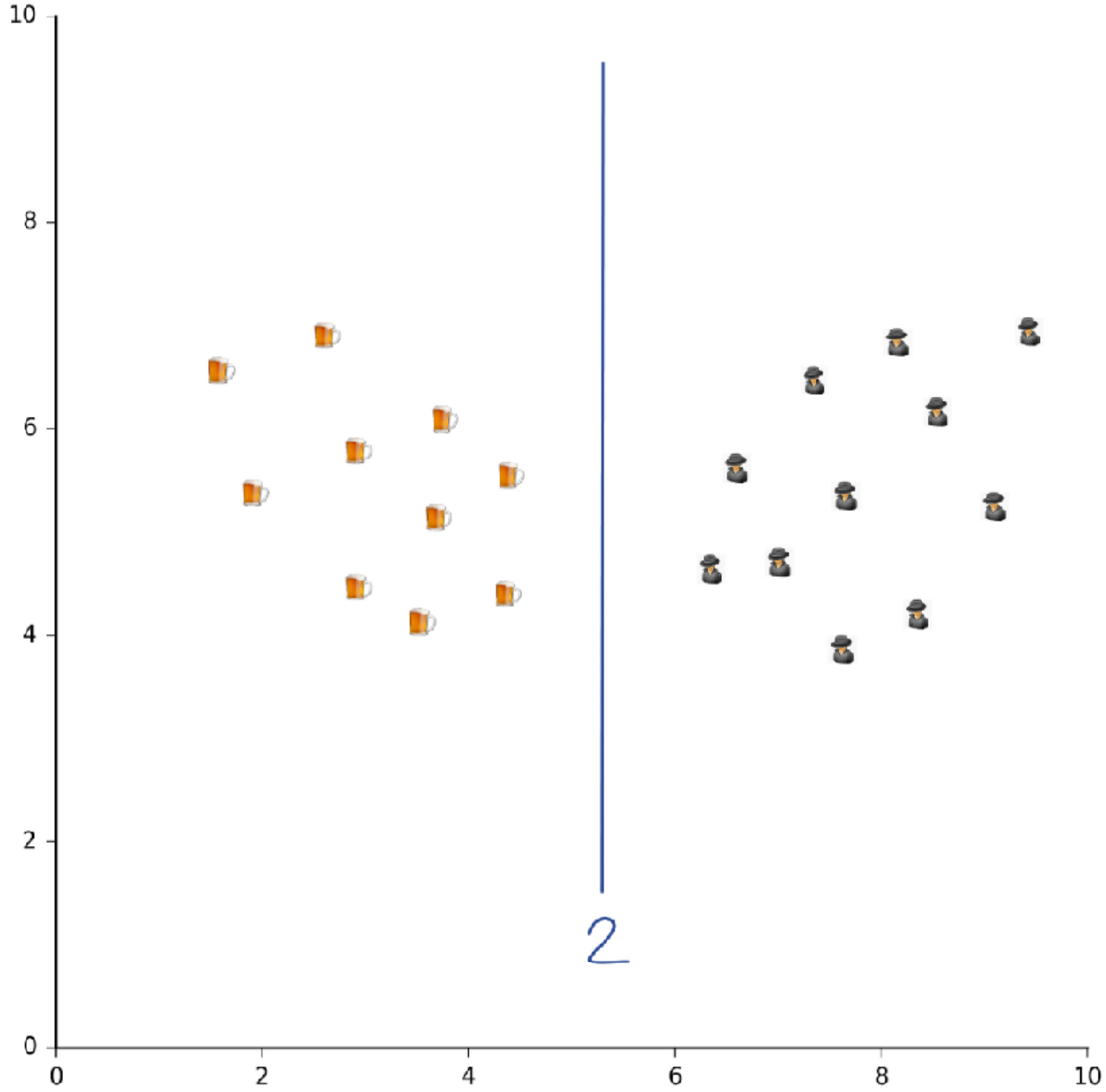


Answer: straight line

Same question, choose the best line!

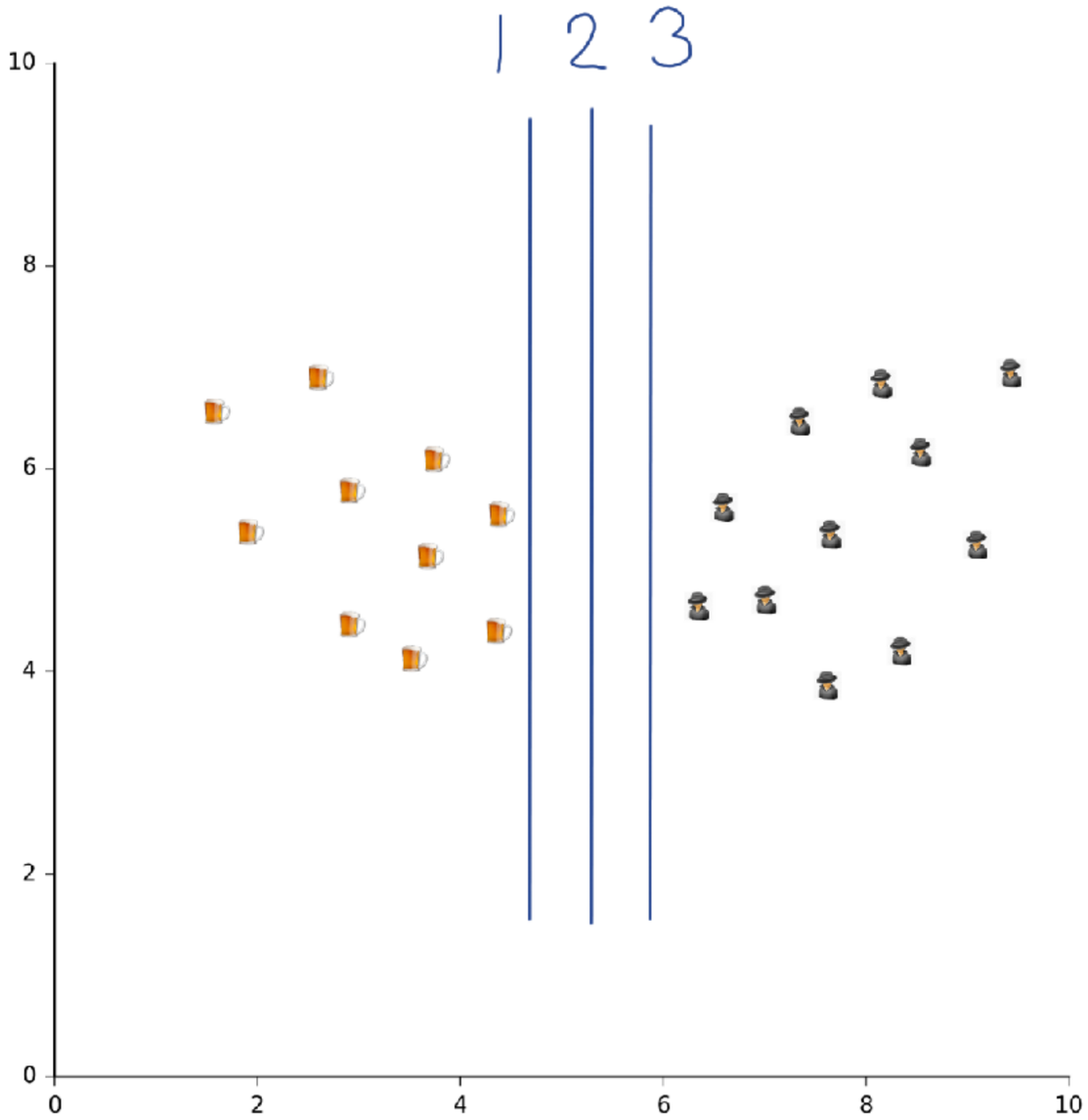


Same question, choose the best line!

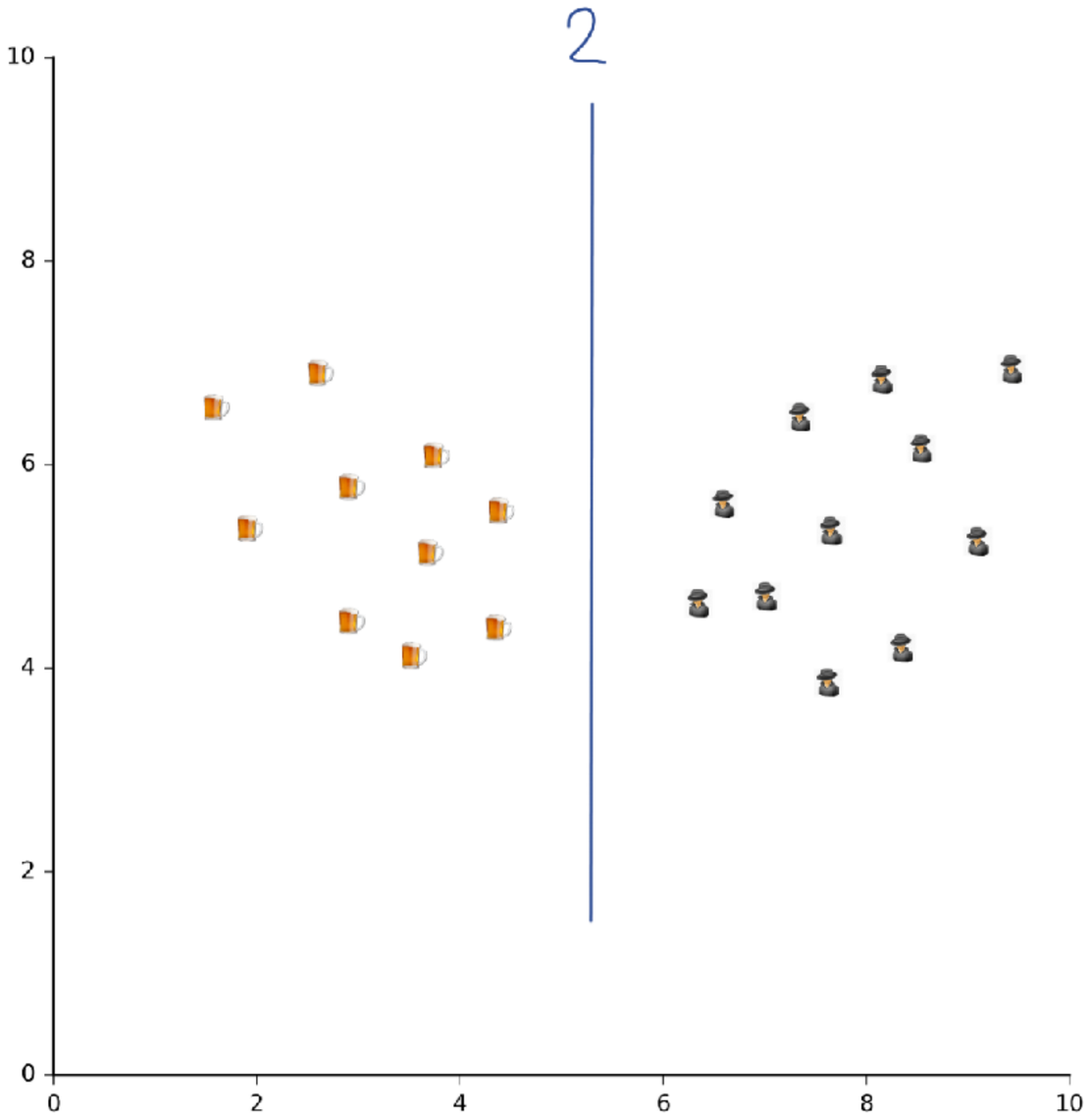


Answer: 2

How about now?

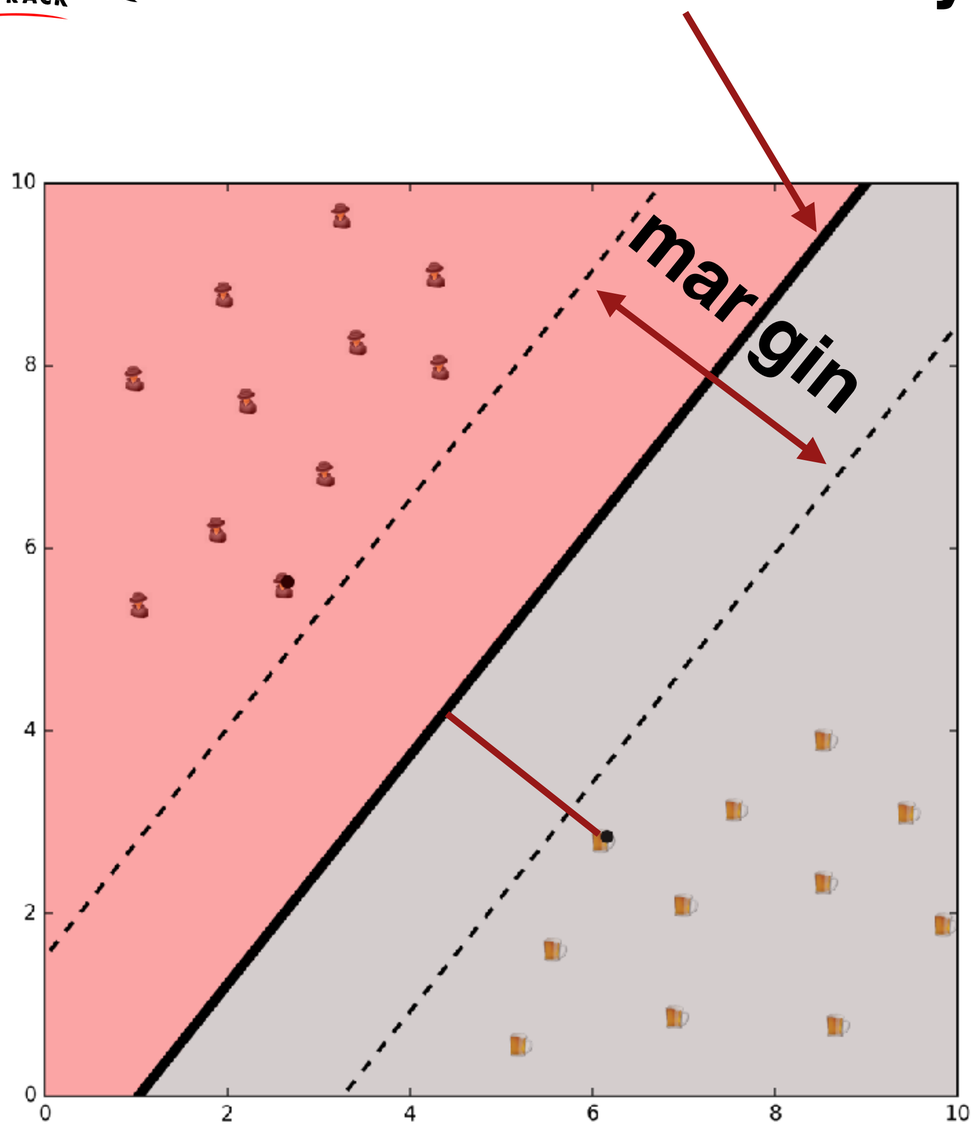


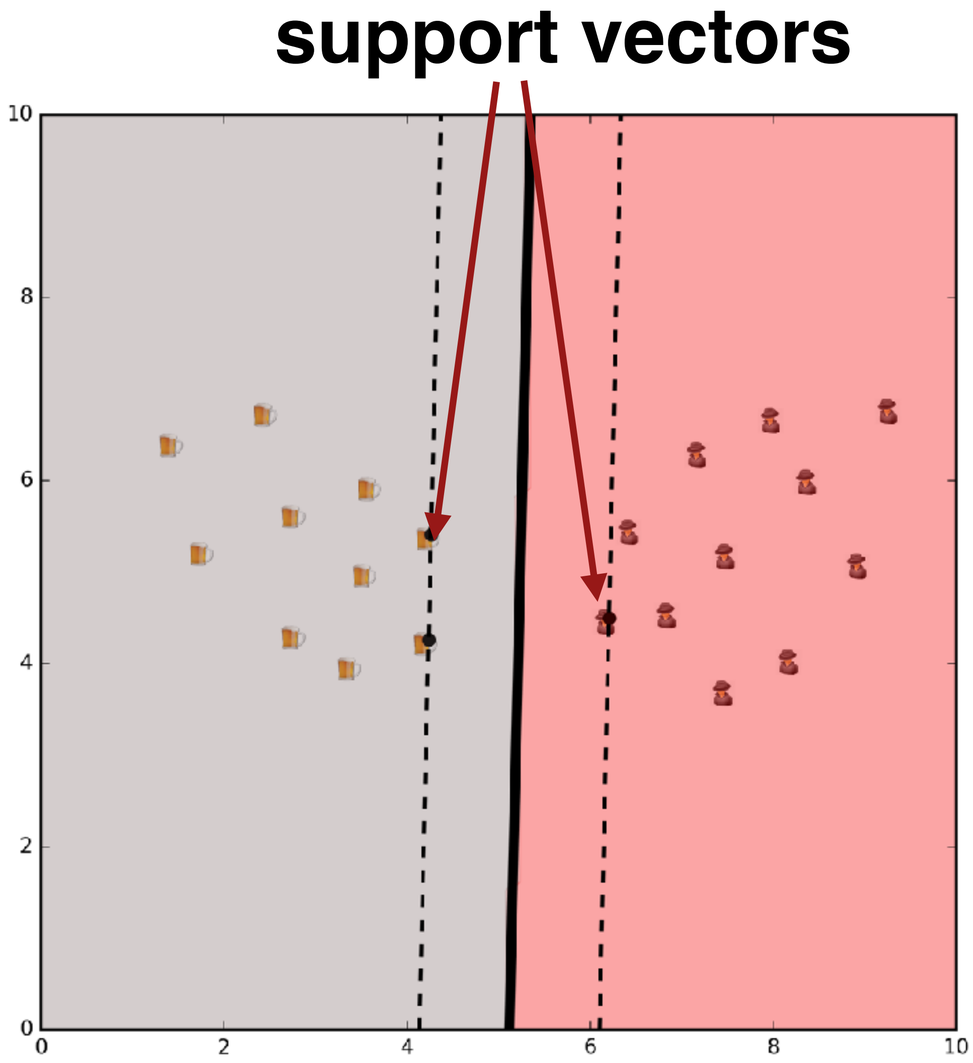
How about now?



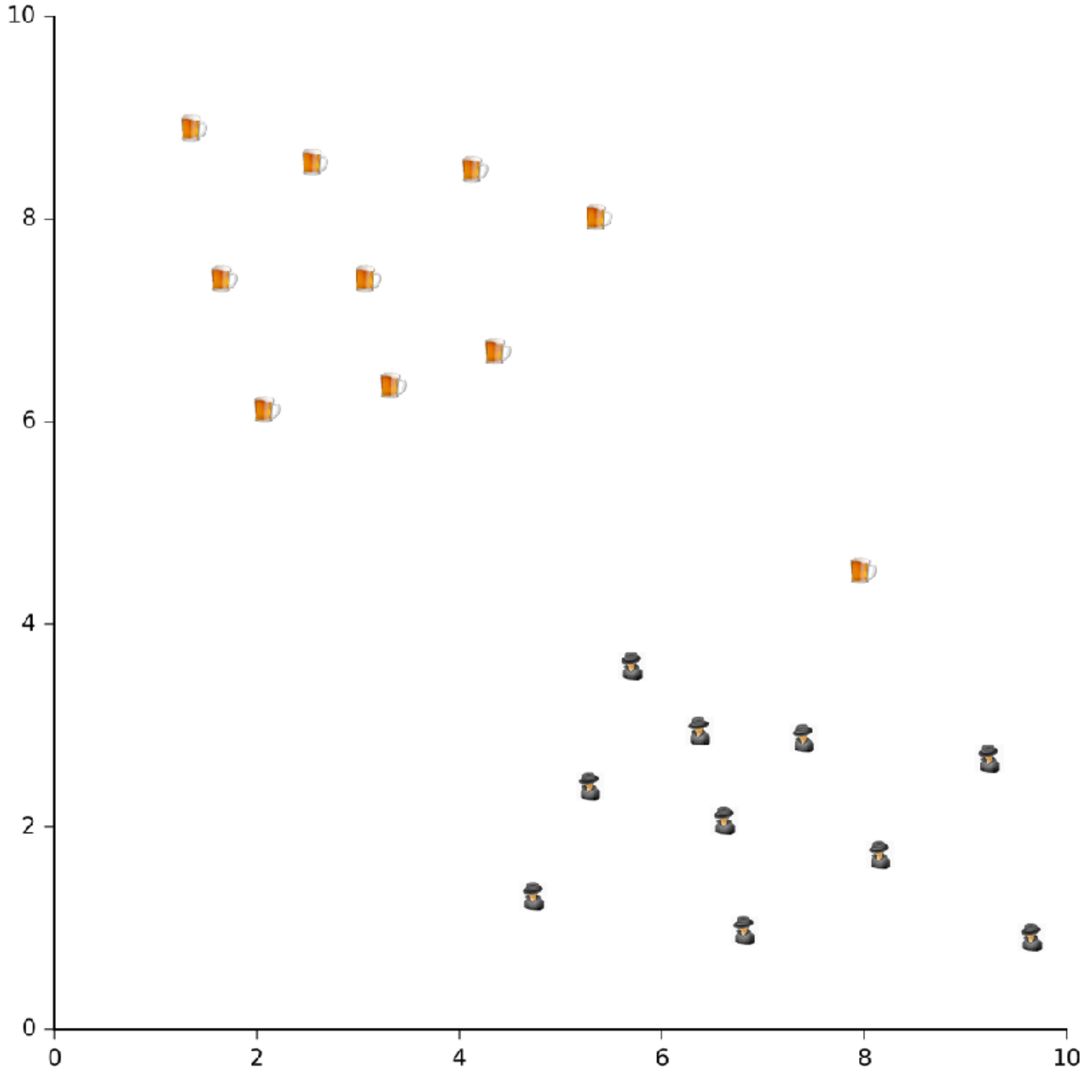
Answer: 2

Open BlackBox…

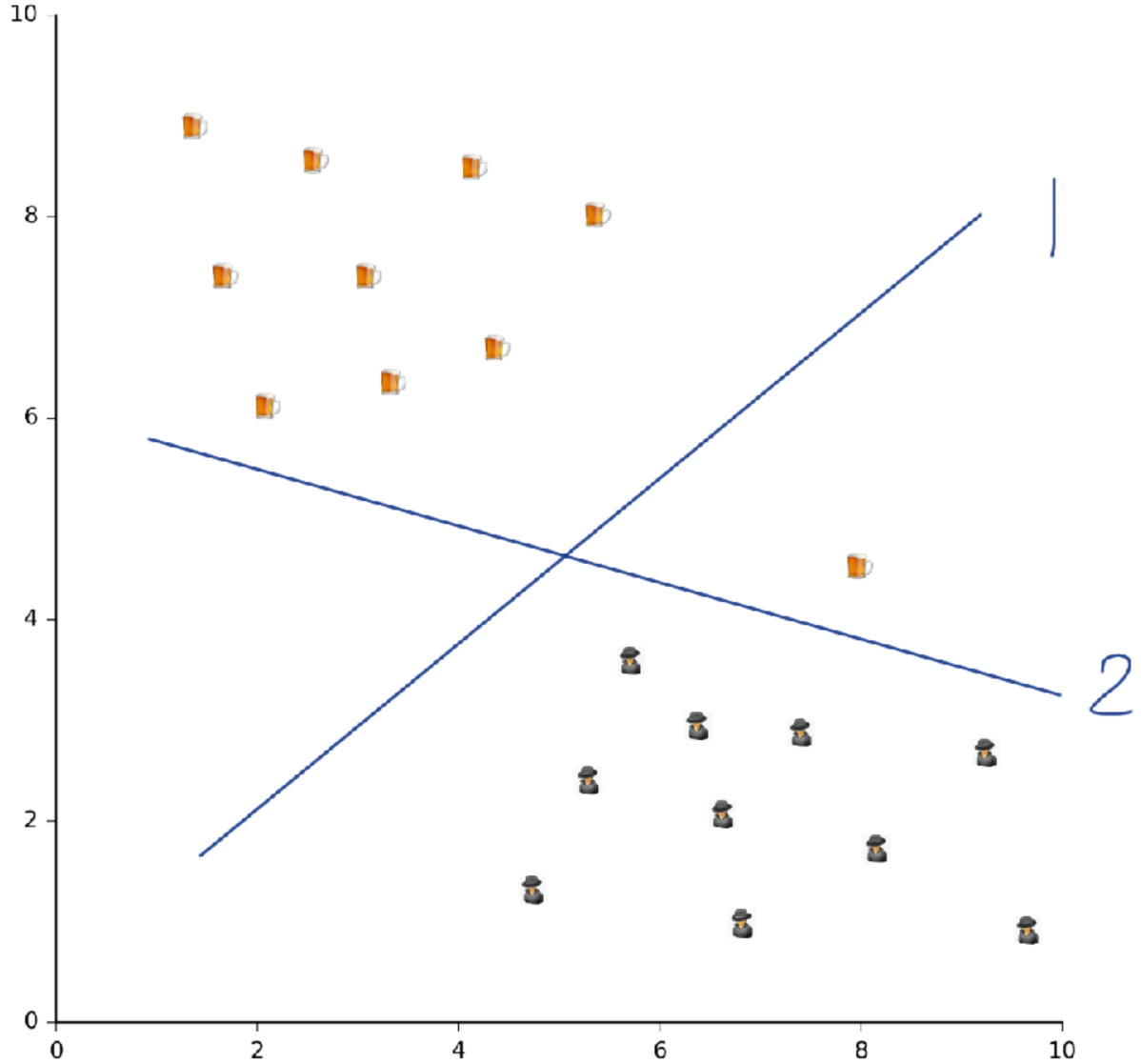




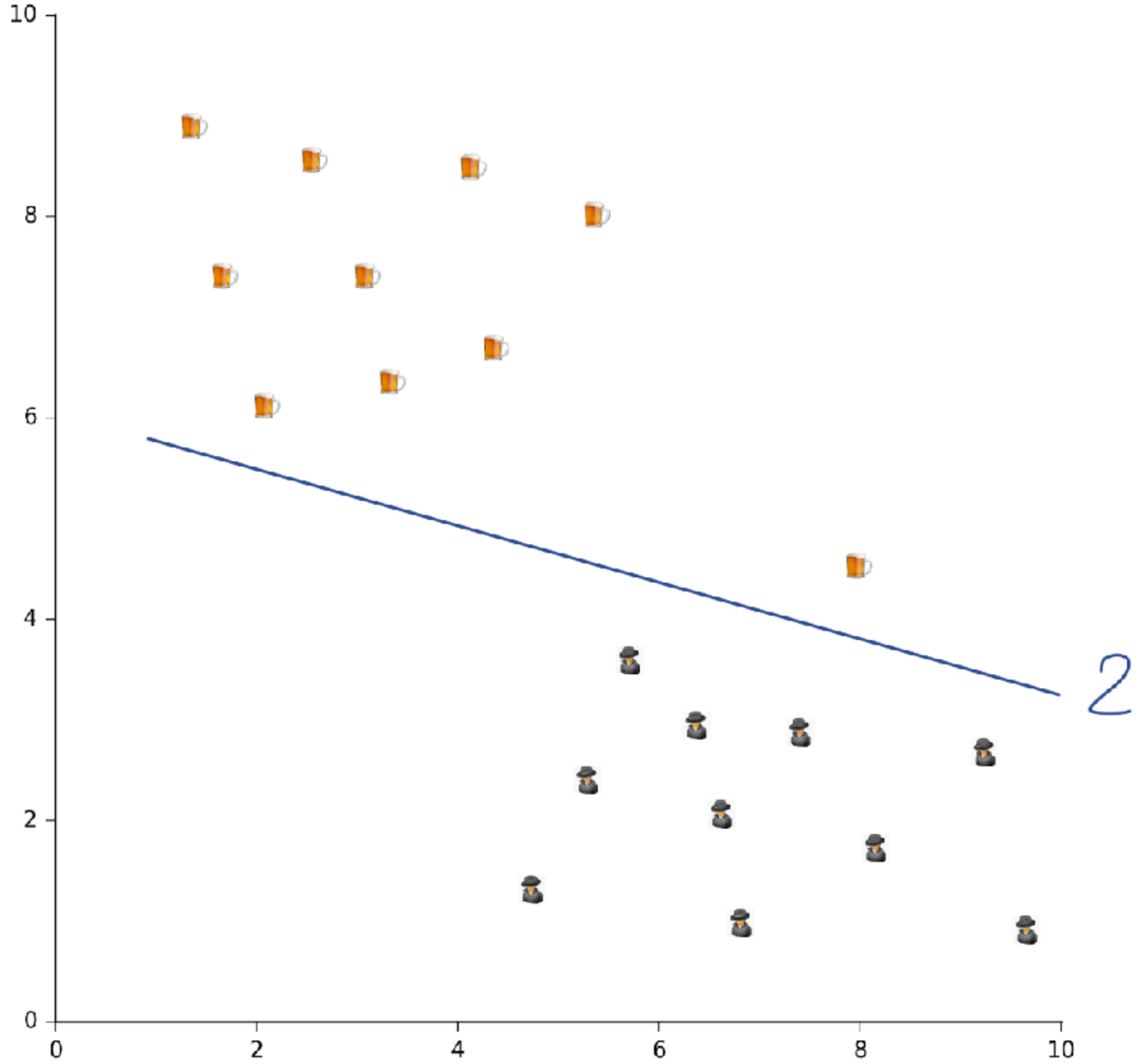
Another test, how do we best separate beer and cyber linearly?



Choose the best line!

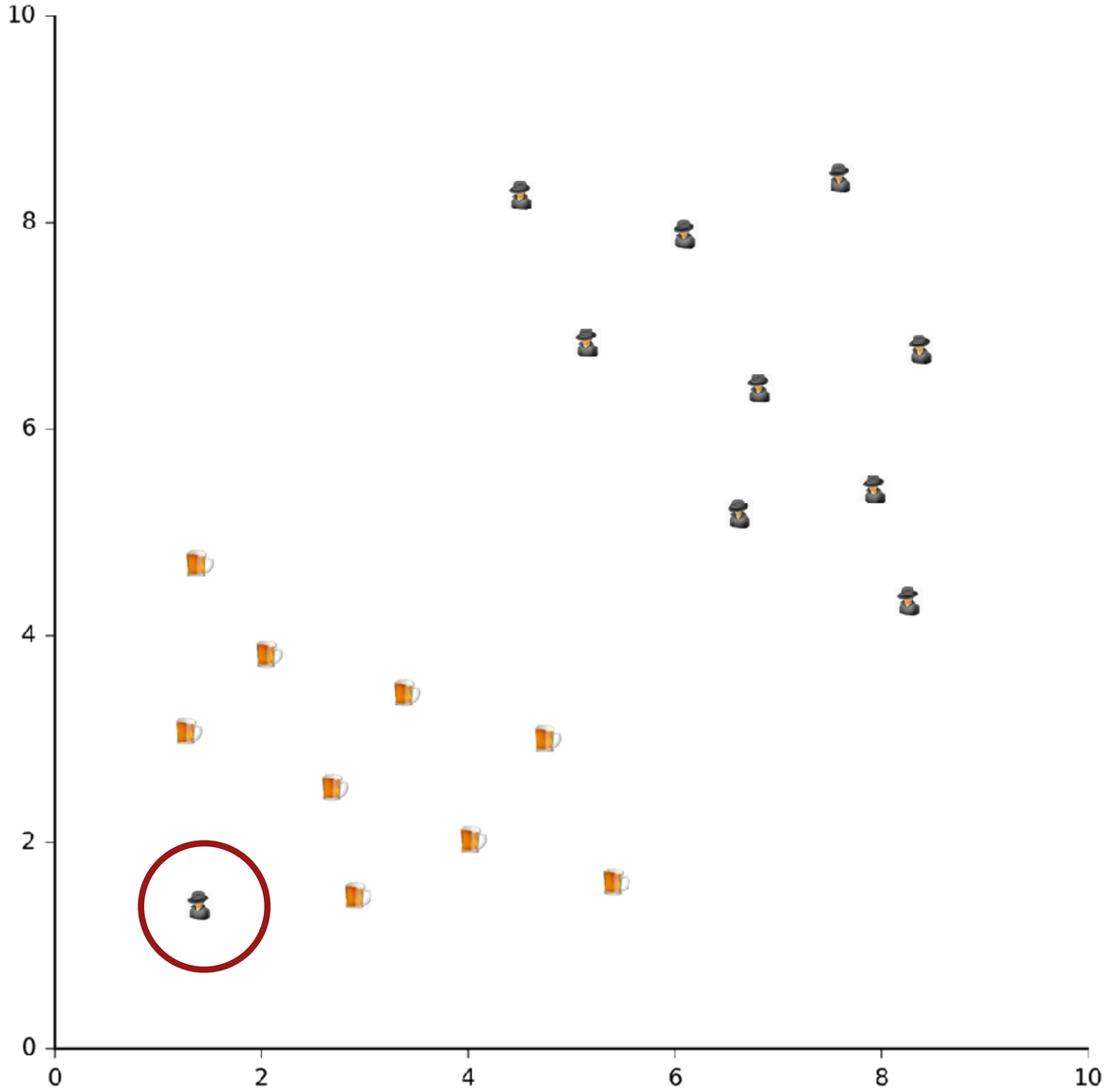


Choose the best line!

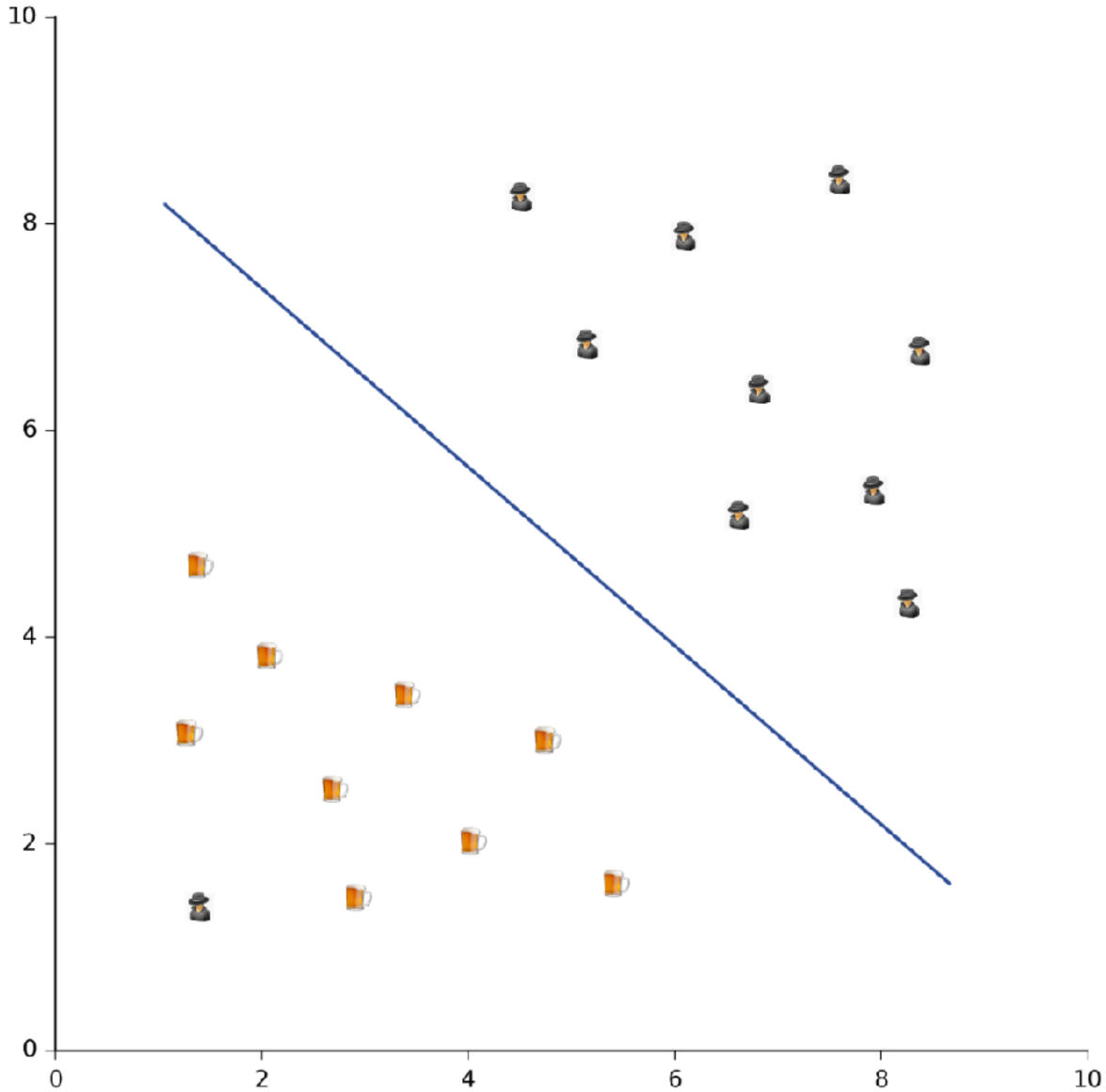


Answer: 2

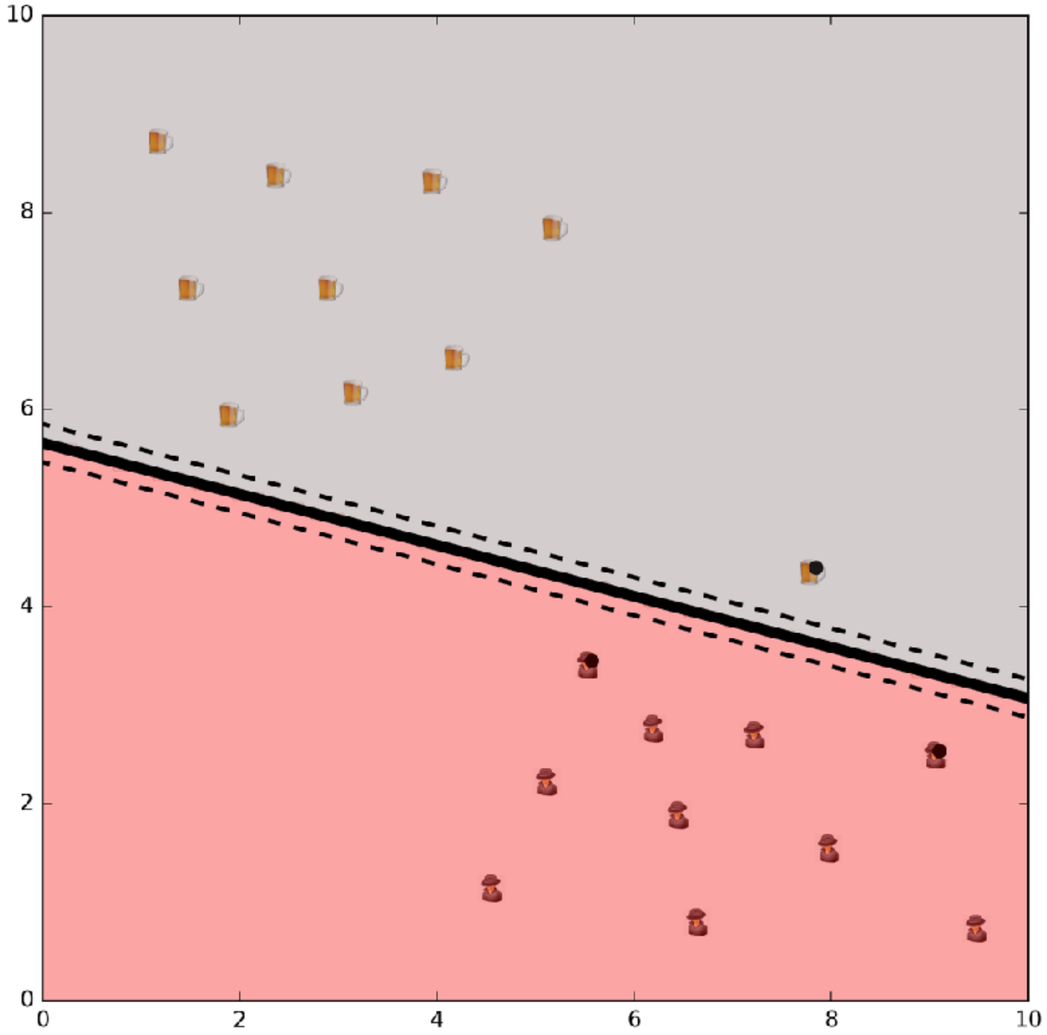
Wait and now?

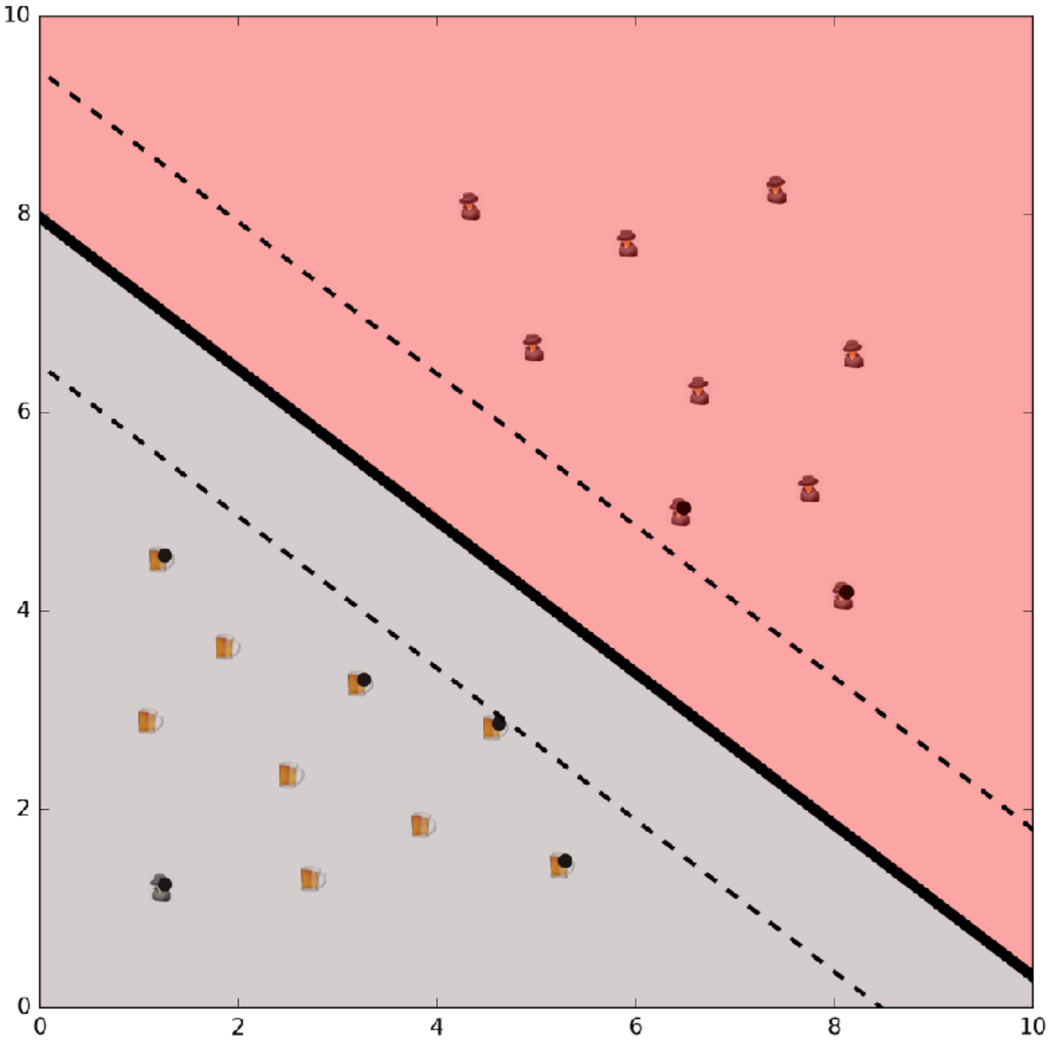


Just ignore?

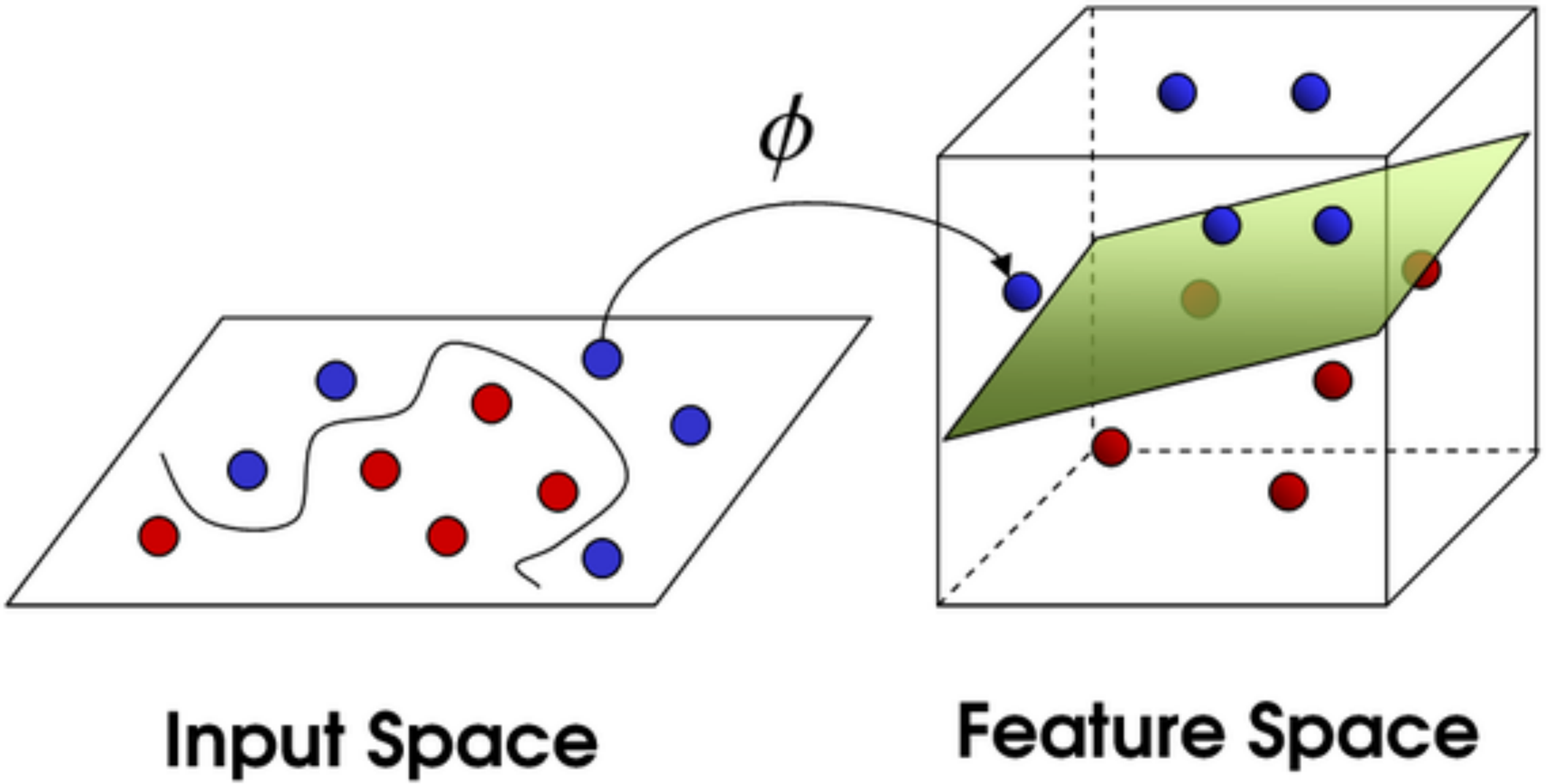


Open BlackBox…





Kernel trick



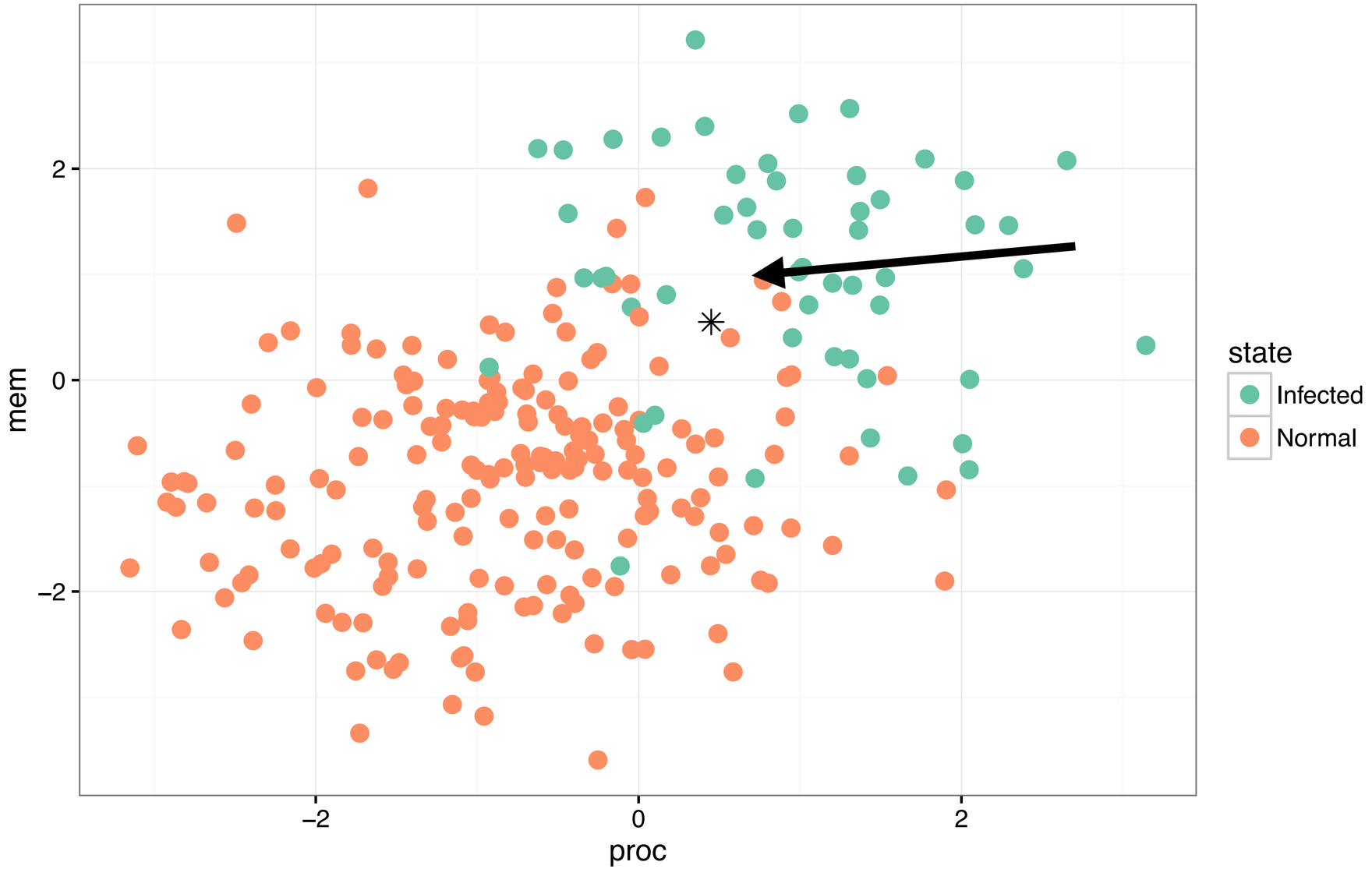
K-Nearest Neighbors Classifier

K-nearest neighbors

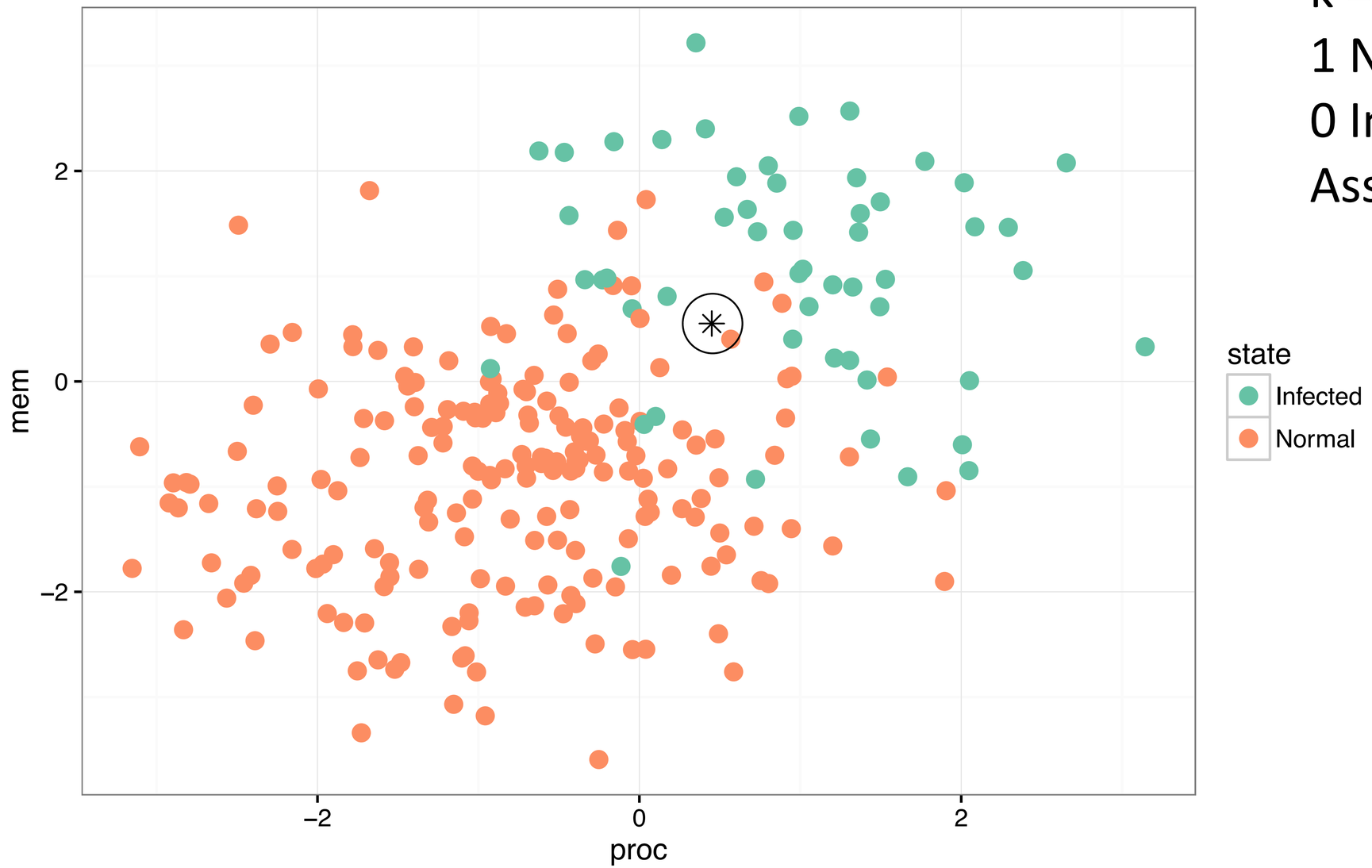
Define K, and for every unknown observation:

1. Find k-nearest neighbors (by *distance*)
2. Assign class based on dominate class
3. Optional weight by distance

K-nearest neighbors



K-nearest neighbors



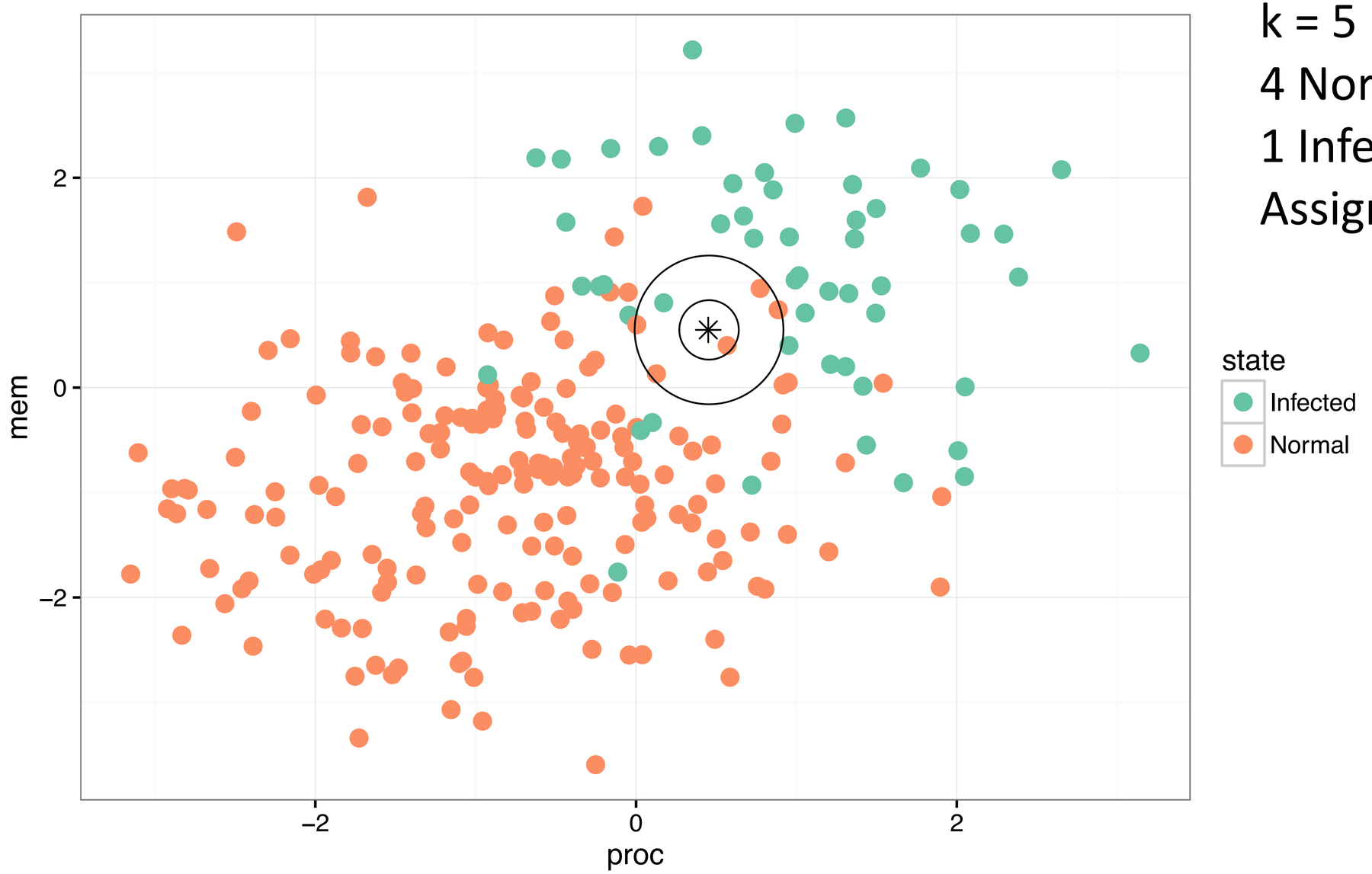
k = 1

1 Normal

0 Infected

Assign “Normal”

K-nearest neighbors



k = 5

4 Normal

1 Infected

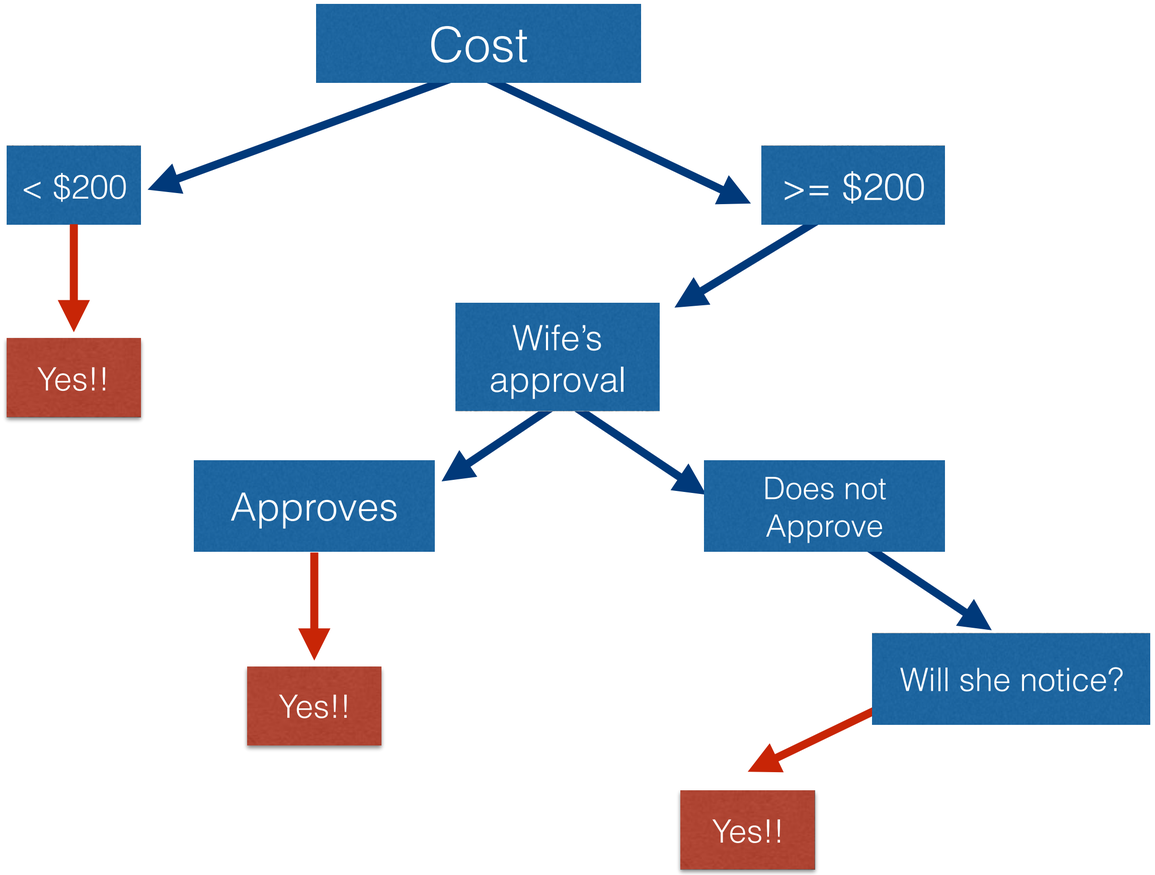
Assign “Normal”

K-nearest neighbors

Further reading: https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

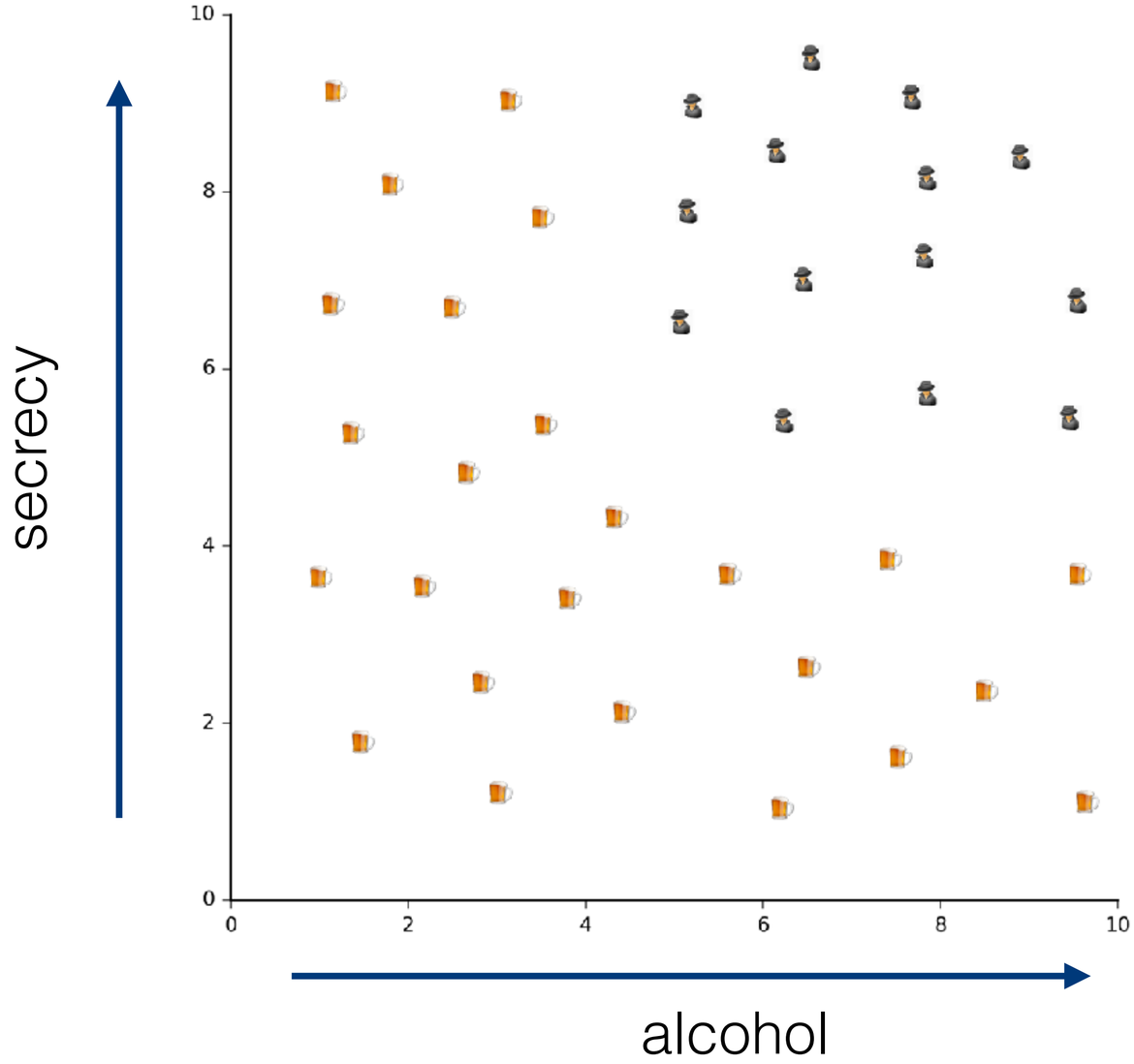
Simple Decision Tree (DT)

Should I buy a new tech gadget?

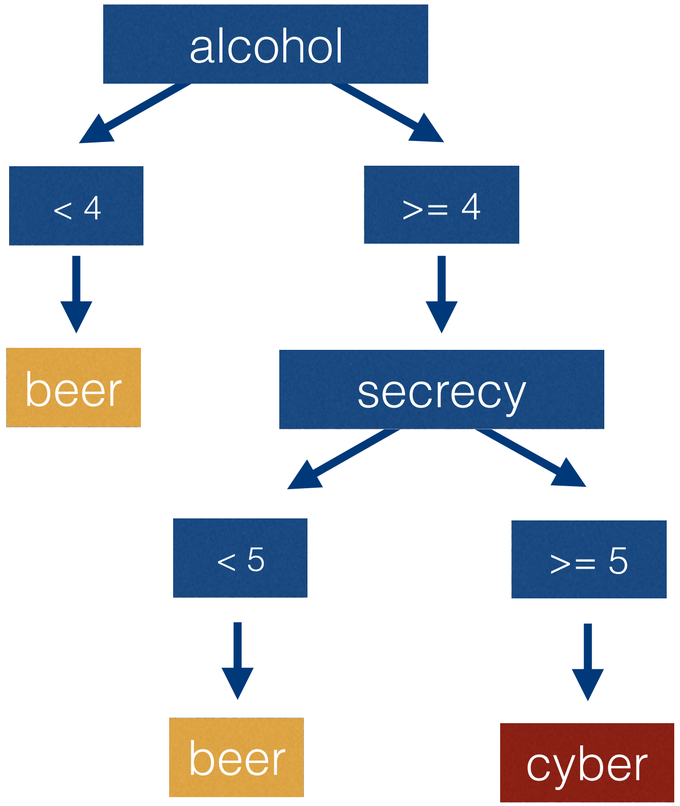


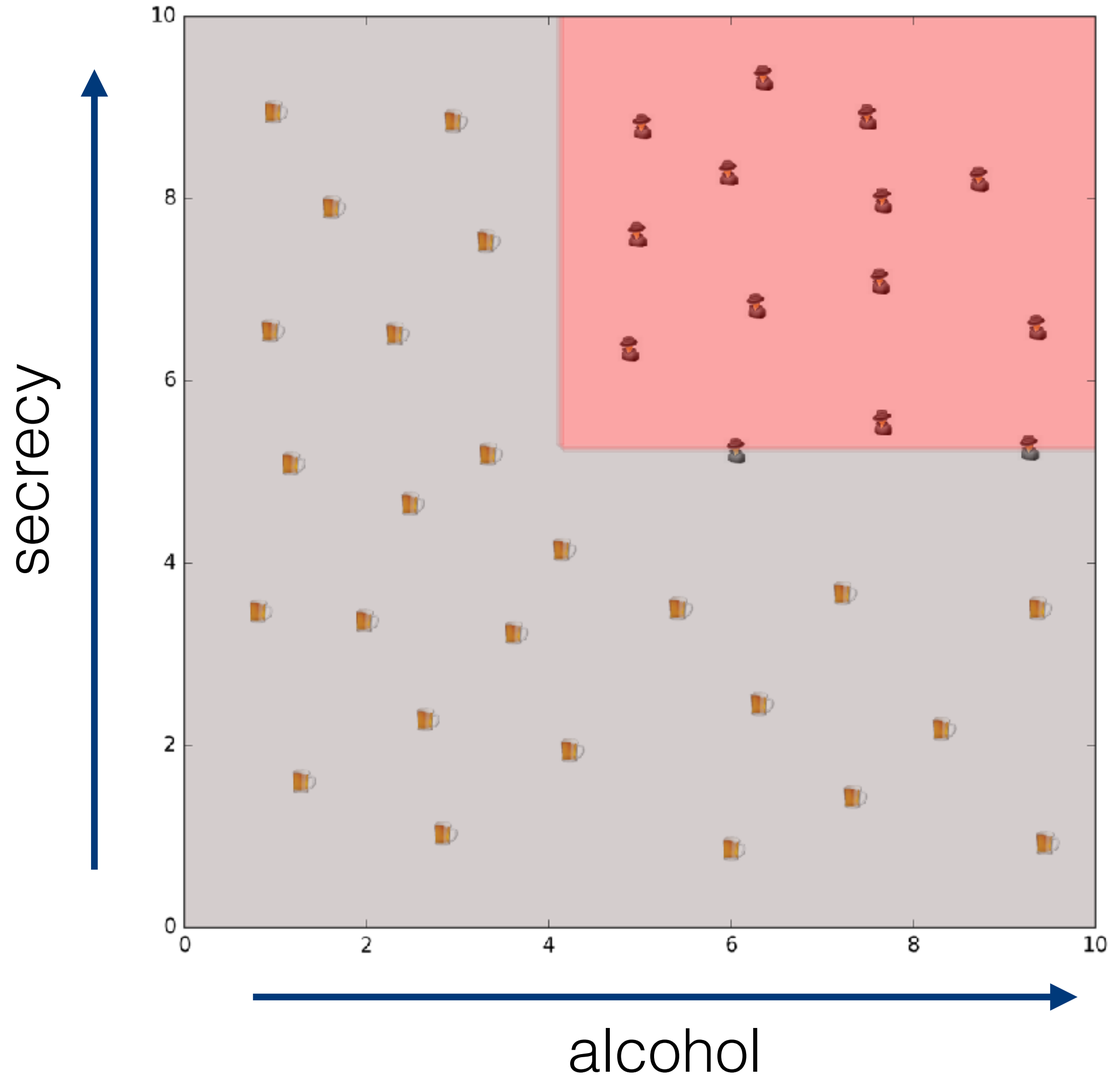
This is Charles' example!

How can we separate beer and cyber?

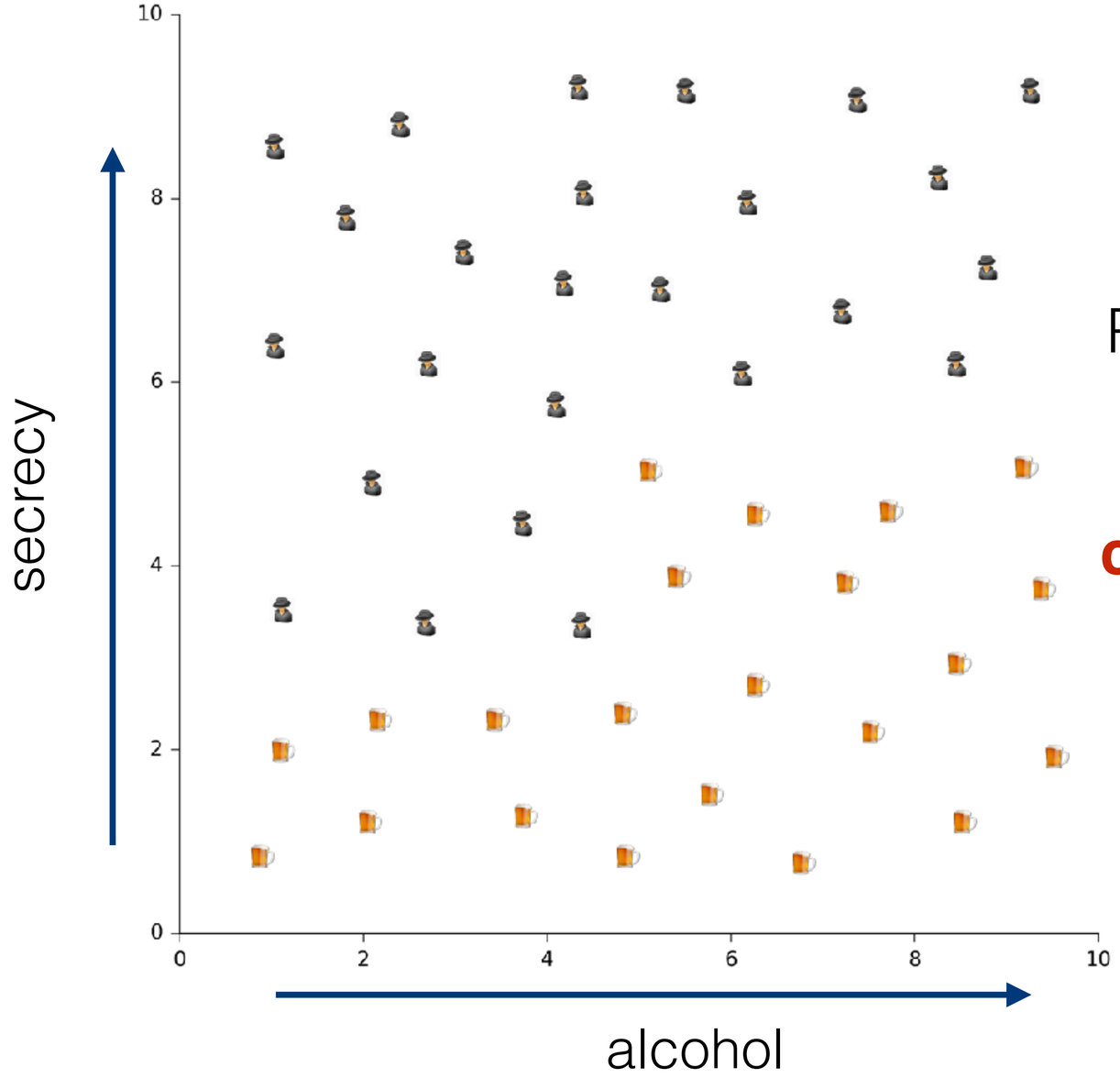


How can we separate beer and cyber?



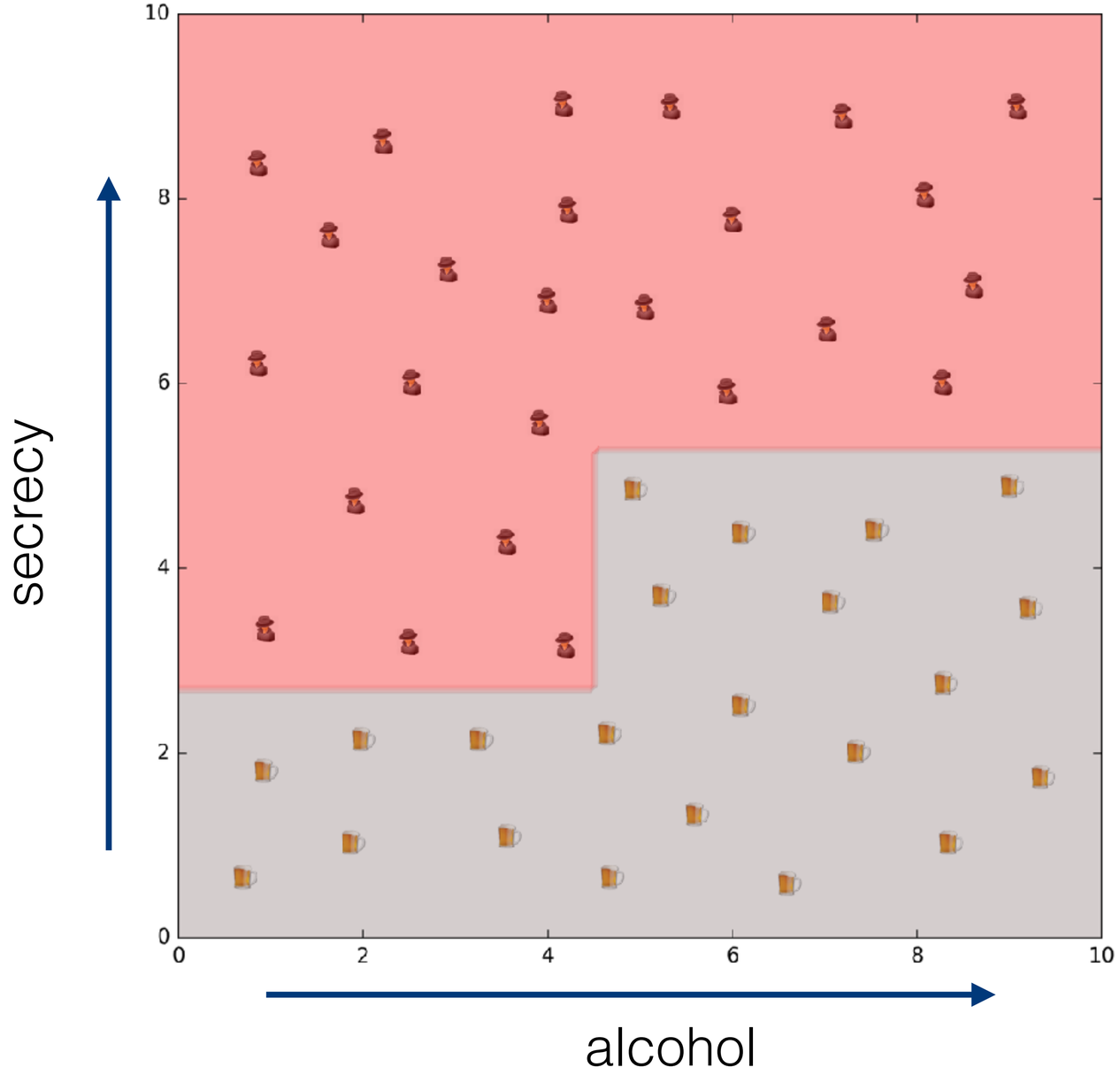


Pen and paper worksheet: solve!



Please take 10 minutes and complete **printed Worksheet - create a Decision Tree by hand!**

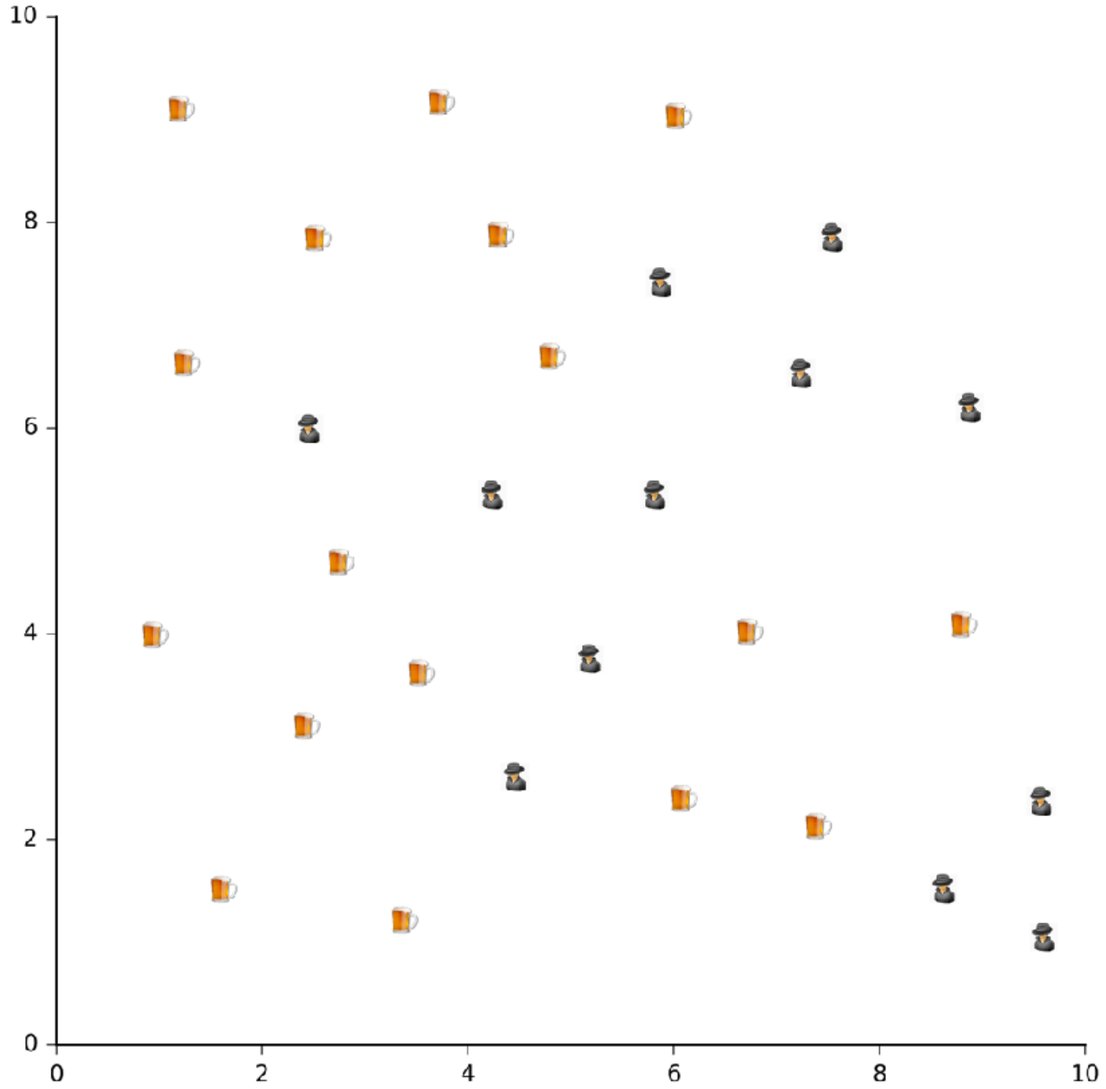
Pen and paper worksheet: solve!



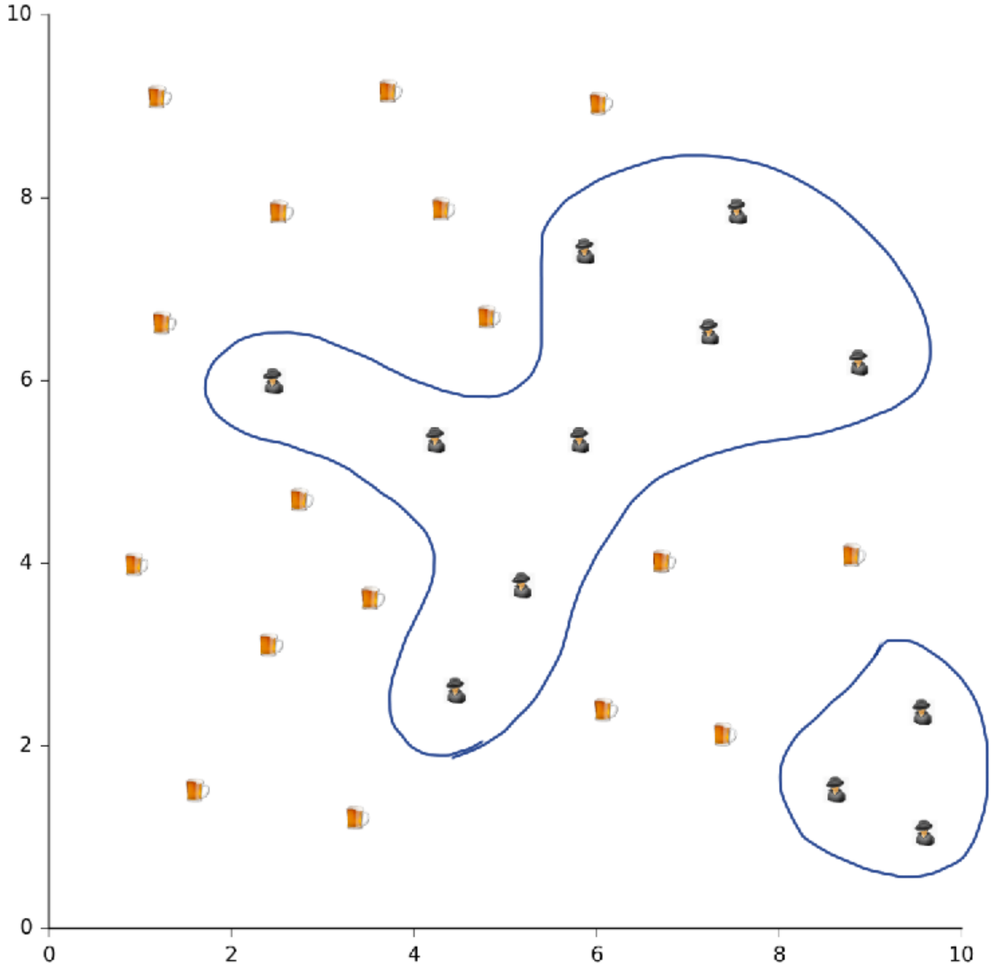
Answer:

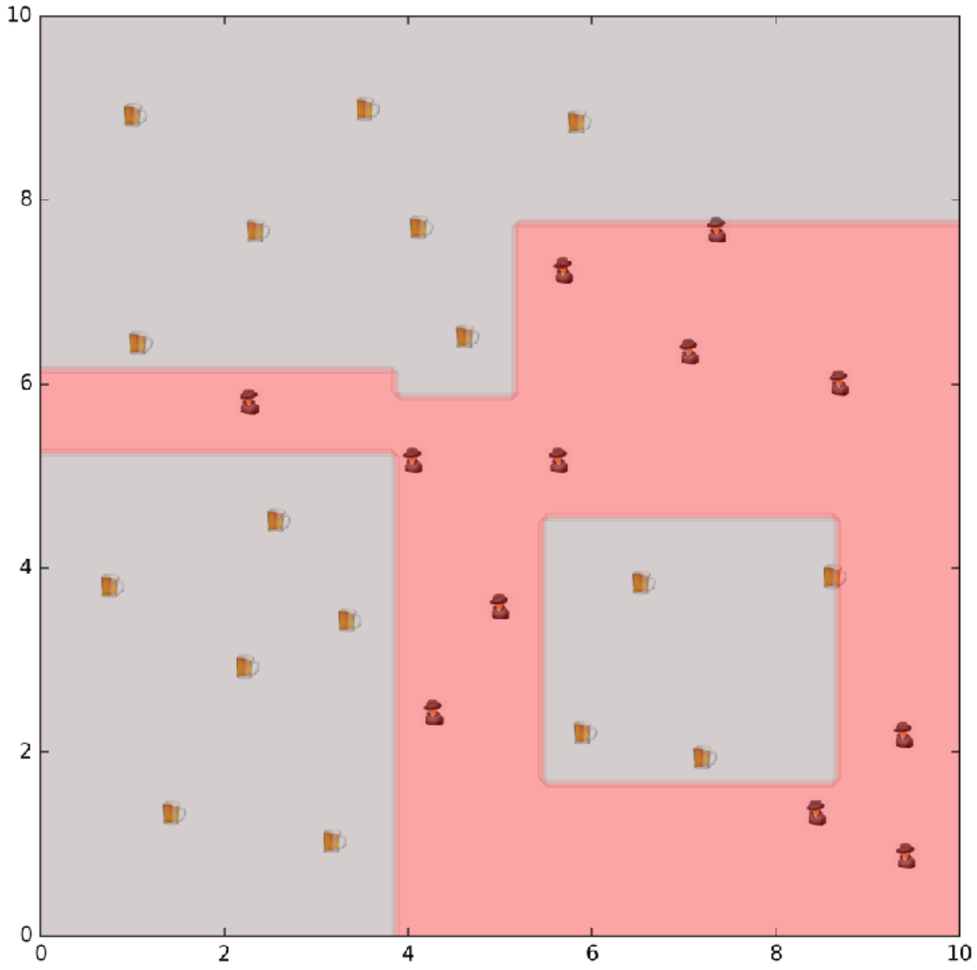
1. secrecy threshold 5.27
2. alcohol threshold 4.47
3. secrecy threshold 2.70

Ultimate last challenge! Who can solve it?



Such boundaries can be a sign of over-fitting!

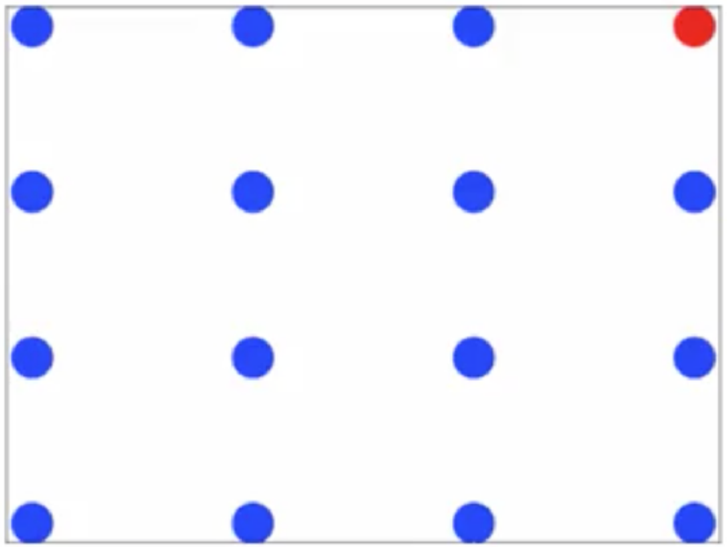




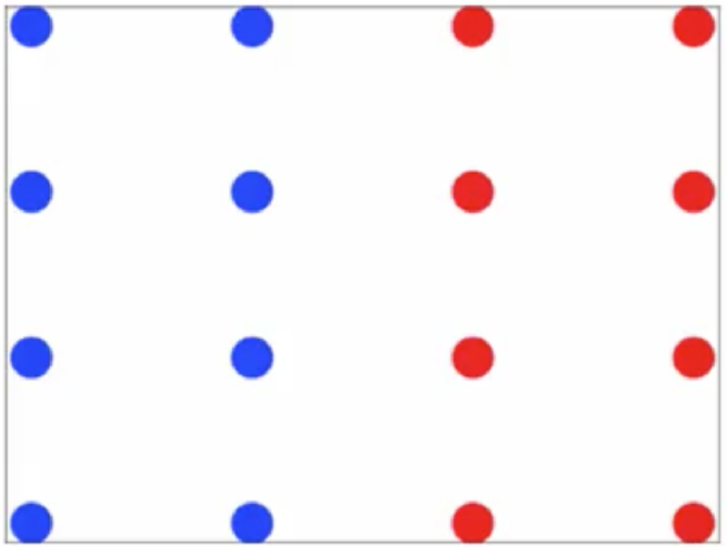
Lastly: How does a Decision Tree exactly decide to split?

Criterion for best splits: **Impurity**

A



B

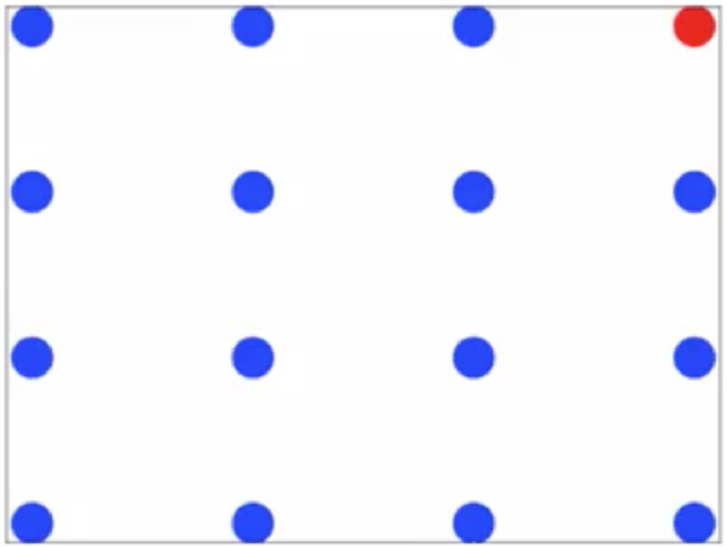


Quiz: By intuition which example appears to be more pure, that is, separates the two classes better?

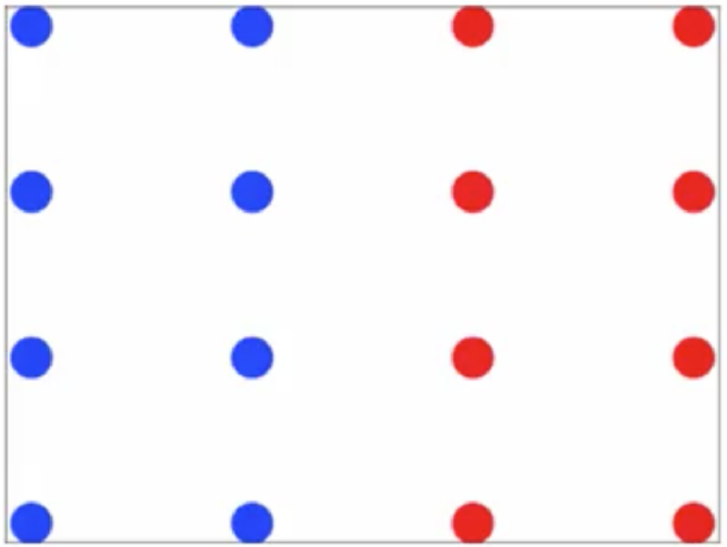
Lastly: How does a Decision Tree exactly decide to split?

Criterion for best splits: **Information Gain**

A



B



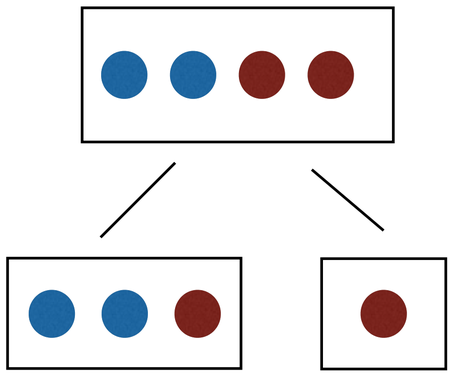
Answer: A

Which feature below gives the best split?

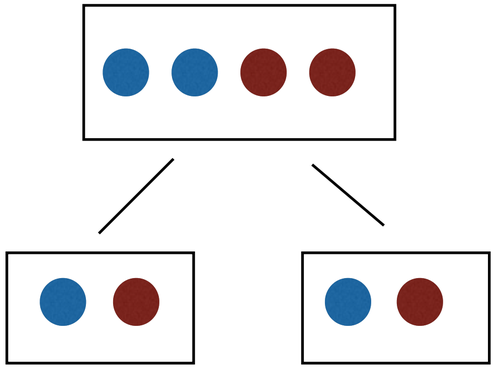
Information Gain = entropy(parent) - [weighted average]entropy(children)

entropy(parent)=1.0
most impure binary system

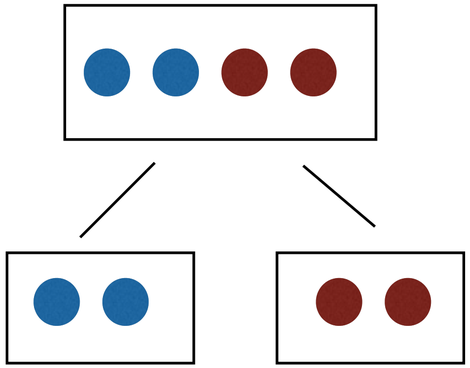
choosing next best feature to split next



feature 1



feature 2



feature 3

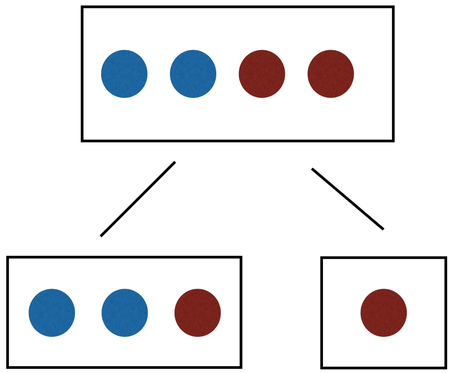
?

Which feature below gives the best split?

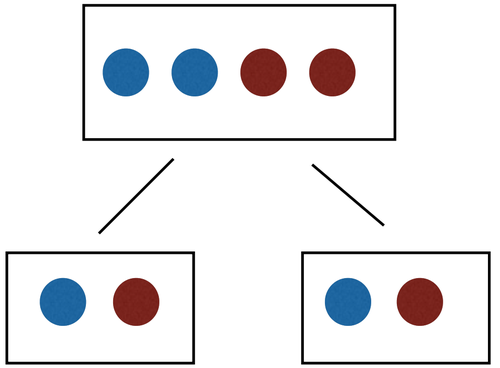
Information Gain = entropy(parent) - [weighted average]entropy(children)

entropy(parent)=1.0
most impure binary system

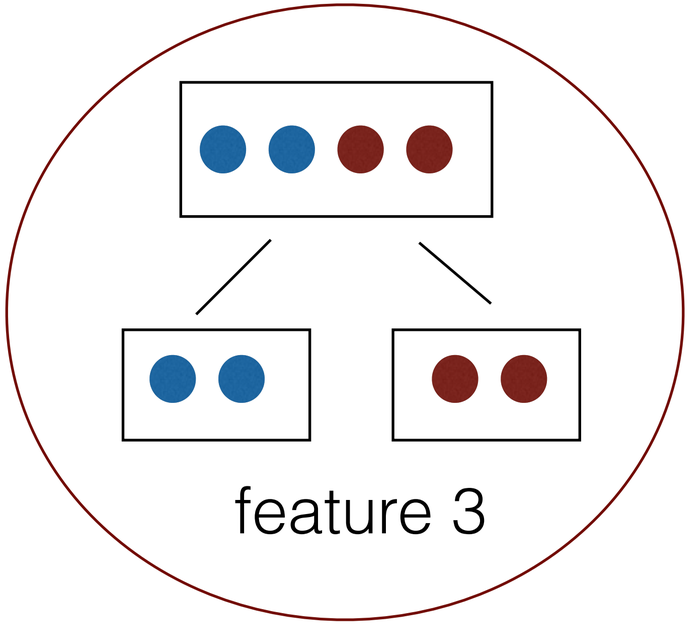
choosing next best feature to split next



feature 1



feature 2



feature 3

Information Gain	**0.3112**	**0**	**1**

Advantages of Decision Trees

- Can be used for regression or classification
- Can be displayed graphically
- Highly interpretable
- Can be specified as a series of rules, and more closely approximate human decision-making than other models
- Prediction is fast
- Features don't need scaling
- Automatically learns feature interactions (they are non-linear models)
- Tend to ignore irrelevant features (especially when there are lots of features)
- Because decision trees are non-linear models they will outperform linear models if the relationship between features and response is highly non-linear

Disadvantages of Decision Trees

- Performance is (generally) not competitive with the best supervised learning methods
- Can easily overfit the training data (tuning is required)
- Small variations in the data can result in a completely different tree (they are high variance models)
- Recursive binary splitting makes "locally optimal" decisions that may not result in a globally optimal tree
- Don't tend to work well if the classes are highly unbalanced
- Don't tend to work well with very small datasets

Ensembles

Ensembles

Ensemble learning is the process of combining several predictive models in order to produce a combined model that is more accurate than any individual model.

- **Regression:** take the average of the predictions
- **Classification:** take a vote and use the most common prediction, or take the average of the predicted probabilities

Ensembles

For ensembling to work well, the models must have the following characteristics:

- **Accurate:** they outperform random guessing
- **Independent:** their predictions are generated using different processes

The big idea: If you have a collection of individually imperfect (and independent) models, the "one-off" mistakes made by each model are probably not going to be made by the rest of the models, and thus the mistakes will be discarded when averaging the models.

Note: As you add more models to the voting process, the probability of error decreases, which is known as Condorcet's Jury Theorem.

Bagging

1. Grow *n* trees using *n* bootstrap samples from the training data.
2. Train each tree on its bootstrap sample and make predictions.
3. Combine the predictions:
    - Average the predictions for **regression trees**
    - Take a majority vote for **classification trees**

Bagging

```
bagreg = BaggingRegressor(DecisionTreeRegressor(),
n_estimators=500, bootstrap=True, oob_score=True)
bagreg.fit(X_train, y_train)
y_pred_bag = bagreg.predict(X_test)

>>> Bagged RMSE with 500 trees: 3.78785968654
```

Bagging

Here's how to calculate **"out-of-bag error":**

1. For every observation in the training data, predict its response value using **only** the trees in which that observation was out-of-bag. Average those predictions (for regression) or take a majority vote (for classification).

2. Compare all predictions to the actual response values in order to compute the out-of-bag error.

When *b* is sufficiently large, the **out-of-bag error** is an accurate estimate of **out-of-sample error**.

```
bagreg.oob_score_
```

Random Forest

Random Forests are a **slight variation of bagged trees** that have even better performance:

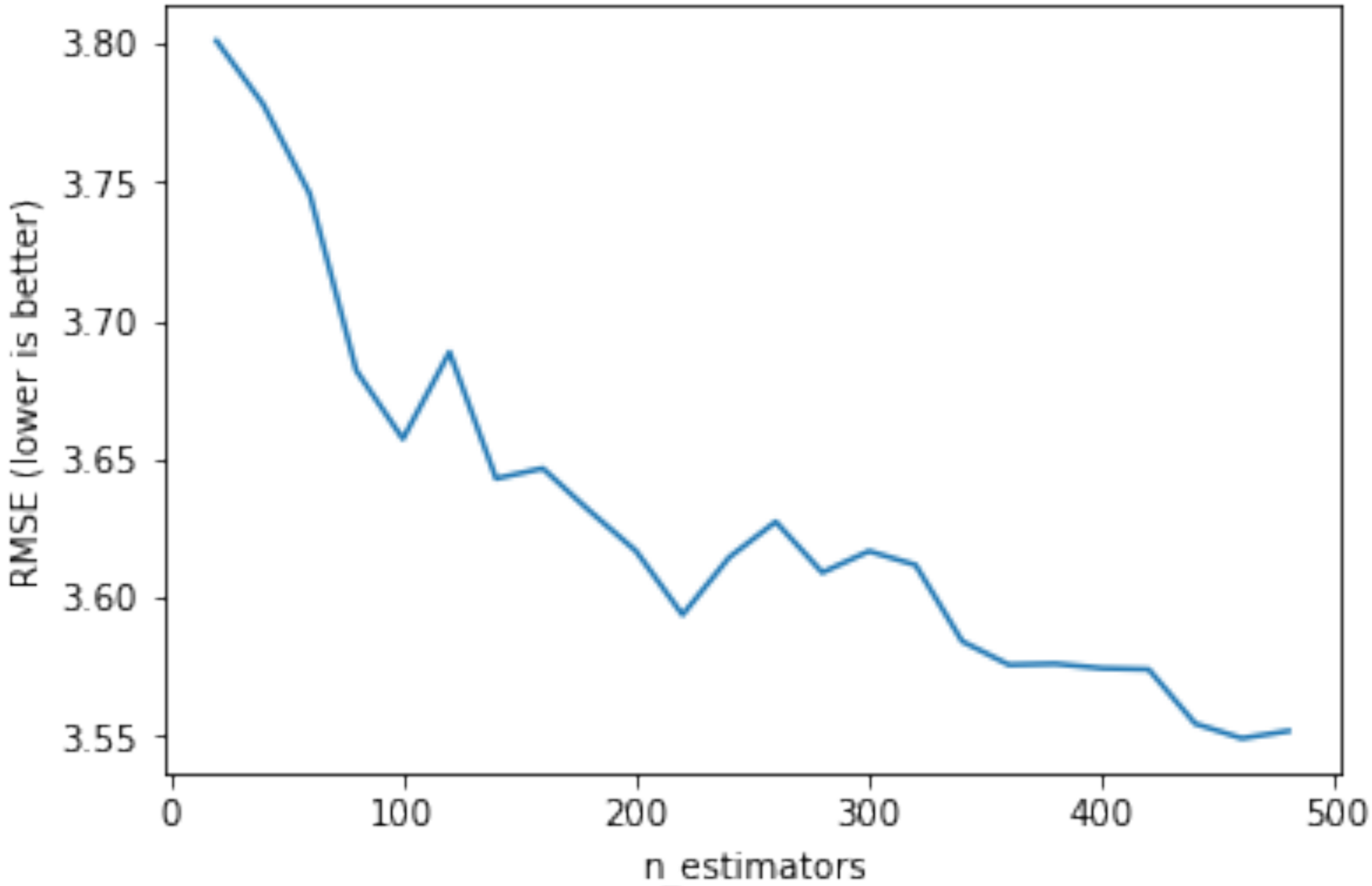
- Just like bagging, we create an ensemble of decision trees using bootstrapped samples of the training set.
- However, when building each tree, each time a split is considered, a **random sample of *m* features** is chosen as split candidates from the **full set of *p* features**. The split is only allowed to use **one of those *m* features**.
  - A new random sample of features is chosen for **every single tree at every single split**.
  - For **classification**, *m* is typically chosen to be the square root of *p* (the total number of features).
  - For **regression**, *m* is typically chosen to be somewhere between *p*/3 and *p*.

Tuning a Random Forest

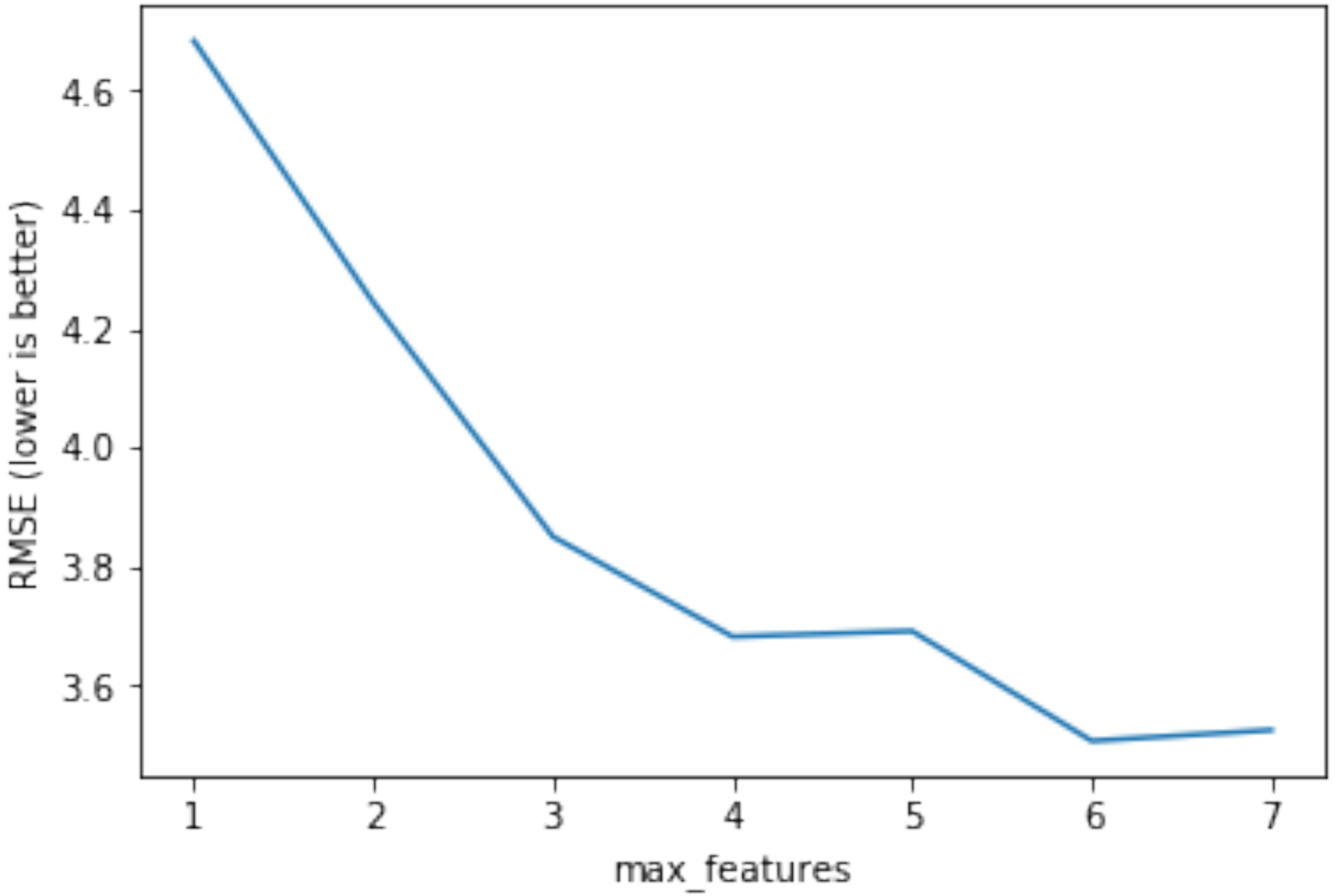
2 important parameters that should be tuned when creating a random forest model are:

- The number of trees to grow (called **n_estimators** in scikit-learn)
- The number of features that should be considered at each split (called **max_features** in scikit-learn)

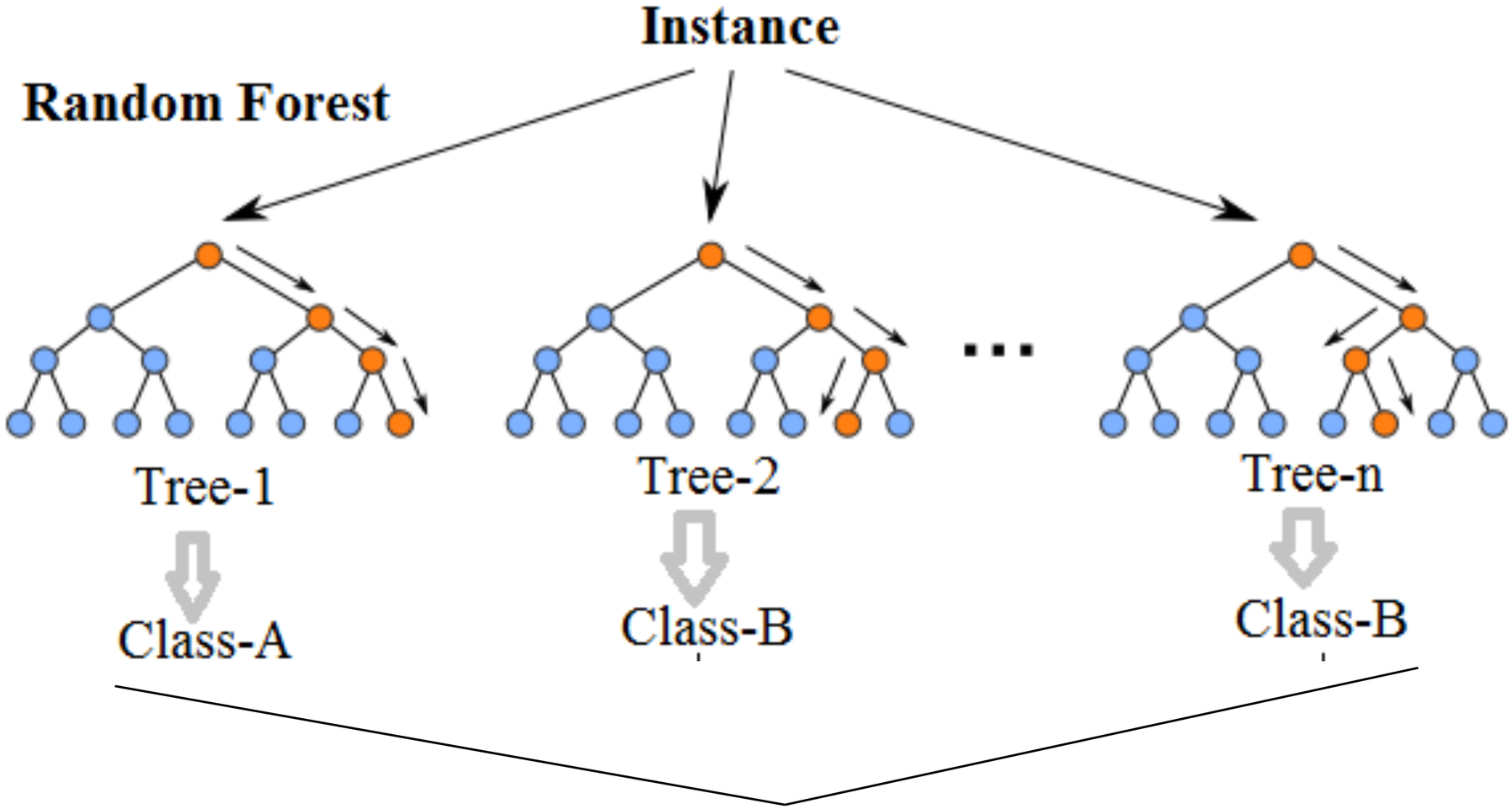
Tuning a Random Forest



Tuning a Random Forest



Tuning a Random Forest



Class A: p% Class B: 1-p%

Majority Win

Random Forests

Advantages of Random Forests:

- Performance is competitive with the best supervised learning methods
- Provides a more reliable estimate of feature importance
- Allows you to estimate out-of-sample error without using train/test split or cross-validation

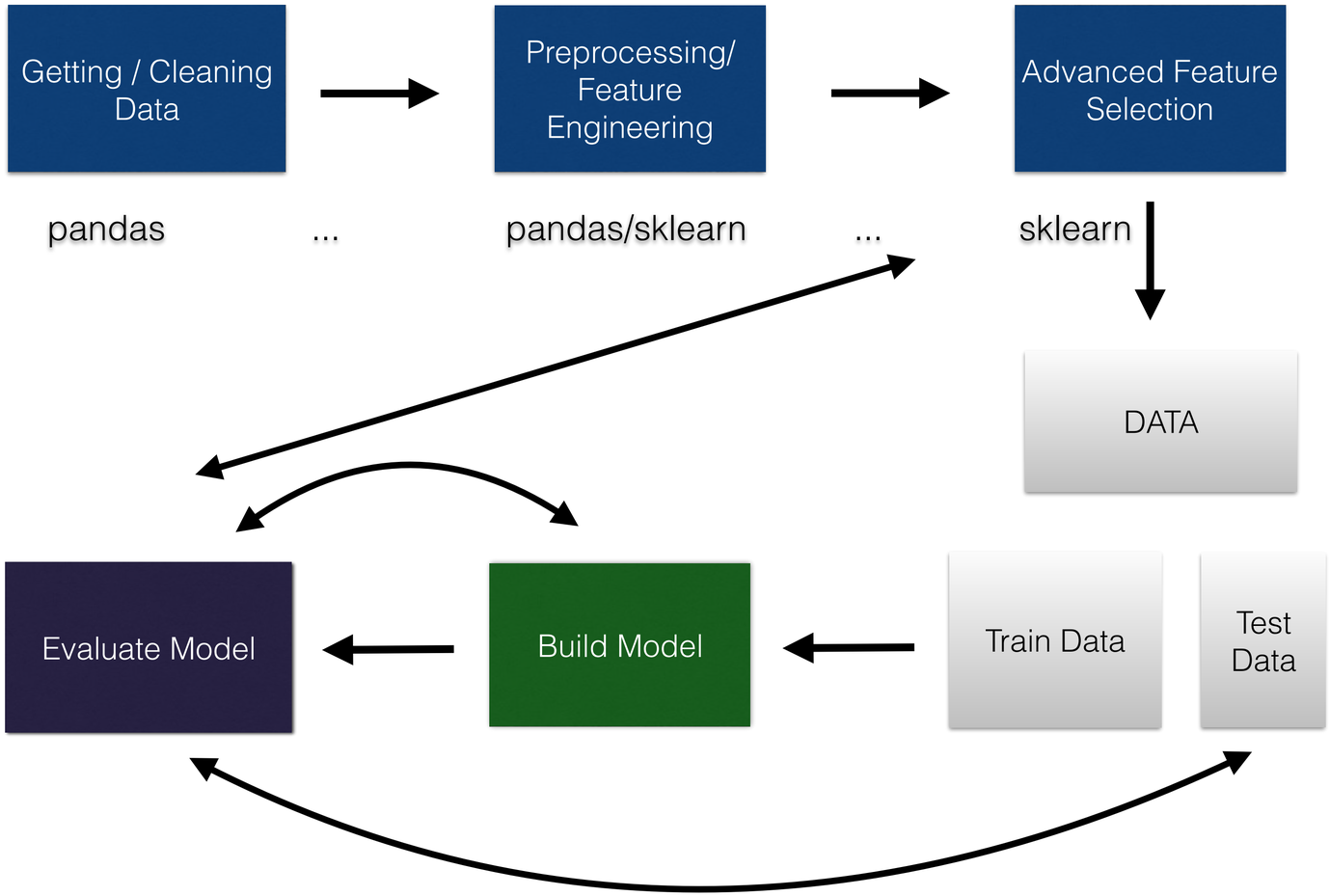
Disadvantages of Random Forests:

- Less interpretable
- Slower to train
- Slower to predict

Supervised Machine Learning

Hands on train - predict with sklearn

Machine Learning Process



Scikit-Learn Process

The basic process for any supervised learning is the same for any supervised learning in Python.

1. Create the Classifier or Regressor object.

```
clf = RandomForestClassifier()
```

2. Call the **`.fit()`** method using the **training feature matrix (X)** and the **training target vector (y)**.

```
clf.fit( X,y )
```

3. Call the **`.predict()`** method using a feature matrix

#Returns vector of predictions

```
clf.predict( X_test )
```

Train-Predict in sklearn

```
## Support Vector Machine Classification

clf = svm.SVC()
clf = clf.fit(feature_matrix, target)
target_pred = clf.predict(features_matrix)
```

clf means classifier

Train-Predict in sklearn

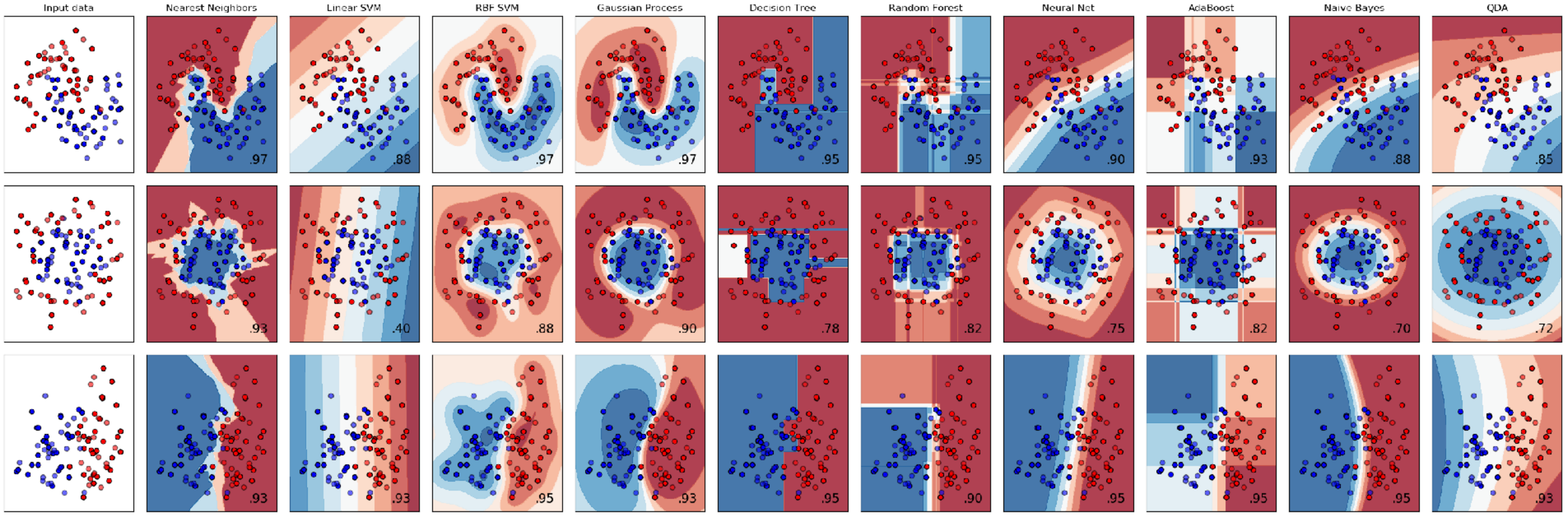
```
## Decision Tree Classification

clf = tree.DecisionTreeClassifier()
clf = clf.fit(feature_matrix, target)
target_pred = clf.predict(feature_matrix)
```

clf means classifier

Classification Algorithms in Scikit-Learn

- Generalized Linear Models
- Kernel Ridge
- Support Vector Machines
- Nearest Neighbors
- Gaussian Processes
- Naive Bayes
- Trees
- Neural Networks
- AdaBoost
- Gradient Tree Boosting
- Ensemble methods

Classification Algorithms in Scikit-Learn

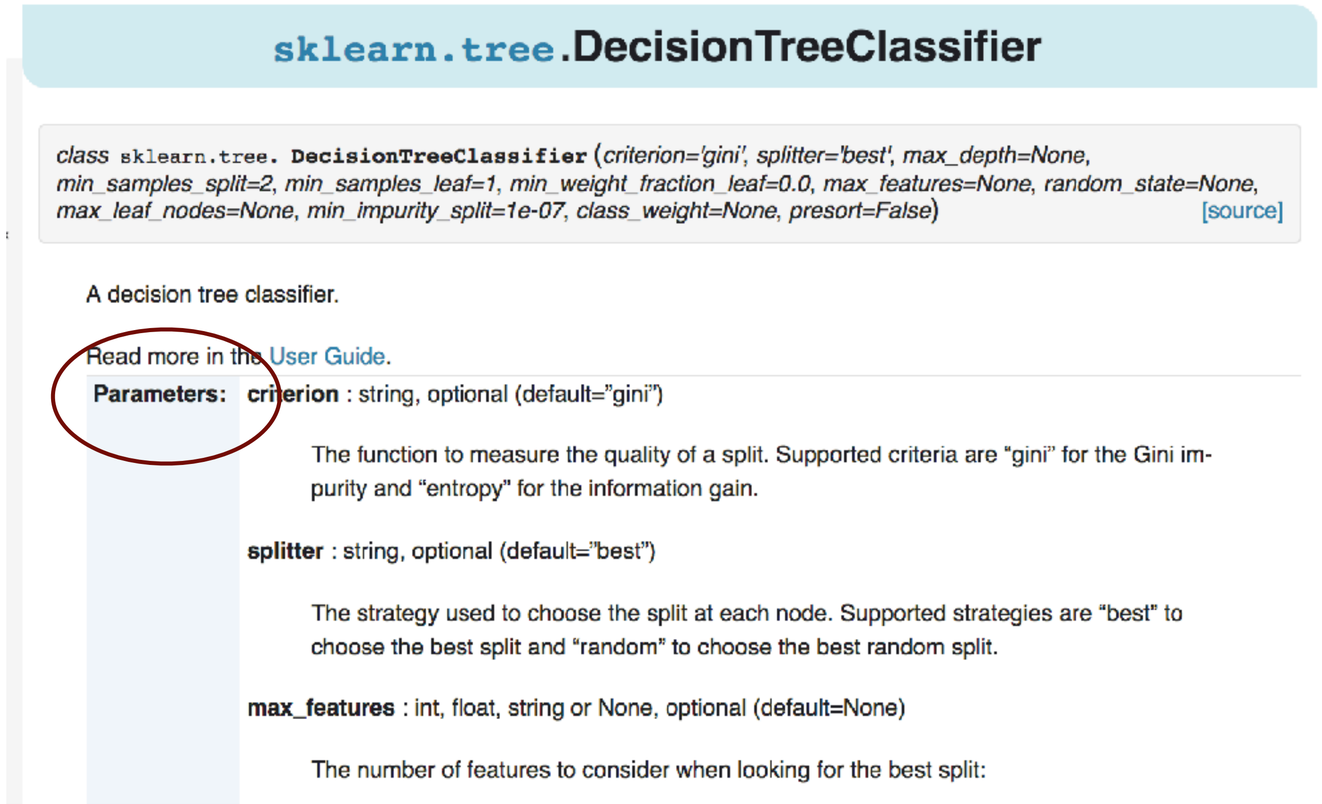


http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Python Sklearn Overview

http://scikit-learn.org/stable/

Sklearn Classifiers Navigation



sklearn.tree.DecisionTreeClassifier

class sklearn.tree. **DecisionTreeClassifier** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_split=1e-07, class_weight=None, presort=False*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters: **criterion** : string, optional (default="gini")

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : string, optional (default="best")

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

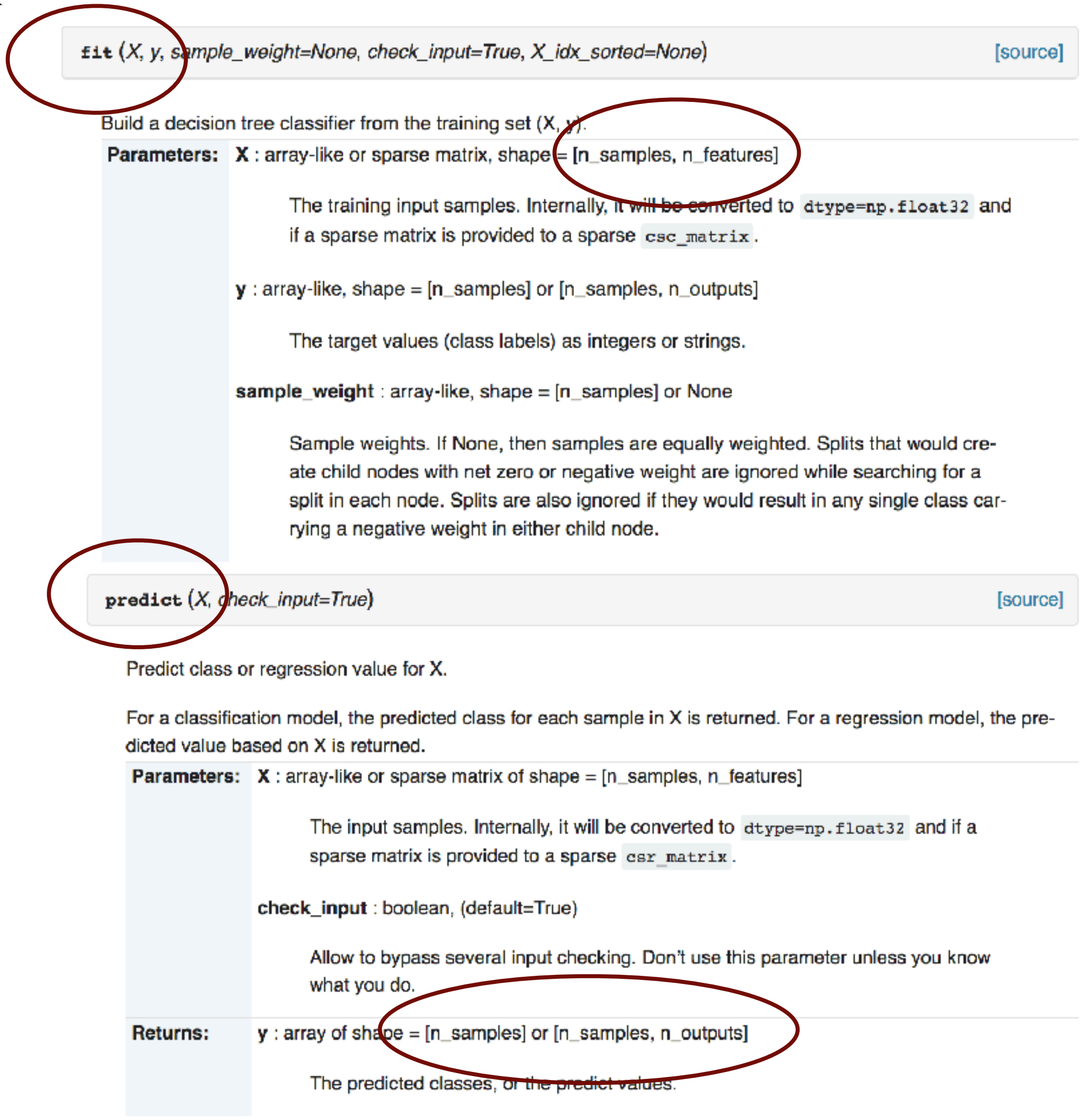
max_features : int, float, string or None, optional (default=None)

The number of features to consider when looking for the best split:

Check parameters for each classifier!

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

Sklearn Classifiers Navigation



fit (*X, y, sample_weight=None, check_input=True, X_idx_sorted=None*) [source]

Build a decision tree classifier from the training set (X, y).

Parameters: **X** : array-like or sparse matrix, shape = [n_samples, n_features]

The training input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csc_matrix`.

y : array-like, shape = [n_samples] or [n_samples, n_outputs]

The target values (class labels) as integers or strings.

sample_weight : array-like, shape = [n_samples] or None

Sample weights. If None, then samples are equally weighted. Splits that would create child nodes with net zero or negative weight are ignored while searching for a split in each node. Splits are also ignored if they would result in any single class carrying a negative weight in either child node.

predict (*X, check_input=True*) [source]

Predict class or regression value for X.

For a classification model, the predicted class for each sample in X is returned. For a regression model, the predicted value based on X is returned.

Parameters: **X** : array-like or sparse matrix of shape = [n_samples, n_features]

The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

check_input : boolean, (default=True)

Allow to bypass several input checking. Don't use this parameter unless you know what you do.

Returns: **y** : array of shape = [n_samples] or [n_samples, n_outputs]

The predicted classes, or the predict values.

Check input and output dimensions of fit and predict methods for feature matrix X and target vector y!

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

Sklearn Classifiers Navigation

Attributes:

classes_ : array of shape = [n_classes] or a list of such arrays

The classes labels (single output problem), or a list of arrays of class labels (multi-output problem).

feature_importances_ : array of shape = [n_features]

The feature importances. The higher, the more important the feature. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance [R245].

max_features_ : int,

The inferred value of max_features.

n_classes_ : int or list

The number of classes (for single output problems), or a list containing the number of classes for each output (for multi-output problems).

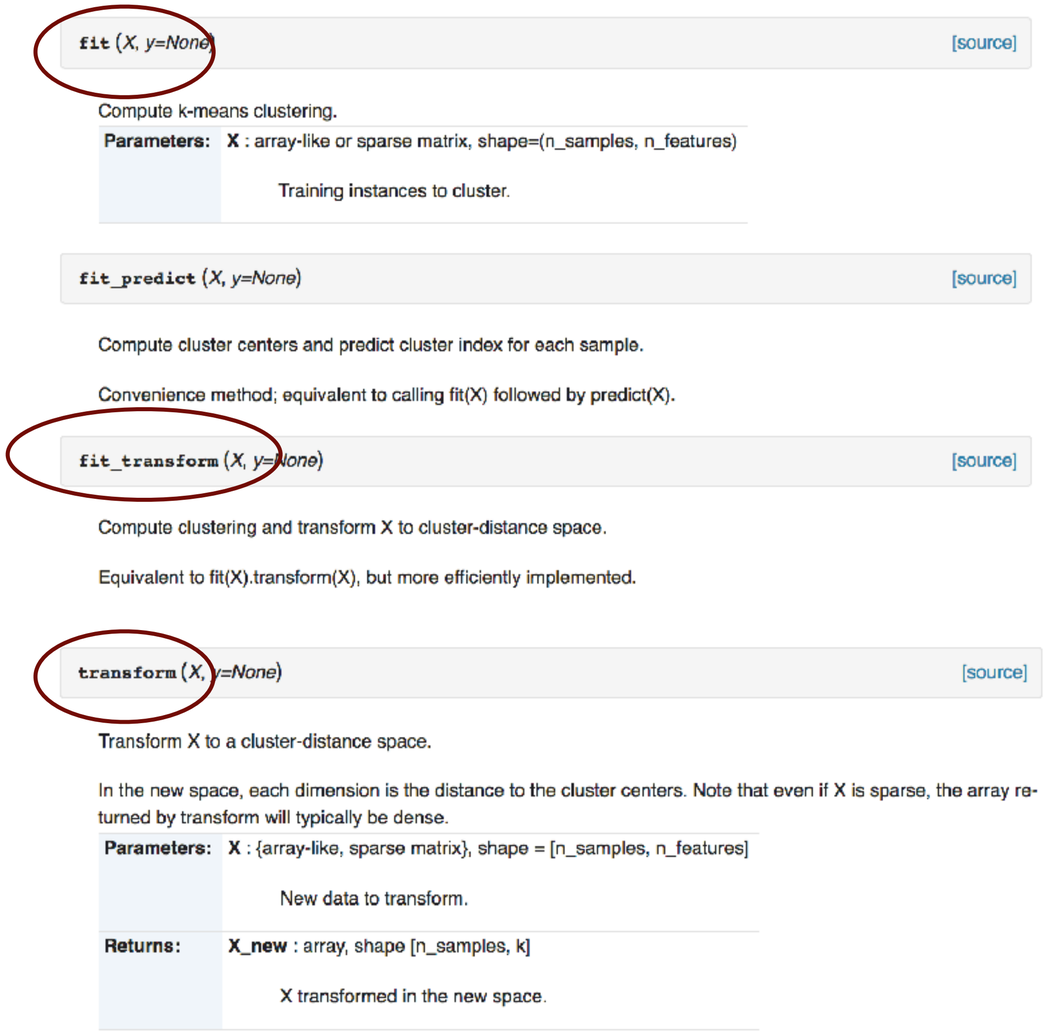
n_features_ : int

The number of features when `fit` is performed.

After calling fit method you can retrieve certain attributes of your clf object!

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

Sklearn Classifiers Navigation



fit (*X*, *y=None*) [source]

Compute k-means clustering.

Parameters: **X** : array-like or sparse matrix, shape=(n_samples, n_features)

Training instances to cluster.

fit_predict (*X*, *y=None*) [source]

Compute cluster centers and predict cluster index for each sample.

Convenience method; equivalent to calling fit(X) followed by predict(X).

fit_transform (*X*, *y=None*) [source]

Compute clustering and transform X to cluster-distance space.

Equivalent to fit(X).transform(X), but more efficiently implemented.

transform (*X*, *y=None*) [source]

Transform X to a cluster-distance space.

In the new space, each dimension is the distance to the cluster centers. Note that even if X is sparse, the array returned by transform will typically be dense.

Parameters: **X** : {array-like, sparse matrix}, shape = [n_samples, n_features]

New data to transform.

Returns: **X_new** : array, shape [n_samples, k]

X transformed in the new space.

Clustering and dimensionality reduction methods like PCA have more fit and transform rather than predict and often combine fit_transform in one method!

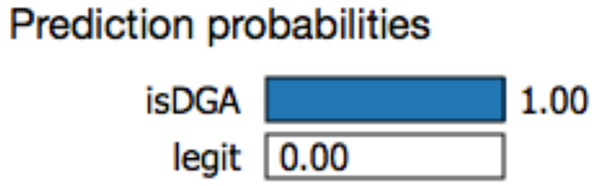
http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

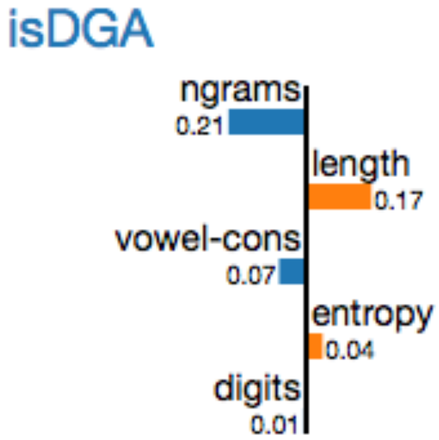
Supervised Machine Learning

Understanding the classifier

"Why should I trust you?"
Explaining the predictions of any classifier

```
import lime
import lime.lime_tabular
explainer = lime.lime_tabular.LimeTabularExplainer(<training_data>,
        feature_names=<feature names>,
        class_names=<class names>,
        discretize_continuous=False)

exp = explainer.explain_instance( <test_row>,
   <model>.predict_proba,
   num_features=5,
   top_labels=1)

exp.show_in_notebook(show_table=True, show_all=False)
```





Feature	Value
length	6.00
digits	0.00
entropy	2.58
vowel-cons	0.50
ngrams	1606.84

https://github.com/marcotcr/lime

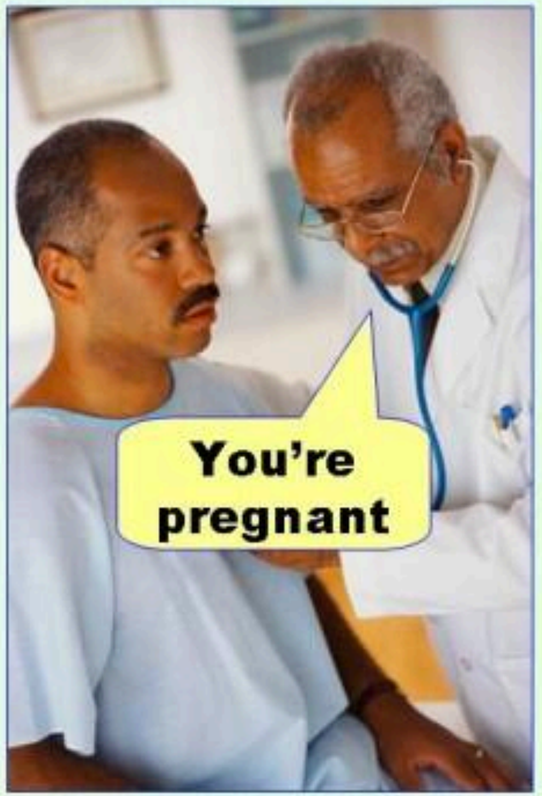
gtkcyber.cc

Supervised Machine Learning

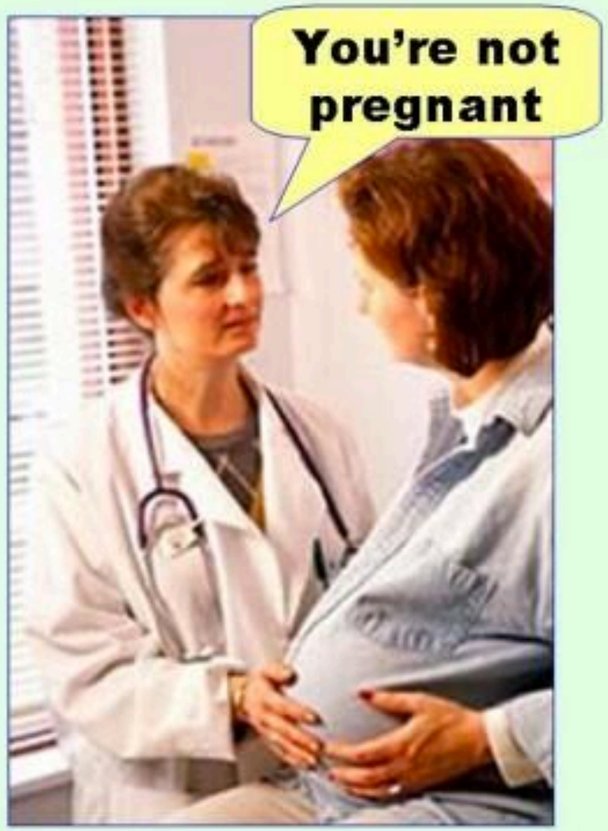
Metrics and cross-validation

Classifier Performance

Type I error
(false positive)



Type II error
(false negative)



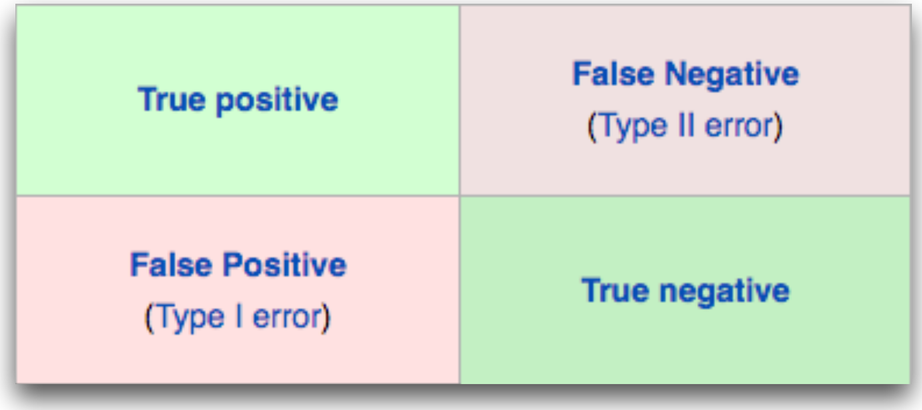
Confusion Matrix

```
target =      [1,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0]
target_pred = [0,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0]
print(metrics.confusion_matrix(target, target_pred))
```

True target

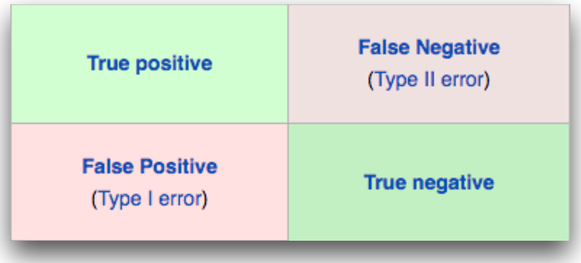
```
[[7 2]
 [3 4]]
```

Predicted target



Accuracy

```
accuracy = metrics.accuracy_score(target, target_pred)
```



True target

0 [[7 2]

1   [3 4]]

0 1

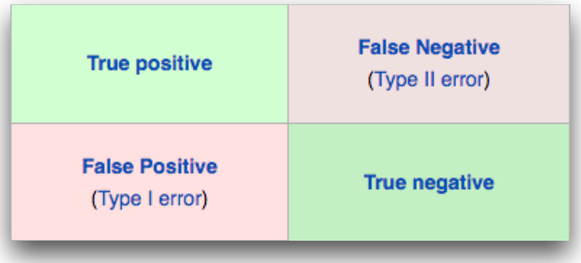
Predicted target

Accuracy = (TP + TN)/ N

Quiz: Calculate by hand!

Accuracy

```
accuracy = metrics.accuracy_score(target, target_pred)
```



True target

0 [[7 2]

1 [3 4]]

0 1

Predicted target

Accuracy = (TP + TN)/ N

= (7+4)/16

= 0.6875

Precision

Column-wise!

Precision

```
precision_class = metrics.precision_score(target, target_pred, average = None)
precision_avg = metrics.precision_score(target, target_pred, average = 'binary')
```

True positive	**False Negative** (Type II error)
False Positive (Type I error)	**True negative**

True target

0 1

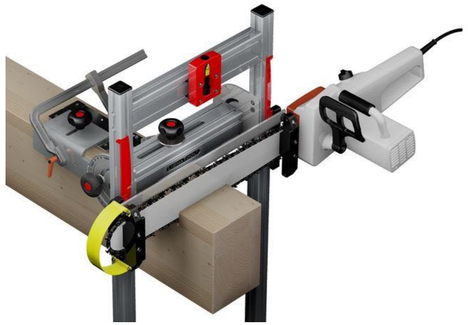
```
[ [ 7  2 ]
  [ 3  4 ] ]
```

0 1

Predicted target

Precision = TP/ (TP + FP)

Quiz: Calculate precision by hand for both classes!



Precision

```
precision_class = metrics.precision_score(target, target_pred, average = None)
precision_avg = metrics.precision_score(target, target_pred, average = 'binary')
```

True positive	**False Negative** (Type II error)
False Positive (Type I error)	**True negative**

True target

0 [[7 2]

1 [3 4]]

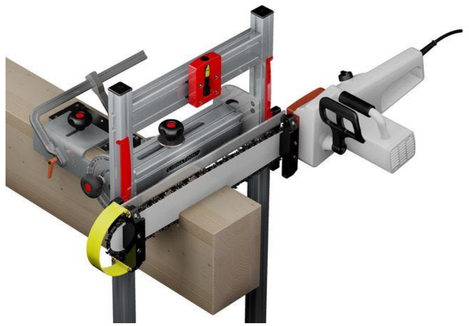
0 1

Predicted target

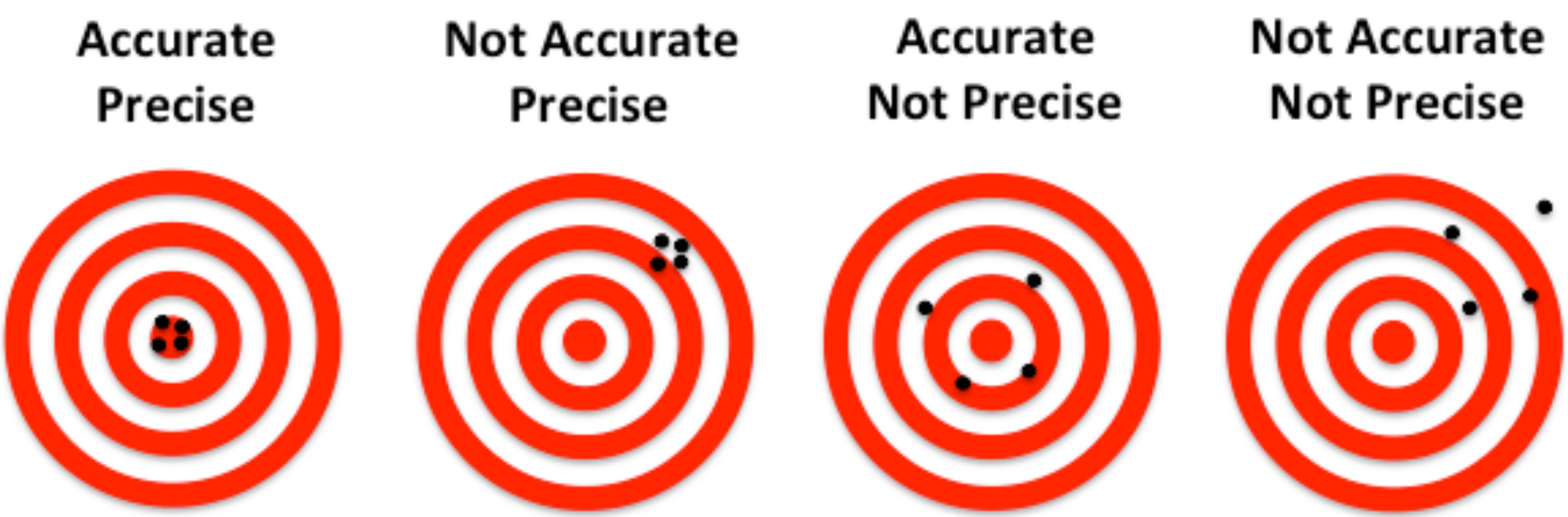
Precision = TP/ (TP + FP)

Prec_Class0 = 7/(7+3) = 0.7

Prec_Class1 = 4/(4+2) = 0.666



Precision

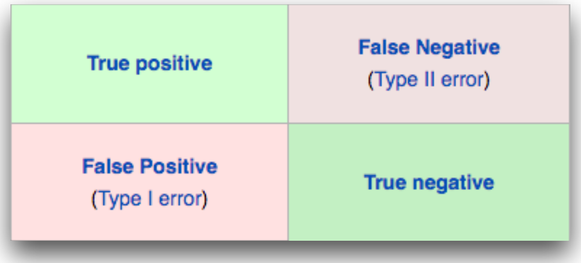

Accurate
Precise
Not Accurate
Precise
Accurate
Not Precise
Not Accurate
Not Precise

Recall

Row-wise!

Recall

```
recall_class = metrics.recall_score(target, target_pred, average = None)
recall_avg = metrics.recall_score(target, target_pred, average = 'binary')
```



True target

0 [[7 2]

1   [3 4]]

0 1

Predicted target

Recall = TP/ (TP + FN)
Quiz: Calculate recall by hand for both classes!

Recall

```
recall_class = metrics.recall_score(target, target_pred, average = None)
recall_avg = metrics.recall_score(target, target_pred, average = 'binary')
```

True positive	**False Negative** (Type II error)
False Positive (Type I error)	**True negative**

True target

0 1

```
[[7 2]
 [3 4]]
```

0 1

Predicted target

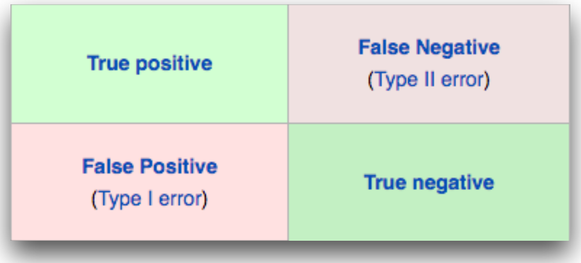
Recall = TP/ (TP + FN)

Rec_Class0 = 7/(7+2) = 0.777

Rec_Class1 = 4/(4+3) = 0.571

Quiz: compute metrics for each class of 3-class confusion matrix

```
target =      [1,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,2,2,2,2,1,1,2,0,0,0,0,0]
target_pred = [0,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,2,2,2,2,2,2,1,2,2,2,2,2]
print(metrics.confusion_matrix(target, target_pred))
```



True target

```
[[7  2  5]
 [3  4  2]
 [0  1  4]]
```

Predicted target

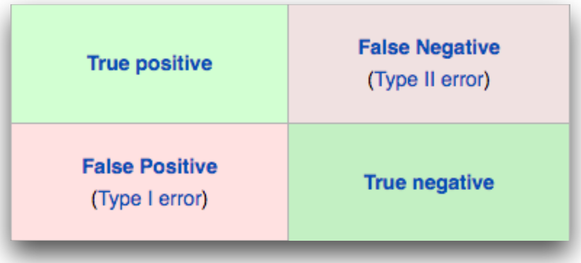
Accuracy = (Sum Diagonal)/ N

Precision = TP/ (TP + FP)

Recall = TP/ (TP + FN)

Quiz: compute metrics for each class of 3-class confusion matrix

```
target =      [1,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,2,2,2,2,1,1,2,0,0,0,0,0]
target_pred = [0,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,2,2,2,2,2,2,1,2,2,2,2,2]
print(metrics.confusion_matrix(target, target_pred))
```



True target

```
[[7  2  5]
 [3  4  2]
 [0  1  4]]
```

Predicted target

Accuracy = (7+4+4)/28 = 0.5357
Prec_Class2 = 4/(4 + 5 + 2) = 0.3636
Rec_Class2 = 4/(4 + 1 + 0) = 0.8
…

```
Precision:  [ 0.7  0.57142857  0.36363636]
Recall:  [ 0.5  0.44444444  0.8]
```

Where is the biggest crime scene?

Confusion matrices all with equal accuracy 0.6875!!!
How about precision and recall?

```
[[10  0]
 [ 5  1]]
```

A

```
[[7 2]
 [3 4]]
```

B

```
[[0  4]
 [1 11]]
```

C

Where is the biggest crime scene?

Confusion matrices all with equal accuracy 0.6875!!!
How about precision and recall?

```
Precision:  [ 0.7  1]
Recall:  [ 1  0.2]

[[10  0]
 [5  1]]
```

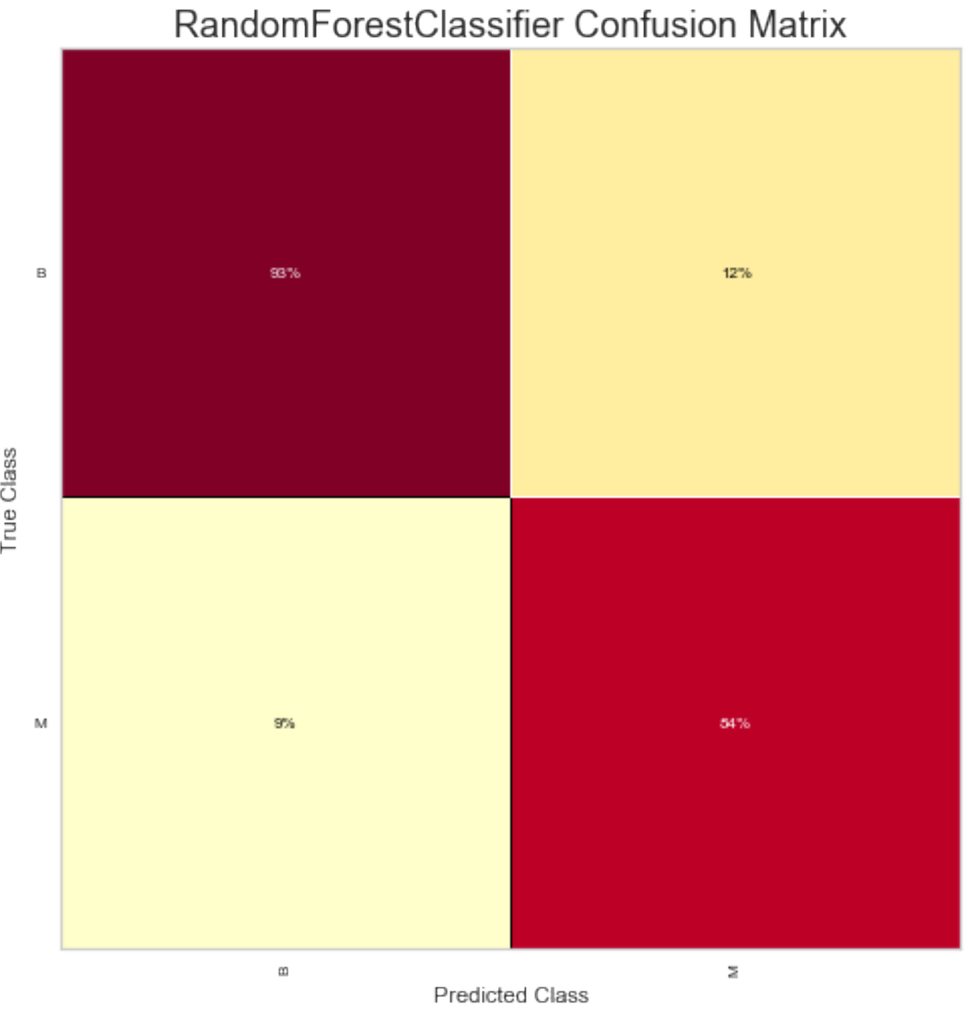
A

```
Precision:  [ 0.7  0.7]
Recall:  [ 0.8  0.6]

[[7  2]
 [3  4]]
```

B

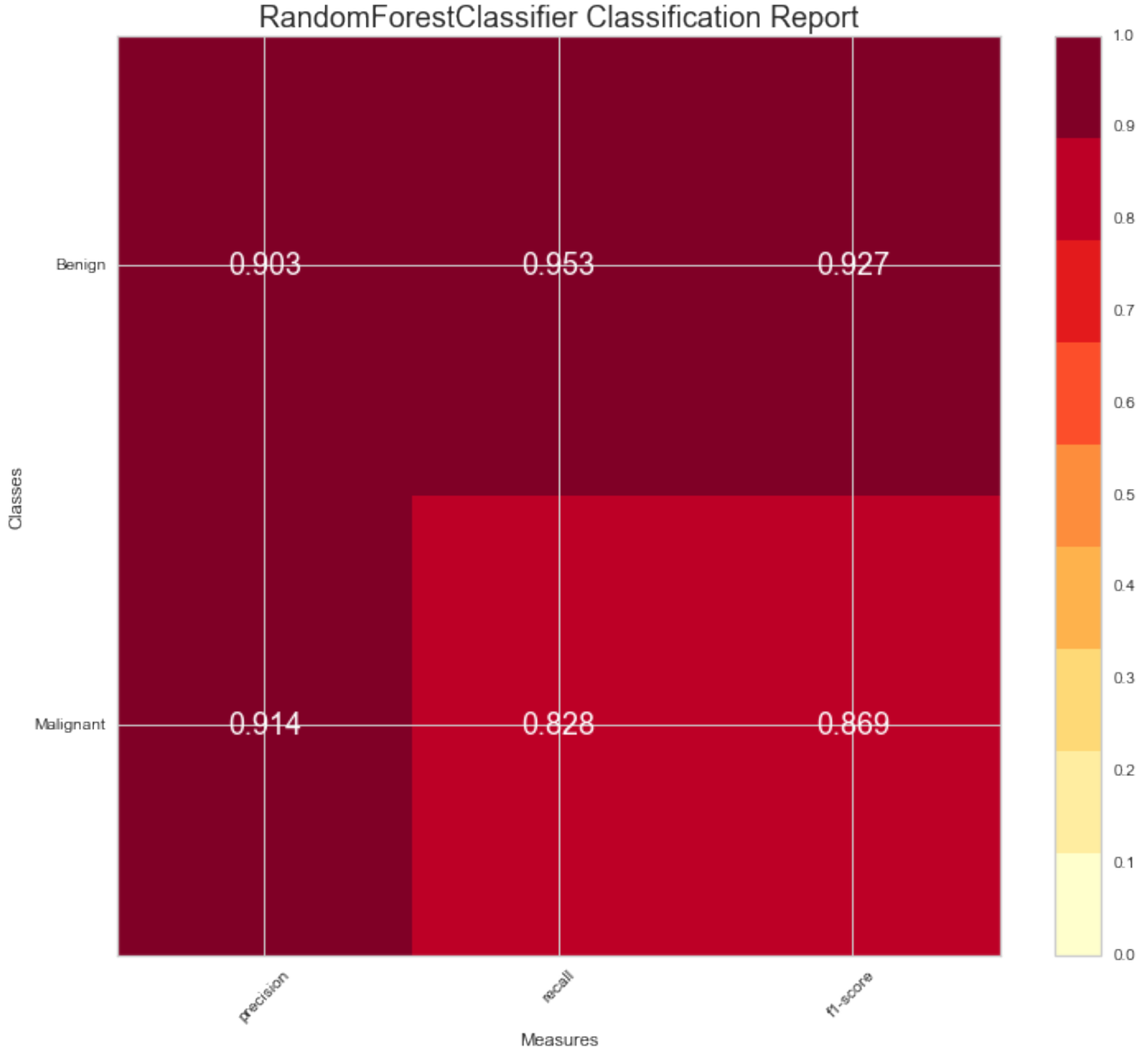
```
Precision:  [ 0  0.7]
Recall:  [ 0  0.9]

[[0  4]
 [1  11]]
```

Visualizing the Confusion Matrix

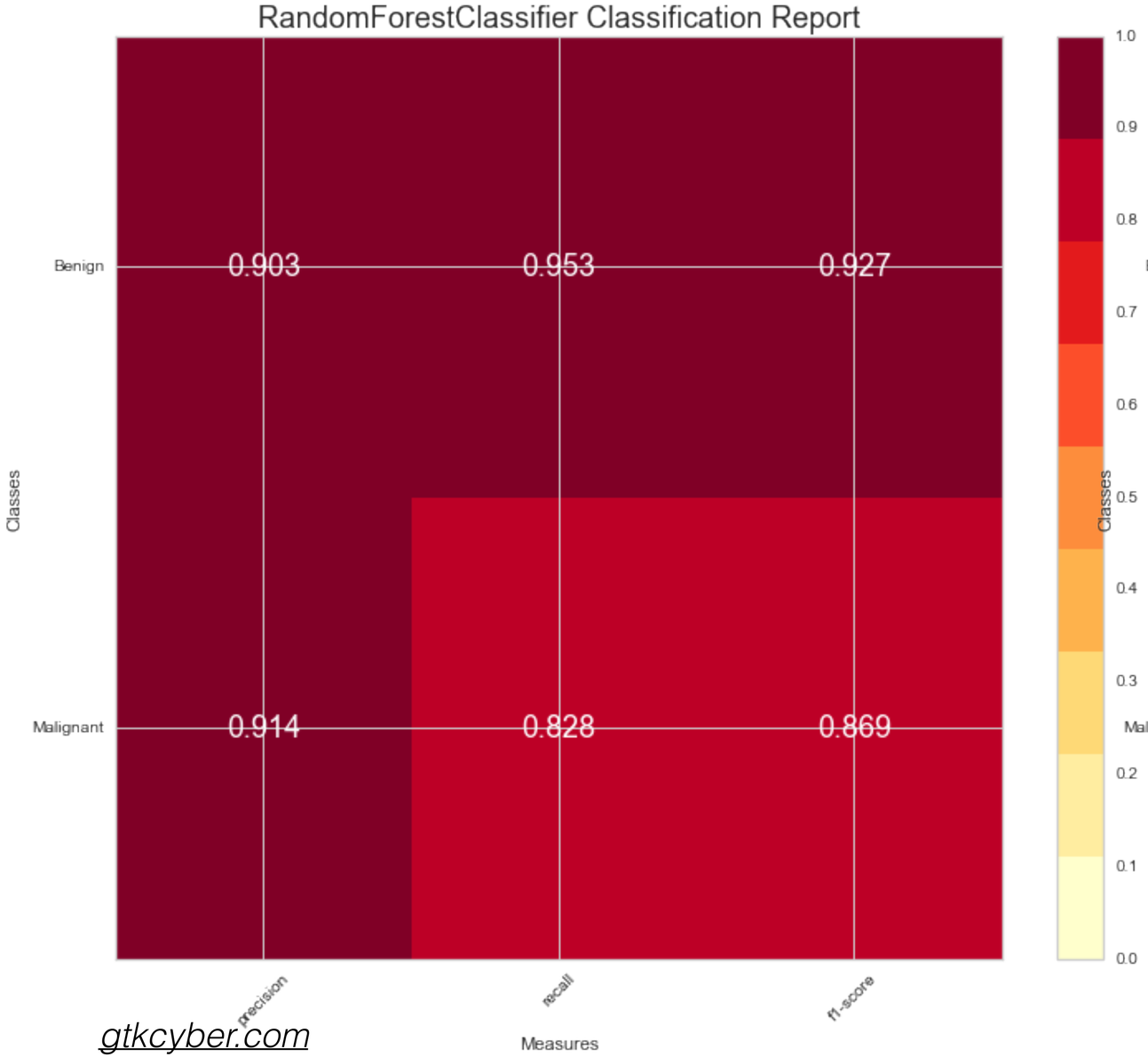


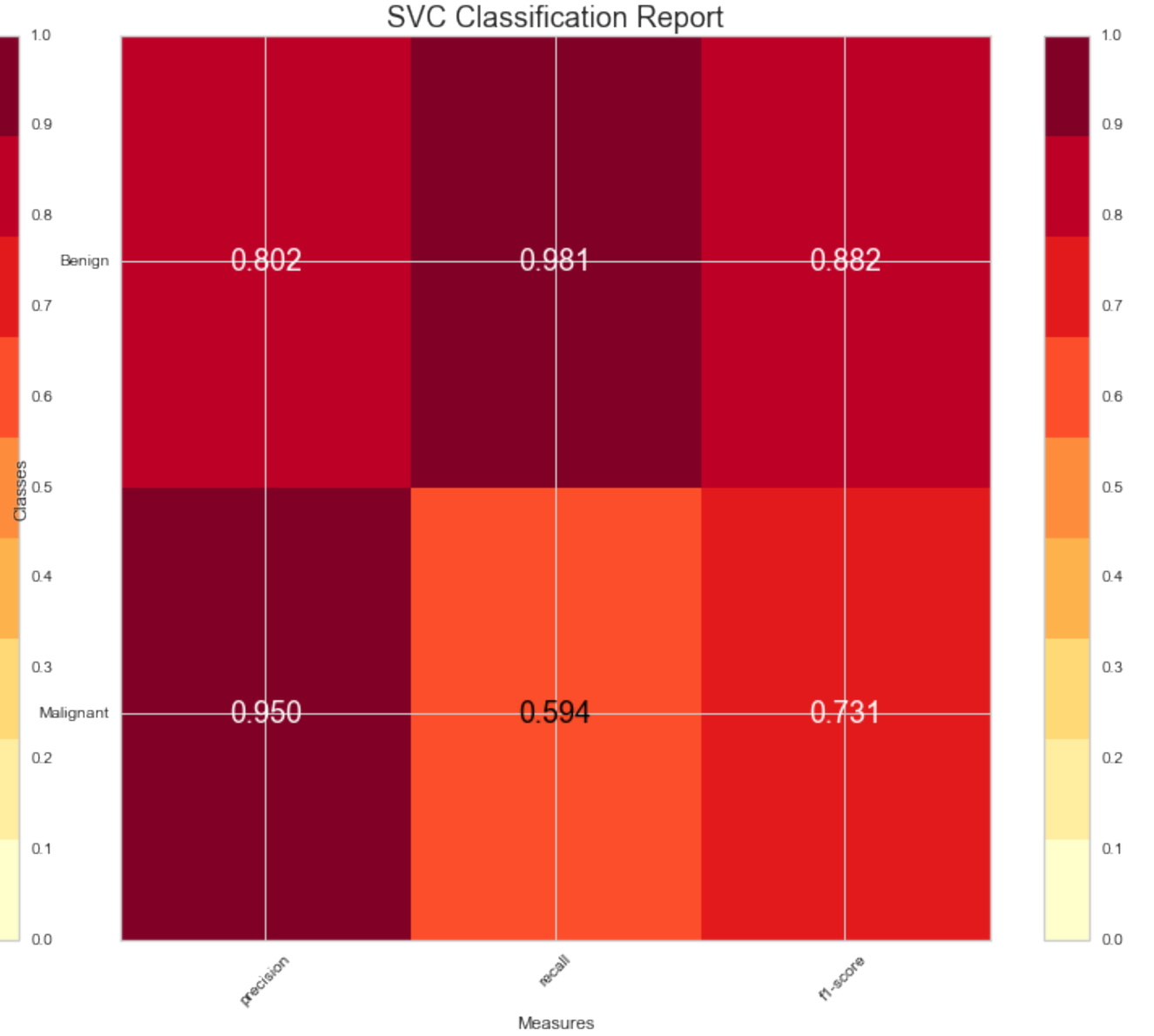

```
from yellowbrick.classifier import ConfusionMatrix

#Create the models for both the Random Forest
random_forest_model = RandomForestClassifier()

random_forest_conf_matrix = ConfusionMatrix(
                                    random_forest_model )
random_forest_conf_matrix.fit( X_train, y_train )
random_forest_conf_matrix.score( X_test, y_test )
random_forest_conf_matrix.poof()
```

Visualizing the Classification Report

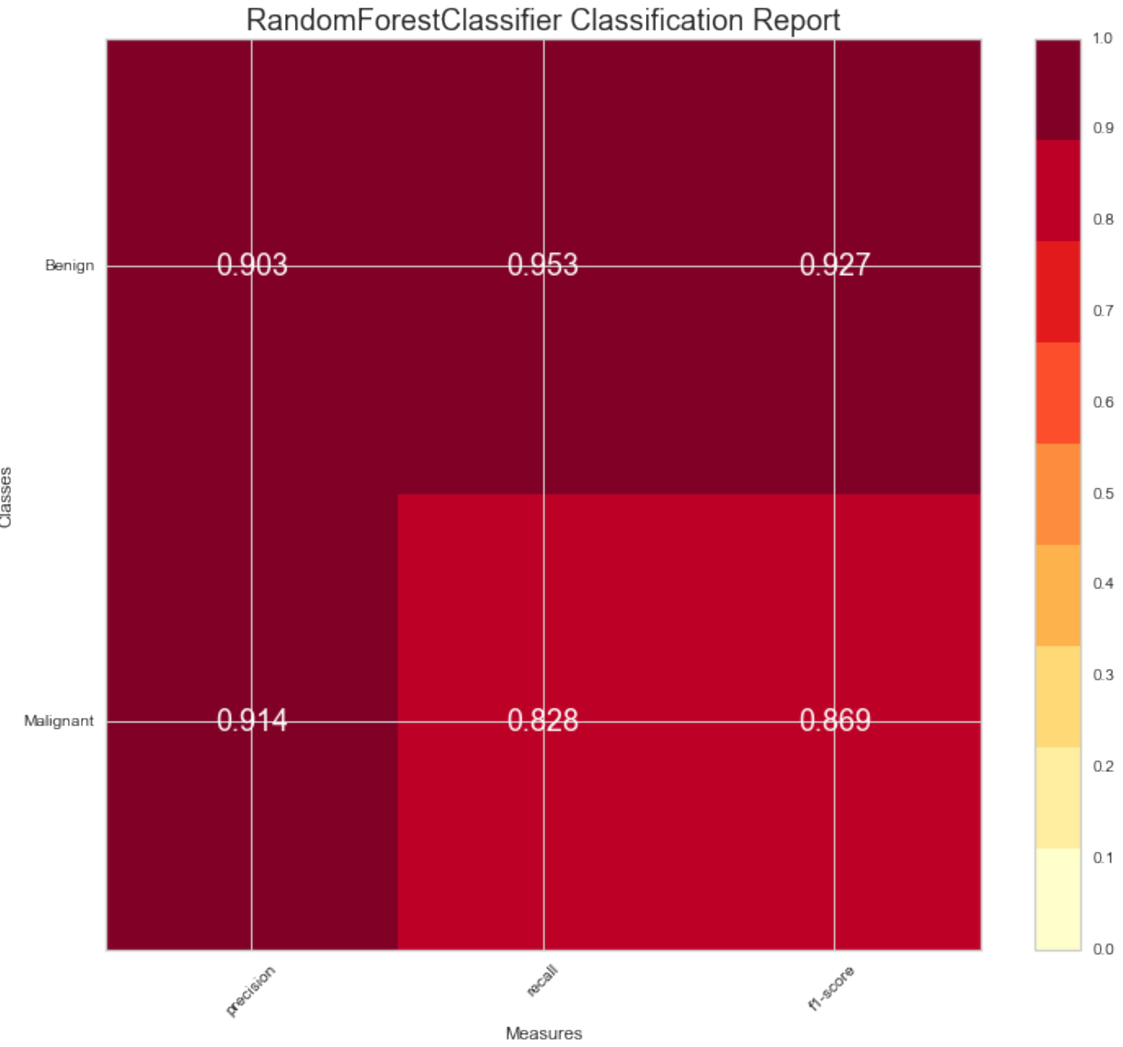


Visualizing the Classification Report





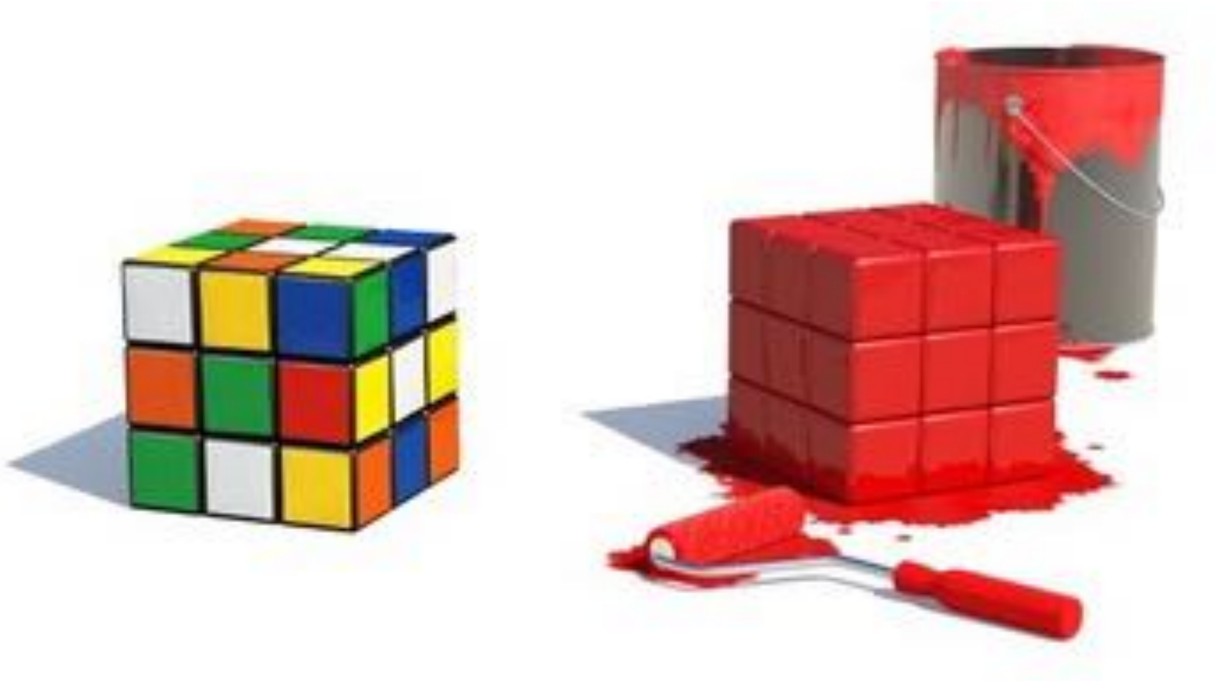
Visualizing the Classification Report




```
from yellowbrick.classifier import ClassificationReport

random_forest_class_report = ClassificationReport( random_forest_model,
                                classes=['Benign', 'Malignant'])
random_forest_class_report.fit(X_train, y_train)
random_forest_class_report.score(X_test, y_test)
random_forest_class_report.poof()
```

Cross-Validation

Why cross-validation?



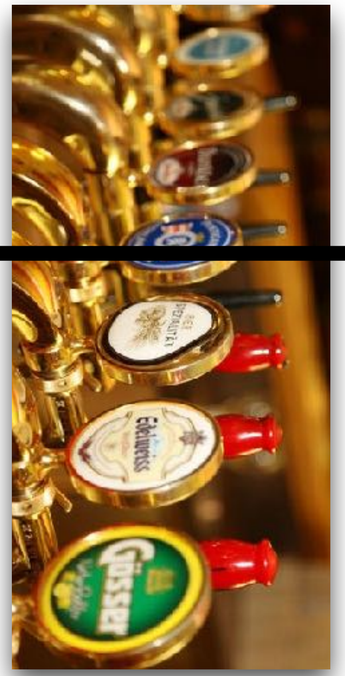
Because you don’t wanna be that guy!!!!

Split data for train and test

X

70%

target



Simple train test split!

```
# Simple Cross-Validation: Split the data set into training and test data

X_train, X_test, target_train, target_test =
model_selection.train_test_split(X, target, test_size=0.25)
```

Simple cross-validation!

```
## Train the classifier
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, target_train)

## Making predictions using test data
target_pred = clf.predict(X_test)

## Report metrics
accuracy = metrics.accuracy_score(target_test, target_pred)
```

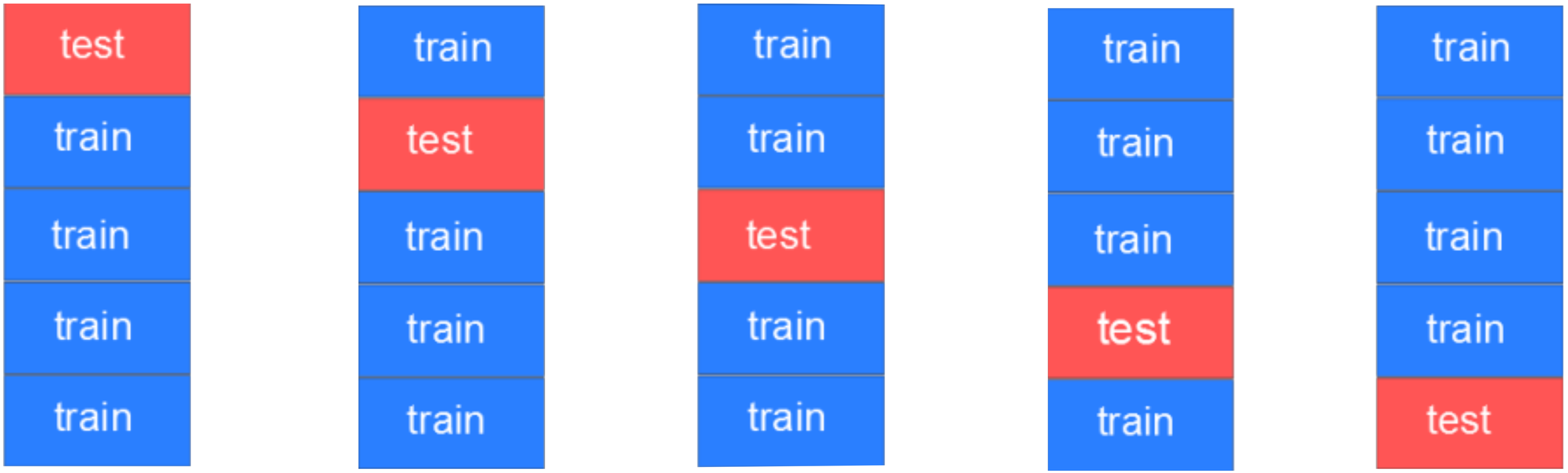
Cross Validation

Better approach: Create a bunch of train/test splits, calculate the testing accuracy for each, and average the results together.

Cross Validation

1. Split the dataset into K **equal** partitions (or "folds").
2. Use fold 1 as the **testing set** and the union of the other folds as the **training set**.
3. Calculate testing accuracy.
4. Repeat steps 2 and 3 *K* times, using a **different fold** as the testing set each time.
5. Use the **average testing accuracy** as the estimate of out-of-sample accuracy.

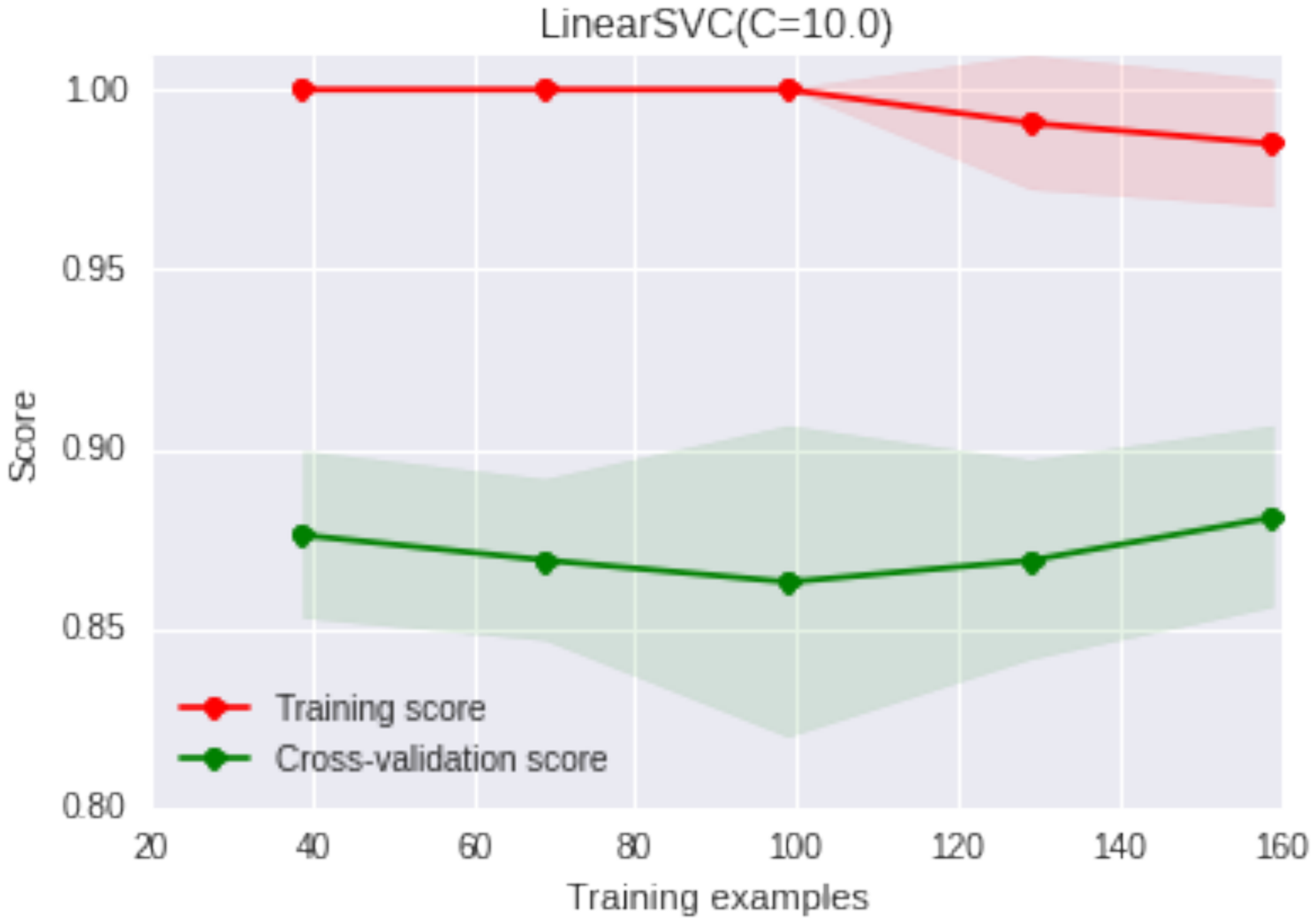
Cross Validation



K-fold Cross-Validation

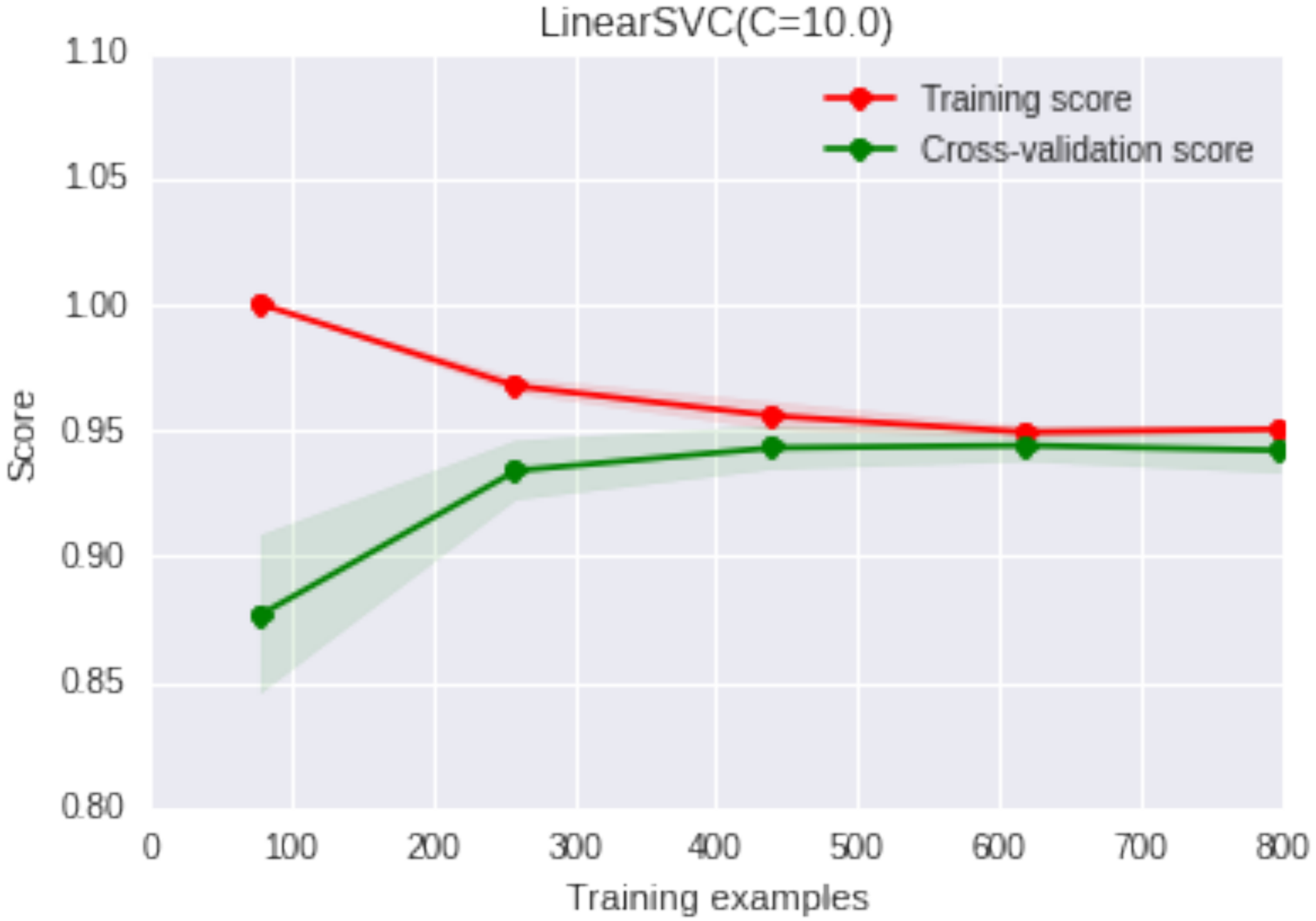
```
## Set-up generator for k subsets
cvKFold = model_selection.KFold(n_splits=3, shuffle=True,
random_state=33)
cvKFold.get_n_splits(features)

## List of k accuracies on test data
scores = model_selection.cross_val_score(clf, features, target,
cv=cvKFold)
```

Visualizing Goodness of Fit



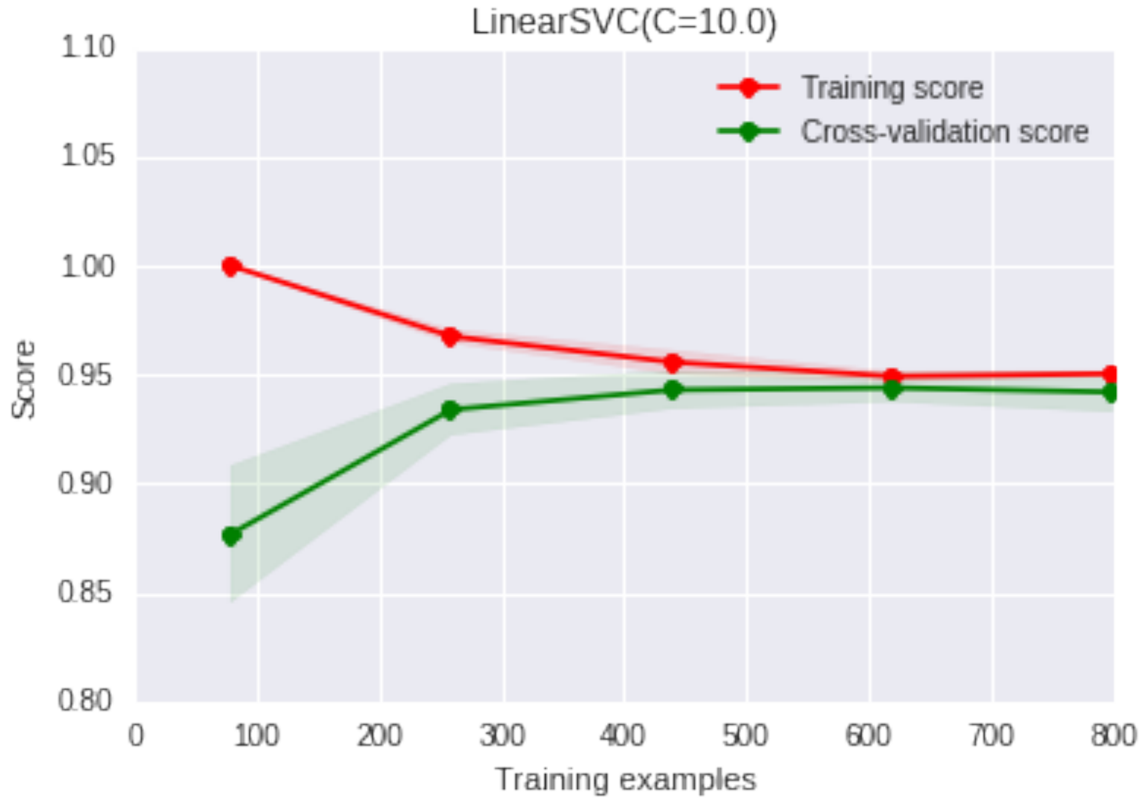
https://jmetzen.github.io/2015-01-29/ml_advice.html

Visualizing Goodness of Fit



https://jmetzen.github.io/2015-01-29/ml_advice.html

Visualizing Goodness of Fit




```
from yellowbrick.classifier.learning_curve import LearningCurveVisualizer
viz = LearningCurveVisualizer(GaussianNB())
viz.fit(X,y)
viz.poof()
```


https://jmetzen.github.io/2015-01-29/ml_advice.html

In Class Exercise

Please take 45 minutes and complete

Day 2 - DGA Detection ML Classification

Discussion In Class Exercise

- Save (e.g. pickle) fitted classifier and all models to transform raw data to features to make predictions on new URLs (in addition have all functions available)
- Feature Importances: “DurationCreated”, “Length” and “EntropyDomain” are top features. Downside is that when creation date of a new URL is not in data base or not available model becomes useless and most likely make wrong predictions if you just supply a random creation date.
- CountVectorizer and/or other Bag-of-Words methods result in a sparse matrix (=most elements are zeros) and a high-dimensional feature space. Consider word2vec embedding (e.g. https://www.tensorflow.org/tutorials/word2vec)
- Is accuracy of 0.86 acceptable for production? NO! Will result in high false positive rate!
- To achieve model accuracies of 0.999… you need much more data and/or better data, tweak feature engineering/selection and model selection (however don’t forget about the risk to overfit!)
- Consider Deep Learning! (You still need lots of data, but chances of catching “new” malicious URLs may increase due to the fact that neural network learn the feature space themselves!)

Feature Importances

0.524576	DurationCreated
0.092815	Length
0.080639	EntropyDomain
0.059617	DigitsCount
0.057386	FirstDigitIndex
0.057095	LengthDomain
0.008454	com
0.006945	net
0.005946	news.1
0.005724	org
0.004589	ru

Hyper-Parameter Tuning!

General Optimization: e.g. Gradient Descent!

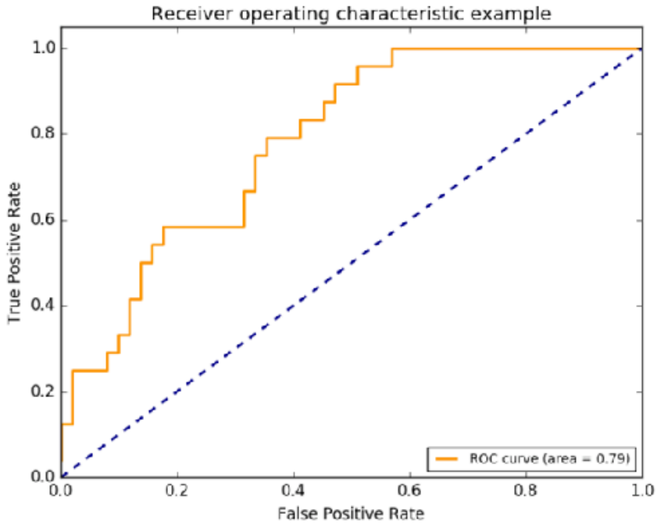
Hyper-Parameter Tuning!

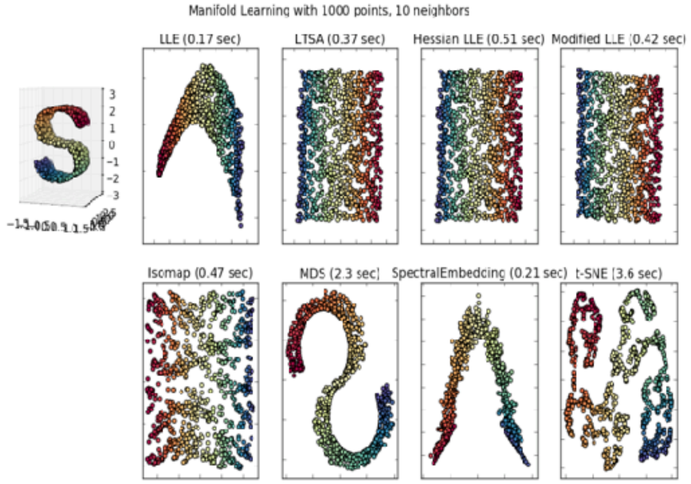
```
clf = ensemble.AdaBoostClassifier()
sss = model_selection.StratifiedShuffleSplit(n_splits=3, test_size=0.25,
random_state=33)
sss.get_n_splits(X, target)

# parameters for AdaBoost Classifier
param_dist = {'learning_rate': [.1, .2, .3, .4, .5, .6, .7, .8, .9, 1.0],
'n_estimators': [5, 10, 25, 50, 75, 100]}

random_search = model_selection.RandomizedSearchCV(clf,
param_distributions=param_dist,cv=sss.split(X, target), n_iter=12)
```

You don’t have to manually train predict all the time, while changing some parameter values !

Future Studies - sklearn

- Feature Selection using Classifiers
- Ranking of best features
- Compare multiple Classifiers and pick best!
- Model Stacking or Majority Voting
- Plotting (ROC curve, manifold learning)





What we covered

- Feature Selection & Engineering
- Math free overview of classification models
- Evaluating Model Performance
- Improving model performance

Agenda for Tomorrow

- Measuring Distances
- Overview of Unsupervised Learning Techniques
- Pipelines and Pickles

Questions?

Thank you!

gtkcyber.com