

SCIKIT LEARN

CSE 599 Prototyping Interactive Systems | Lecture 15 | May 21

Jon Froehlich • Jasper O'Leary (TA)

TODAY

LEARNING GOALS

Using **machine learning toolkits** like scikit-learn

The **scikit-learn** ML framework

How the **K-Nearest Neighbor (kNN)** classification model works

How the **Support Vector Machine (SVM)** classification model works

Briefly: **feature selection, feature scaling, parameter tuning**

ML TOOLKITS: LANGUAGES AND LIBRARIES

scikit-learn
Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification
Identifying to which category an object belongs to.
Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...

Regression
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...

Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ...

Dimensionality reduction
Reducing the number of random variables to consider.
Applications: Visualization, Increased efficiency.
Algorithms: PCA, feature selection, non-negative matrix factorization.

Model selection
Comparing, validating and choosing parameters and models.
Goal: Improved accuracy via parameter tuning.
Modules: grid search, cross validation, metrics.

Preprocessing
Feature extraction and normalization.
Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.

Scikit-Learn

Python-based library for classification, regression, clustering, dimensionality reduction, model selection, preprocessing data, etc.

Weka 3: Data Mining Software in Java

Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

Found only on the islands of New Zealand, the Weka is a flightless bird with an inquisitive nature. The name is pronounced like [this](#), and the bird sounds like [this](#).

Weka is open source software issued under the [GNU General Public License](#).

We have put together several free online courses that teach machine learning and data mining using Weka. Check out the [website for the courses](#) for details on when and how to enrol. The videos for the courses are available [on YouTube](#).

Yes, it is possible to apply Weka to [big data!](#)

Getting started	Further information	Developers
<ul style="list-style-type: none"> Requirements Download Documentation FAQ Getting Help 	<ul style="list-style-type: none"> Citing Weka Datasets Related Projects Miscellaneous Code Getting Help 	<ul style="list-style-type: none"> Development History Subversion Contributors Commercial licenses

Weka

Java-based library for pre-processing, classification, regression, clustering, and visualizing data. Used to be the standard. People now shifting to sci-kit learn, R, and cloud-based APIs.

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and Mac OS. To download R, please choose your preferred CRAN mirror. If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

News

- R version 3.5.0 (by In Hybrid) pre-release version will appear starting Friday 2018-03-23. Final release is scheduled for Monday 2018-04-23.
- R version 3.4.4 (Econometrics In Lyon) has been released on 2018-03-15.
- useR! 2018 (July 10 - 13 in Brisbane) is open for registration at <https://useR2018.r-project.org>.
- The R Journal Volume 9/2 is available.
- R version 3.3.3 (Another Cance) has been released on Monday 2017-03-06.
- useR! 2017 took place July 4 - 7 in Brussels <https://useR2017.jmssd.eu>.
- The R Logo is available for download in high-resolution PNG or SVG formats.

R

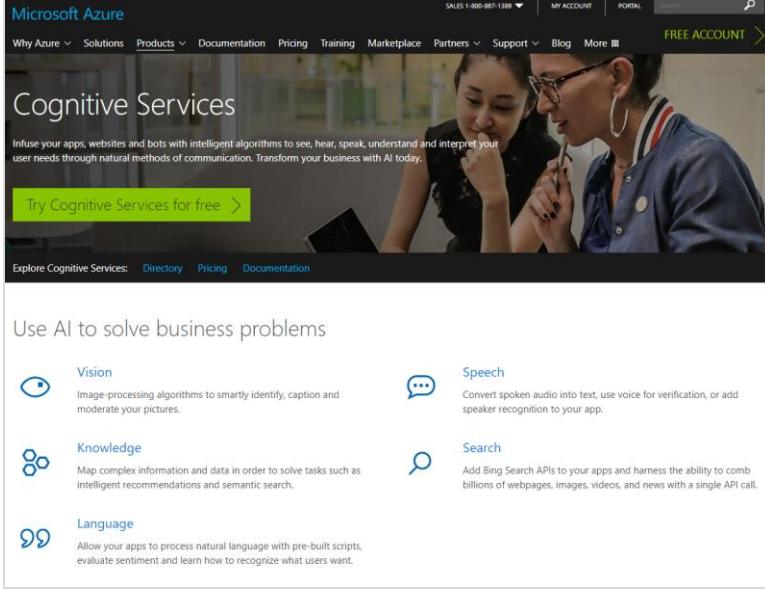
Documentation
Manuals
FAQ
The R Journal
Books

R

R arose as a statistical computing language but is now widely used in ML and has a number of well-respected ML packages.

ML TOOLKITS

ML TOOLKITS: CLOUD APIs



Cognitive Services

Infuse your apps, websites and bots with intelligent algorithms to see, hear, speak, understand and interpret your user needs through natural methods of communication. Transform your business with AI today.

Try Cognitive Services for free >

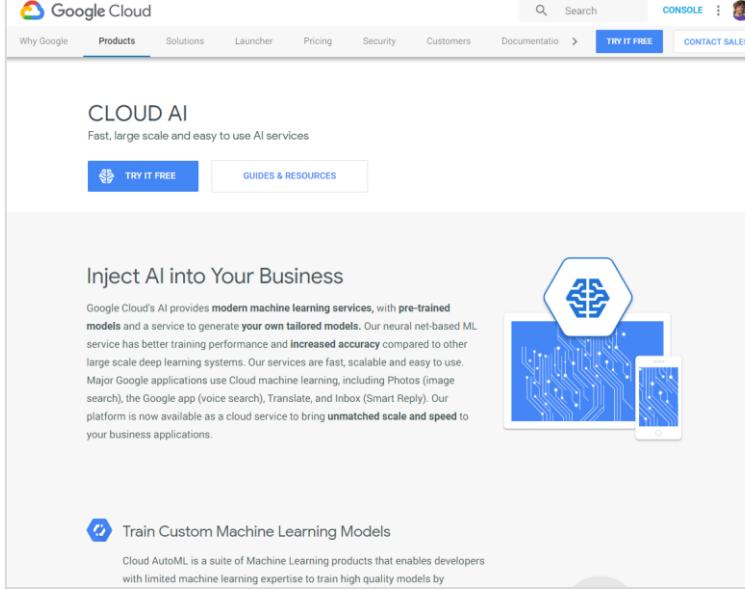
Explore Cognitive Services: [Directory](#) [Pricing](#) [Documentation](#)

Use AI to solve business problems

- Vision**
Image-processing algorithms to smartly identify, caption and moderate your pictures.
- Speech**
Convert spoken audio into text; use voice for verification, or add speaker recognition to your app.
- Knowledge**
Map complex information and data in order to solve tasks such as intelligent recommendations and semantic search.
- Language**
Allow your apps to process natural language with pre-built scripts, evaluate sentiment and learn how to recognize what users want.

Microsoft Cognitive Services

Cloud API for computer vision, NLP, speech recognition, knowledge representation, chatbots, and search.



CLOUD AI
Fast, large scale and easy to use AI services

TRY IT FREE GUIDES & RESOURCES

Inject AI into Your Business

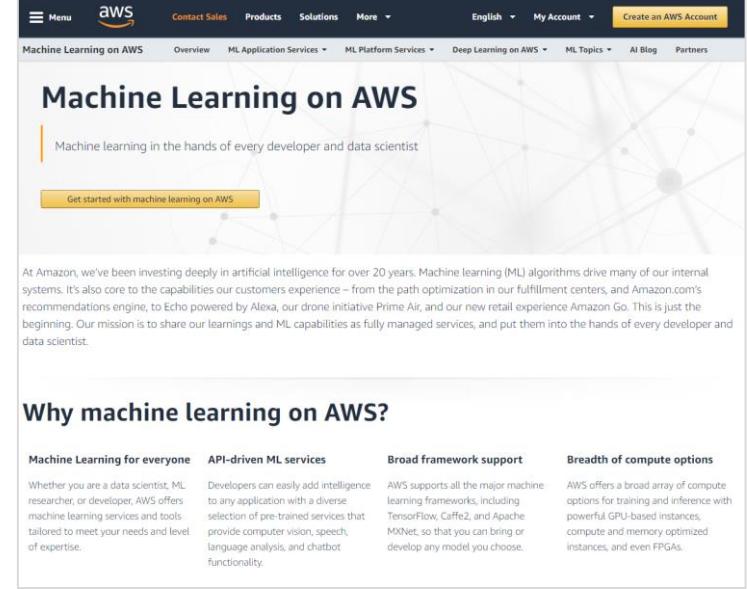
Google Cloud's AI provides **modern machine learning services**, with **pre-trained models** and a service to generate **your own tailored models**. Our neural net-based ML service has better training performance and **increased accuracy** compared to other large scale deep learning systems. Our services are fast, scalable and easy to use. Major Google applications use Cloud machine learning, including Photos (image search), the Google app (voice search), Translate, and Inbox (Smart Reply). Our platform is now available as a cloud service to bring **unmatched scale and speed** to your business applications.

Train Custom Machine Learning Models

Cloud AutoML is a suite of Machine Learning products that enables developers with limited machine learning expertise to train high quality models by

Google Cloud AI

Cloud API for large-scale machine learning, including video and image analysis, speech recognition, speech synthesis, NLP, and language translation



Machine Learning on AWS

Machine learning in the hands of every developer and data scientist

Get started with machine learning on AWS

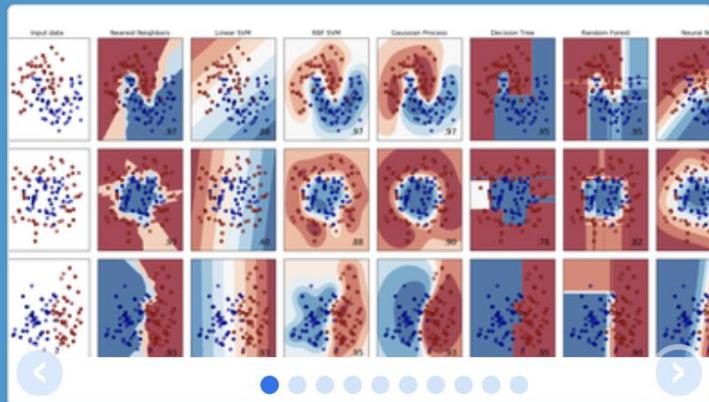
At Amazon, we've been investing deeply in artificial intelligence for over 20 years. Machine learning (ML) algorithms drive many of our internal systems. It's also core to the capabilities our customers experience – from the path optimization in our fulfillment centers, and Amazon.com's recommendations engine, to Echo powered by Alexa, our drone initiative Prime Air, and our new retail experience Amazon Go. This is just the beginning. Our mission is to share our learnings and ML capabilities as fully managed services, and put them into the hands of every developer and data scientist.

Why machine learning on AWS?

Machine Learning for everyone	API-driven ML services	Broad framework support	Breadth of compute options
Whether you are a data scientist, ML researcher, or developer, AWS offers machine learning services and tools tailored to meet your needs and level of expertise.	Developers can easily add intelligence to any application with a diverse selection of pre-trained services that provide computer vision, speech, language analysis, and chatbot functionality.	AWS supports all the major machine learning frameworks, including TensorFlow, Caffe2, and Apache MXNet, so that you can bring or develop any model you choose.	AWS offers a broad array of compute options for training and inference with powerful GPU-based instances, compute and memory optimized instances, and even FPGAs.

Amazon AWS ML Platform

Cloud API for general ML with image and video analysis, chatbots, and language services such as NLP, translation, transcription, and speech synthesis.



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random forest, ...

— Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ...

— Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, ...

— Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization.

— Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning

Modules: grid search, cross validation, metrics.

— Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.

Modules: preprocessing, feature extraction.

— Examples

SCIKIT-LEARN FEATURES INCLUDE

FEATURE EXTRACTION, SELECTION, & REDUCTION

Feature Extraction: helper methods for extracting features from image and text data.

Feature Selection: for identifying meaningful attributes from which to create supervised models.

Dimensionality Reduction: for reducing the number of features (e.g., Principal component analysis)

ML MODELS

Clustering Algorithms: for grouping unlabeled data such as KMeans.

Supervised Models: a vast array of supervised models, including naive bayes, neural networks, SVMs and decision trees.

Ensemble Methods: for combining the predictions of multiple supervised models.

EVALUATION

Cross Validation: for estimating the performance of supervised models on unseen data.

Parameter Tuning: for getting the most out of supervised models.

Datasets: for test datasets and for generating datasets with specific properties for investigating model behavior.

SCIKIT LEARN GETTING STARTED

[Home](#) [Installation](#) [Documentation](#) [Examples](#) [Google Custom Search](#) [Search](#)

An introduction to machine learning with scikit-learn

Section contents

In this section, we introduce the machine learning vocabulary that we use throughout scikit-learn and give a simple learning example.

Machine learning: the problem setting

In general, a learning problem considers a set of n samples of data and then tries to predict properties of unknown data. If each sample is more than a single number and, for instance, a multi-dimensional entry (aka multivariate data), it is said to have several attributes or features.

We can separate learning problems in a few large categories:

- supervised learning, in which the data comes with additional attributes that we want to predict (Click here to go to the scikit-learn supervised learning page). This problem can be either:
 - classification: samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data. An example of classification problem would be the handwritten digit recognition example, in which the aim is to assign each input vector to one of a finite number of discrete categories. Another way to think of classification is as a discrete (as opposed to continuous) form of supervised learning where one has a limited number of categories and for each of the n samples provided, one is to try to label them with the correct category or class.
 - regression: if the desired output consists of one or more continuous variables, then the task is called regression. An example of a regression problem would be the prediction of the length of a salmon as a function of its age and weight.
- unsupervised learning, in which the training data consists of a set of input vectors x without any corresponding target values. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation.

[Home](#) [Installation](#) [Documentation](#) [Examples](#) [Google Custom Search](#) [Search](#)

User Guide

1. Supervised learning

- ▶ 1.1. Generalized Linear Models
- ▶ 1.2. Linear and Quadratic Discriminant Analysis
- 1.3. Kernel ridge regression
- ▶ 1.4. Support Vector Machines
- ▶ 1.5. Stochastic Gradient Descent
- ▶ 1.6. Nearest Neighbors
- ▶ 1.7. Gaussian Processes
- 1.8. Cross decomposition
- ▶ 1.9. Naive Bayes
- ▶ 1.10. Decision Trees
- ▶ 1.11. Ensemble methods
- ▶ 1.12. Multiclass and multilabel algorithms
- ▶ 1.13. Feature selection
- ▶ 1.14. Semi-Supervised
- 1.15. Isotonic regression
- 1.16. Probability calibration

[Home](#) [Installation](#) [Documentation](#) [Examples](#) [Google Custom Search](#) [Search](#)

Examples

General examples

General-purpose and introductory examples for the scikit.

Examples

- General examples
- Examples based on real world datasets
- Biclustering
- Calibration
- Classification
- Clustering
- Covariance estimation
- Cross decomposition
- Dataset examples
- Decomposition
- Ensemble methods
- Tutorial exercises
- Feature Selection
- Gaussian Process for Machine Learning
- Generalized Linear Models
- Manifold learning
- Gaussian Mixture Models
- Model Selection
- Multicategory methods
- Nearest Neighbors
- Neural Networks
- Preprocessing
- Semi Supervised Classification
- Support Vector Machines

Plotting Cross-Validated Predictions  **Concatenating multiple feature extraction methods**  **Pipeline: chaining a PCA and a logistic regression** 

Isotonic Regression  **Imputing missing values before building an estimator**  **Face completion with a multi-output estimators** 

Quick Start Tutorial

Introduces some basic vocabulary and walks through a simple example

User Guide

Goes through each major component of the sci-kit library from supervised and unsupervised models to model selection and evaluation.

Examples

Lots of different examples of using, visualizing, and evaluating models

SCIKIT LEARN CLASSIFICATION FRAMEWORK

```
# Create classifier (often abbreviated clf for classifier)
clf = MyClassifier()

# Train the classifier with the training data and labels
clf.fit(training_data, class_labels)
```

SCIKIT LEARN CLASSIFICATION FRAMEWORK

```
# Create classifier (often abbreviated clf for classifier)
clf = MyClassifier()
```

```
# Train the classifier with the training data and labels
clf.fit(training_data, class_labels)
```

Training Data Class Labels

mag_p.max	mag_p_avg_peak	mag_p_std	gesture
198.900715	124.898957	66.913586	Midair Zorro 'Z'
185.104321	111.061818	55.435780	Midair Zorro 'Z'
155.287426	155.287426	50.887685	Midair Zorro 'Z'
147.918587	147.918587	49.347788	Midair Zorro 'Z'
187.198179	120.283442	48.289167	Baseball Throw
243.532742	243.532742	55.737110	Baseball Throw
290.012557	190.527641	66.258158	Baseball Throw
311.963850	180.246706	81.009928	Baseball Throw

SCIKIT LEARN CLASSIFICATION FRAMEWORK

```
# Create classifier (often abbreviated clf for classifier)
clf = MyClassifier()

# Train the classifier with the training data and labels
clf.fit(training_data, class_labels)

# Provide input test features and obtain a prediction
clf_prediction = clf.predict(test_features)
```

Test Data

mag_p.max	mag_p_avg_peak	mag_p_std
150.356560	100.060016	54.067795

SCIKIT LEARN CLASSIFICATION FRAMEWORK

```
# Create classifier (often abbreviated clf for classifier)
clf = MyClassifier()

# Train the classifier with the training data and labels
clf.fit(training_data, class_labels)

# Provide input test features and obtain a prediction
clf_prediction = clf.predict(test_features)

# Print out prediction (prints out predicted class label)
print(clf_prediction[0])
```

>>> “Baseball Throw”

SCIKIT LEARN FRAMEWORK: SINGLE INPUT ROW

```
# Create classifier
clf = MyClassifier()

# Train the classifier with the training data & labels
clf.fit(training_data, class_labels)

# Provide input test features and obtain a prediction
clf_prediction = clf.predict(test_features)

# Print out prediction
print(clf_prediction[0])
```

>>> "Baseball Throw"

Test Data

mag_p.max	mag_p_avg_peak	mag_p_std
150.356560	100.060016	54.067795

Training Data

mag_p.max	mag_p_avg_peak	mag_p_std	gesture
198.900715	124.898957	66.913586	Midair Zorro 'Z'
185.104321	111.061818	55.435780	Midair Zorro 'Z'
155.287426	155.287426	50.887685	Midair Zorro 'Z'
147.918587	147.918587	49.347788	Midair Zorro 'Z'
187.198179	120.283442	48.289167	Baseball Throw
243.532742	243.532742	55.737110	Baseball Throw
290.012557	190.527641	66.258158	Baseball Throw
311.963850	180.246706	81.009928	Baseball Throw

Class Labels

SCIKIT LEARN FRAMEWORK: MULTIPLE ROWS

```
# Create classifier
clf = MyClassifier()

# Train the classifier with the training data & labels
clf.fit(training_data, class_labels)

# Provide input test features and obtain a prediction
clf_prediction = clf.predict(test_features_array)

# Print out prediction
print(clf_prediction)
```

```
>>> ["Baseball Throw",
"Midair Zorro 'Z'", "Bunny Hops", "Forehand
Tennis"]
```

Test Data

mag_p.max	mag_p_avg_peak	mag_p_std
150.356560	100.060016	54.067795
144.241369	106.726920	49.411597
129.416905	105.980495	49.794466
157.691803	122.343194	56.925764

Training Data

gesture
Midair Zorro 'Z'
Baseball Throw
Baseball Throw
Baseball Throw
Baseball Throw

SWAPPING OUT MODELS

```
clf = svm.SVC(kernel='linear')
clf.fit(training_data, class_labels)
svm_prediction = clf.predict(test_features)
```

```
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(training_data, class_labels)
knn_prediction = clf.predict(test_features)
```

```
clf = DecisionTreeClassifier(random_state=0)
clf.fit(training_data, class_labels)
tree_prediction = clf.predict(test_features)
```

```
clf = MLPClassifier(solver='lbfgs', random_state=0, hidden_layer_sizes=[100])
clf.fit(training_data, class_labels)
tree_prediction = clf.predict(test_features)
```

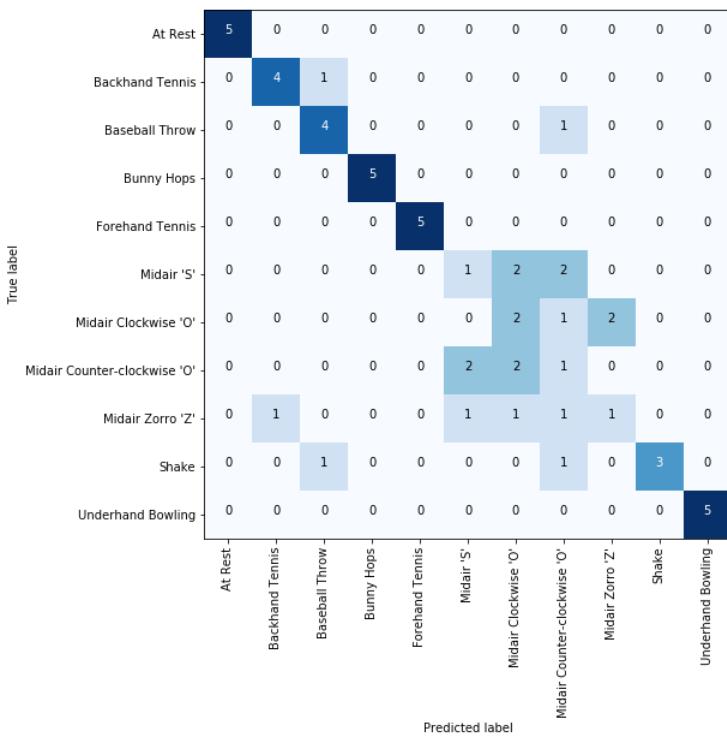
MODELS

DECISION TREE CLASSIFIER RESULTS

```
clf = DecisionTreeClassifier(random_state=0)
clf.fit(training_data, class_labels)
tree_prediction = clf.predict(test_features)
```

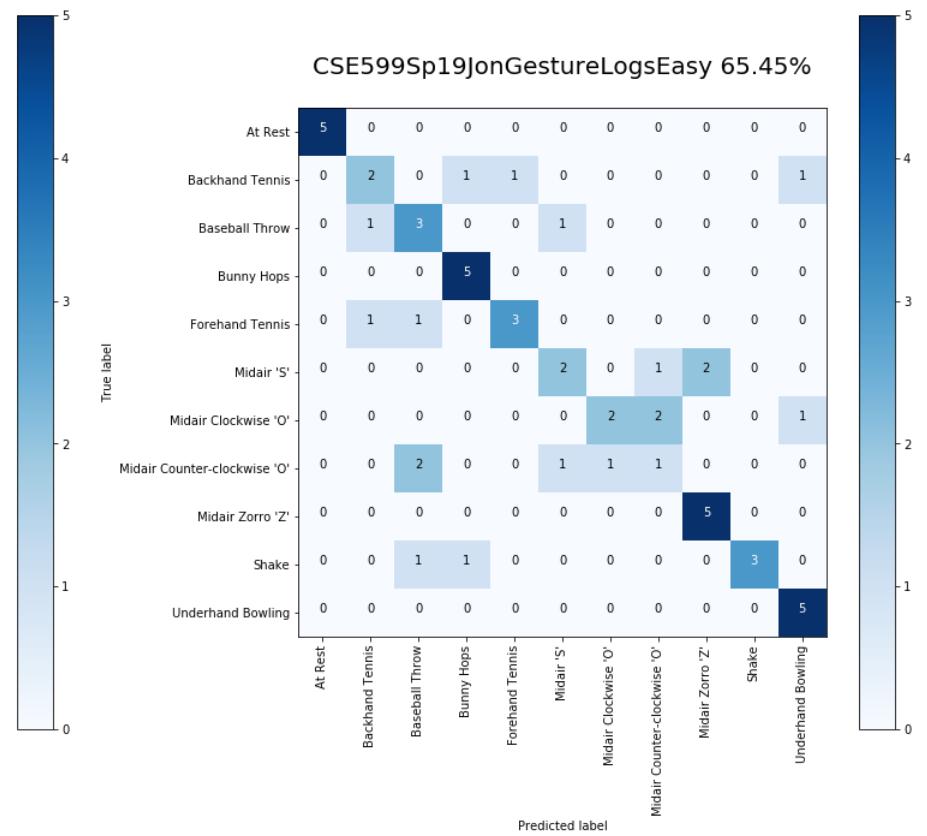
Jon Gestures (Hard): 65.45%

CSE599Sp19JonGestureLogsHarder 65.45%



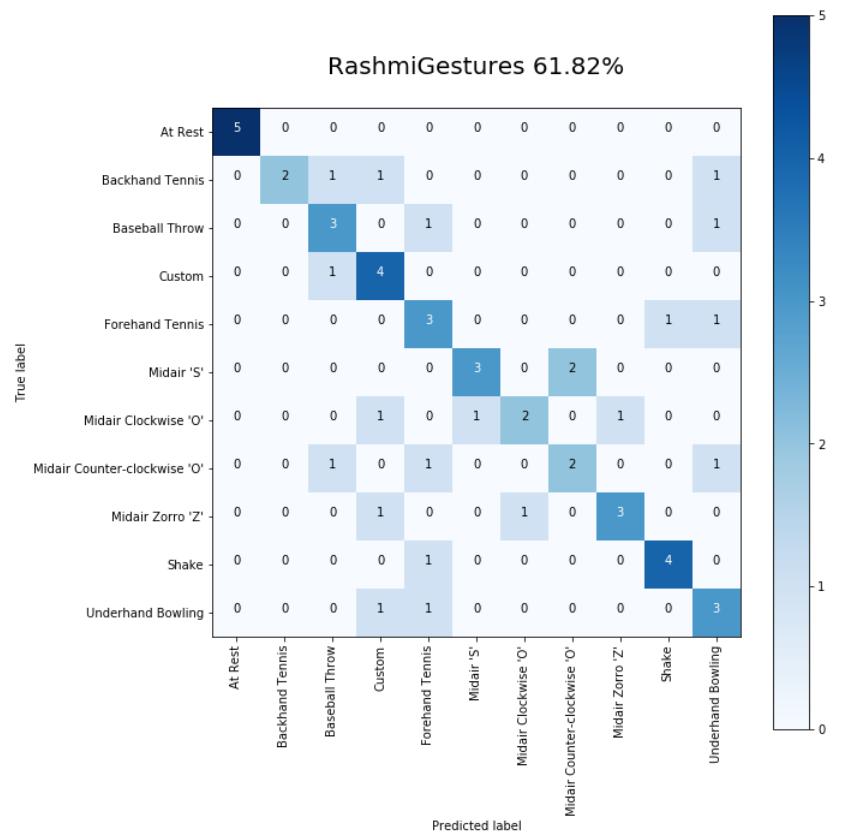
Jon Gestures (Easy): 65.45%

CSE599Sp19JonGestureLogsEasy 65.45%



Rashmi: 61.82%

RashmiGestures 61.82%

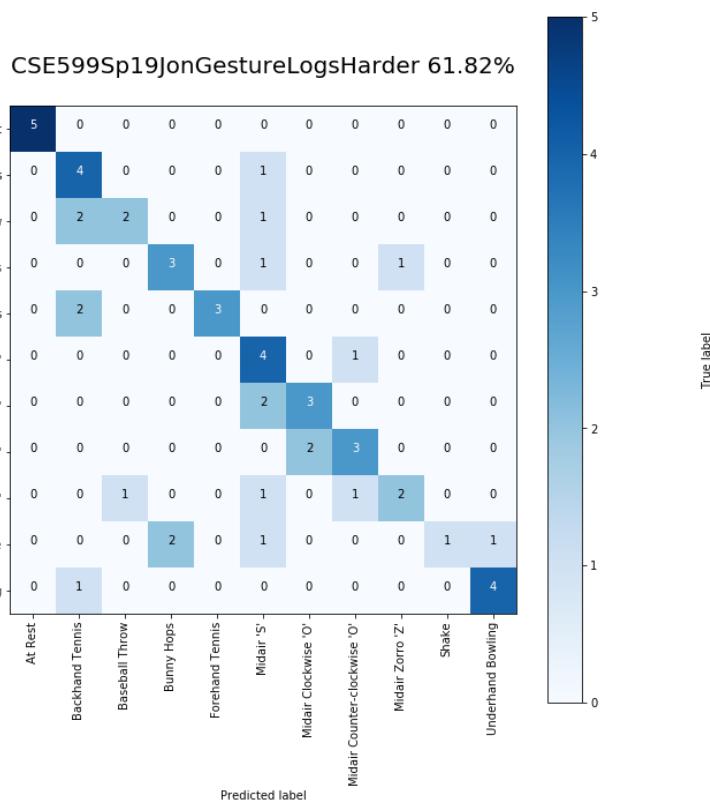


MODELS

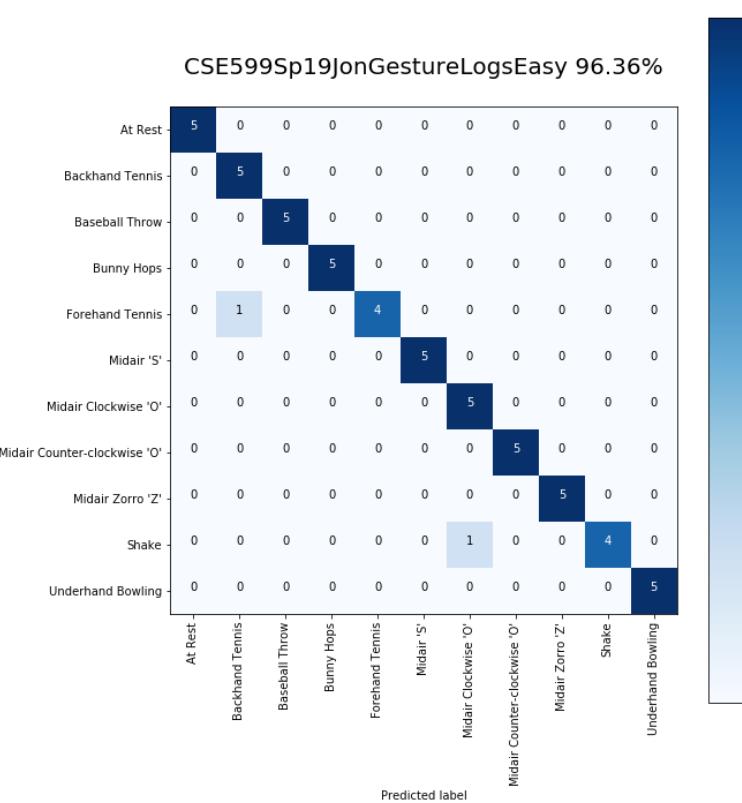
KNN RESULTS

```
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(training_data, class_labels)
knn_prediction = clf.predict(test_features)
```

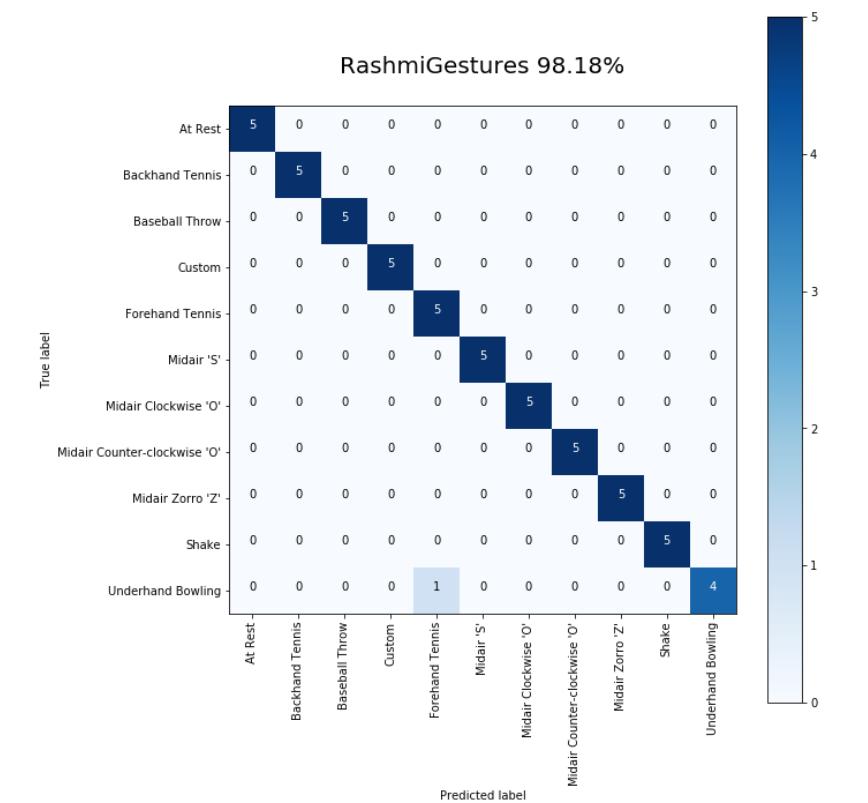
Jon Gestures (Hard): 61.82%



Jon Gestures (Easy): 96.36%



Rashmi: 98.18%

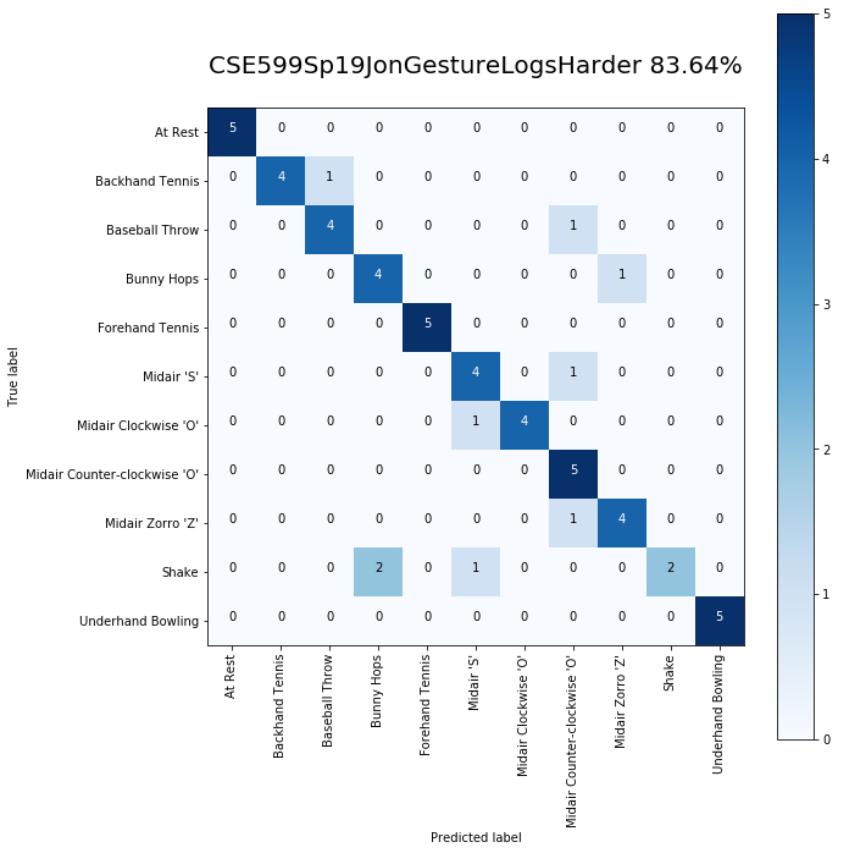


MODELS

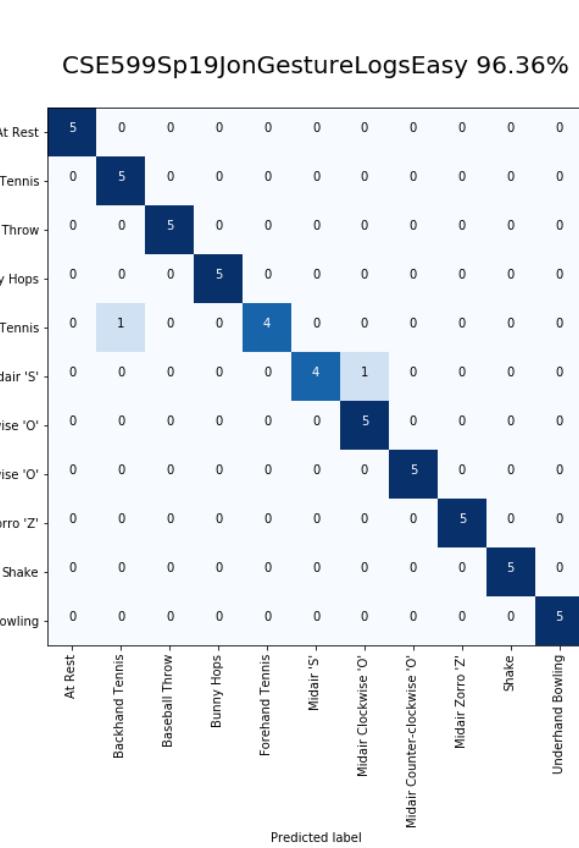
NEURAL NET CLASSIFIER RESULTS

```
clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=[100])
clf.fit(training_data, class_labels)
tree_prediction = clf.predict(test_features)
```

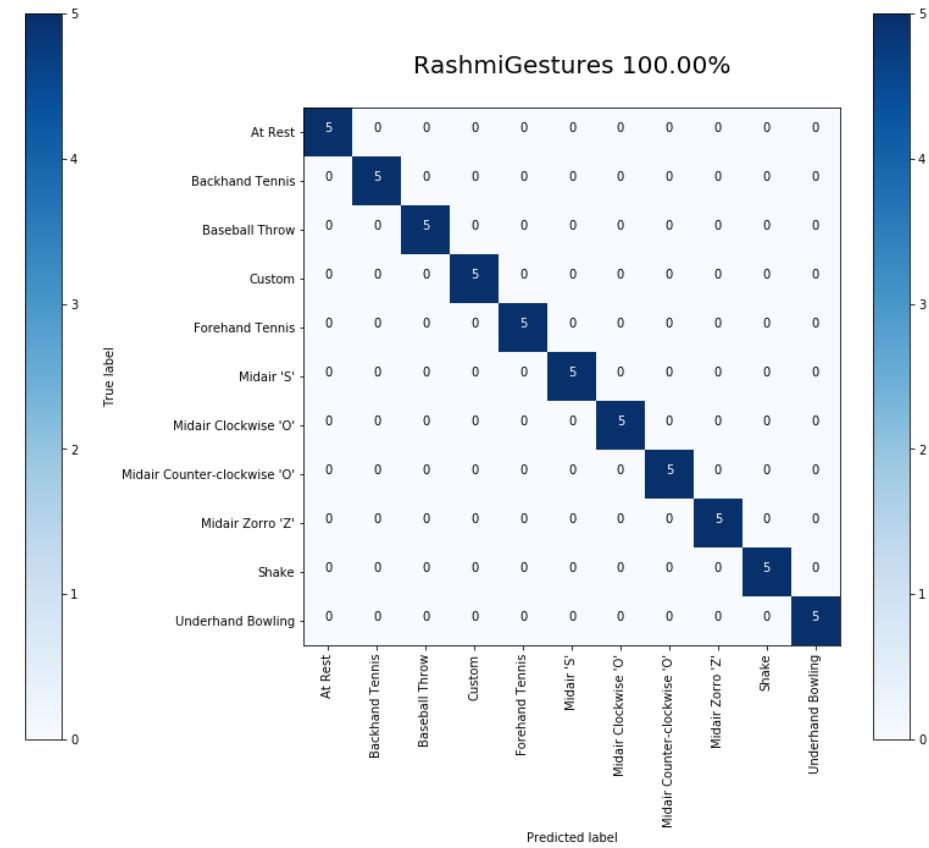
Jon Gestures (Hard): 83.64%



Jon Gestures (Easy): 96.36%



Rashmi: 100%

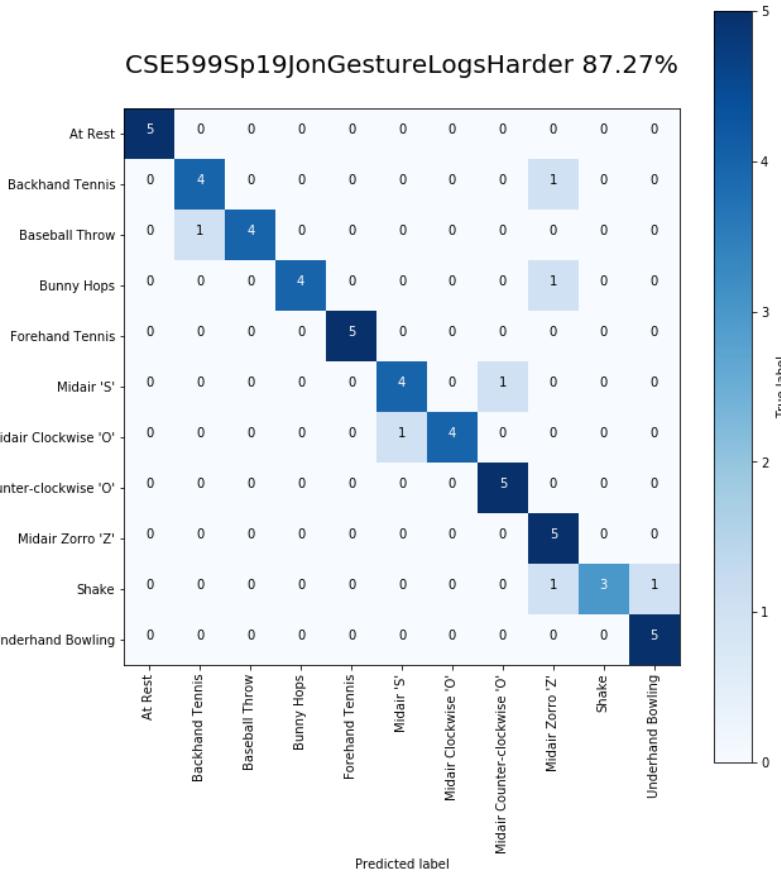


MODELS

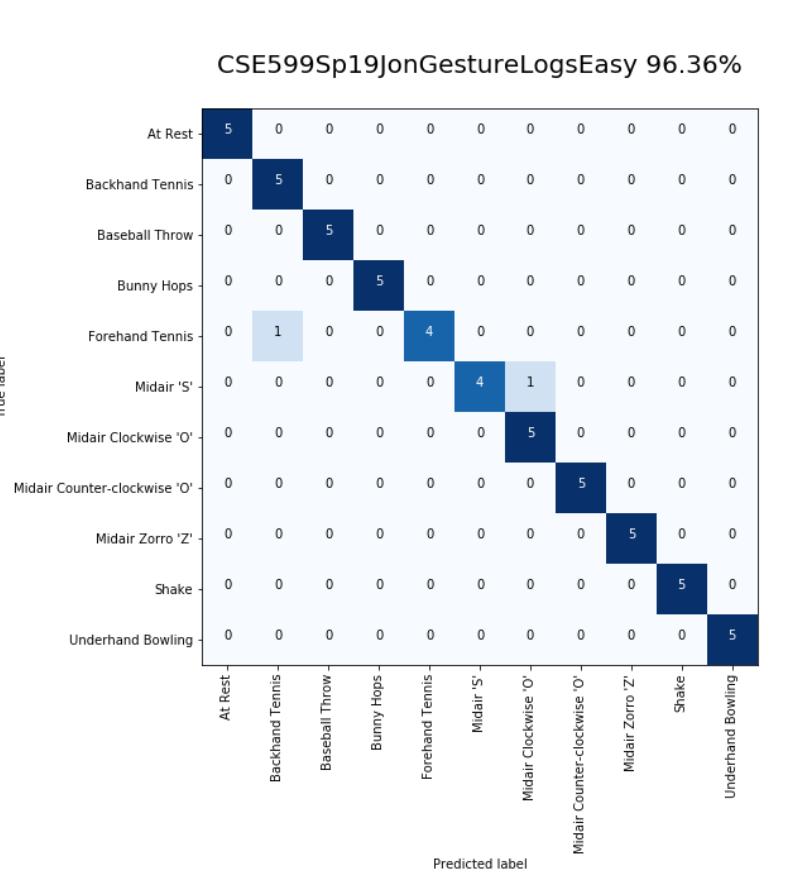
SVM RESULTS

```
clf = svm.SVC(kernel='linear')
clf.fit(training_data, class_labels)
svm_prediction = clf.predict(test_features)
```

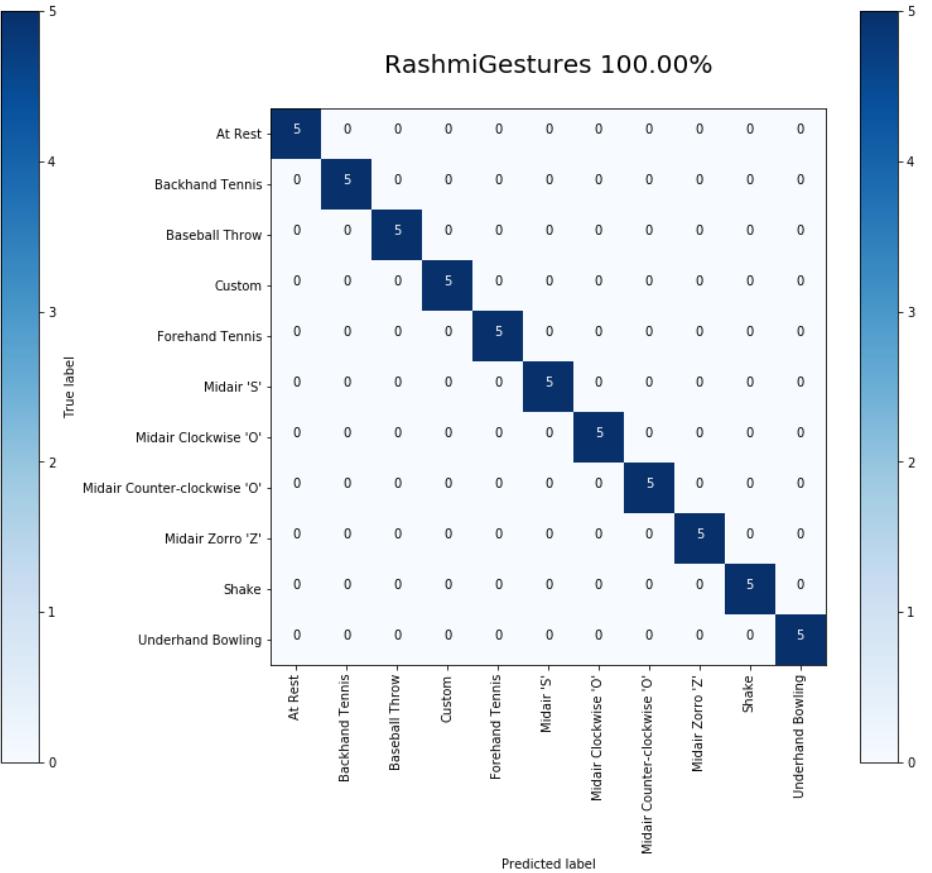
Jon Gestures (Hard): 87.27%



Jon Gestures (Easy): 96.36%



Rashmi: 100%



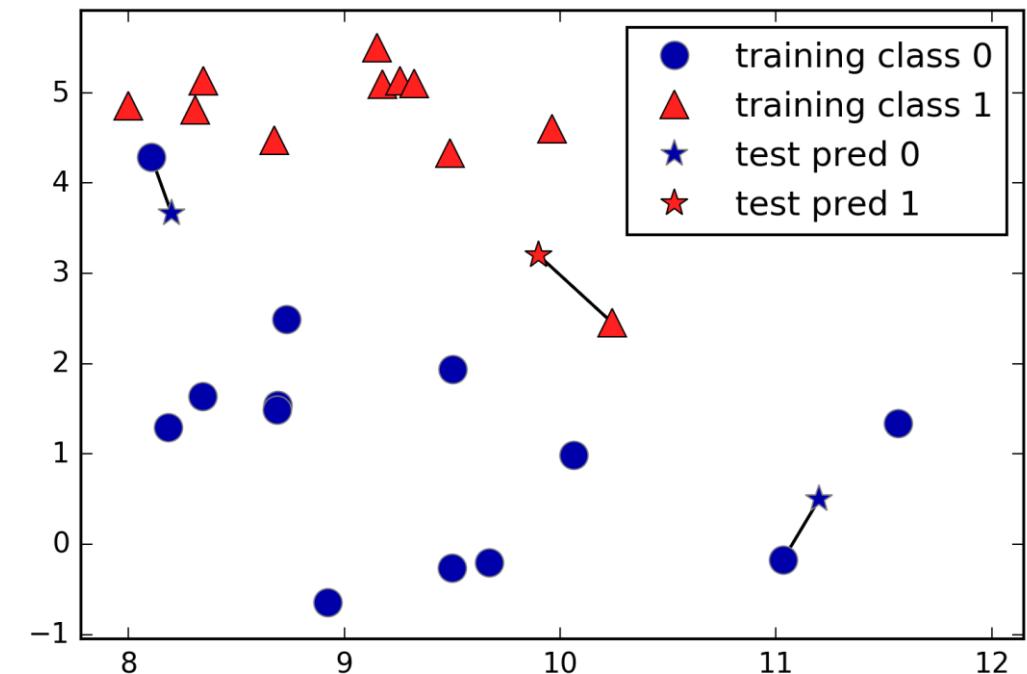
K-NEAREST NEIGHBORS (kNN)

Arguably the simplest ML algorithm

Model consists of only “storing” the training set

To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset—its “nearest neighbors.”

Simplest kNN Model Where k=1



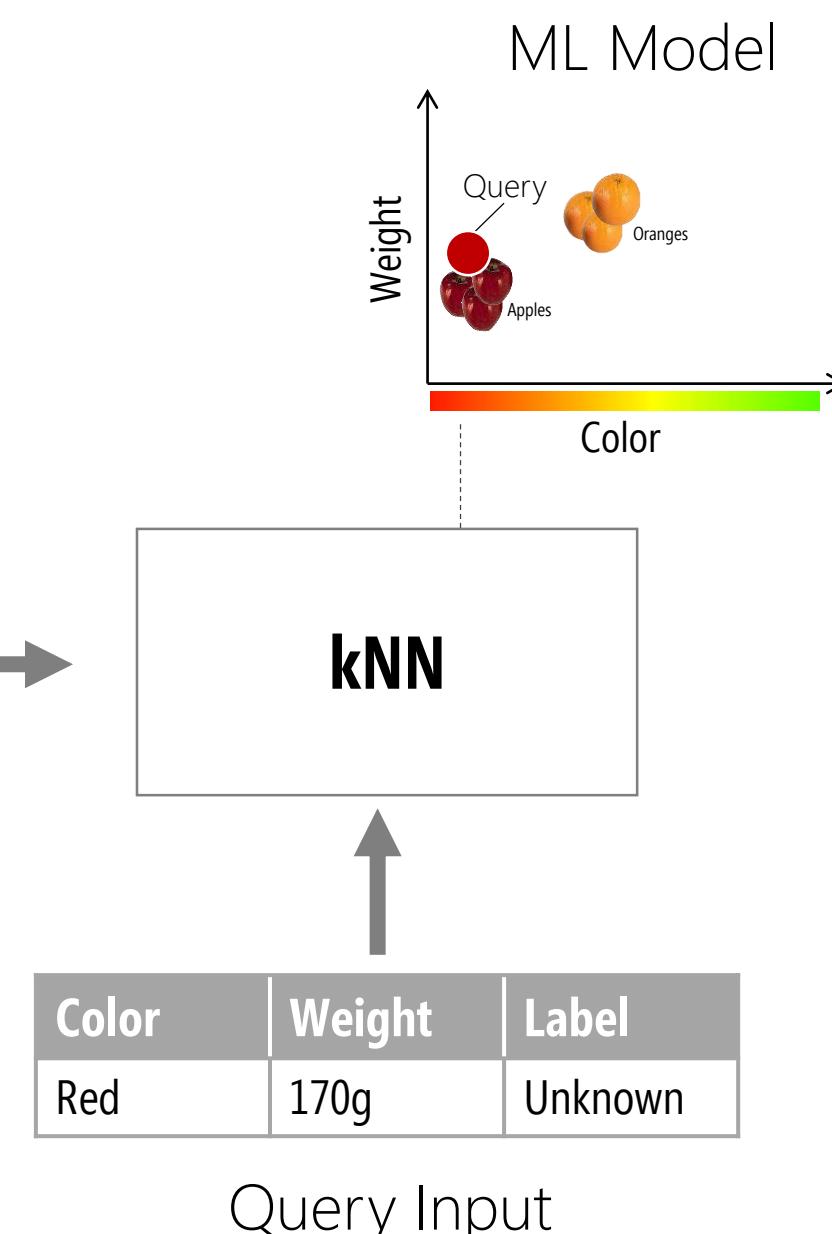
When $k=1$, the prediction is always the closest training point

K-NEAREST NEIGHBORS

KNN WHERE K=1

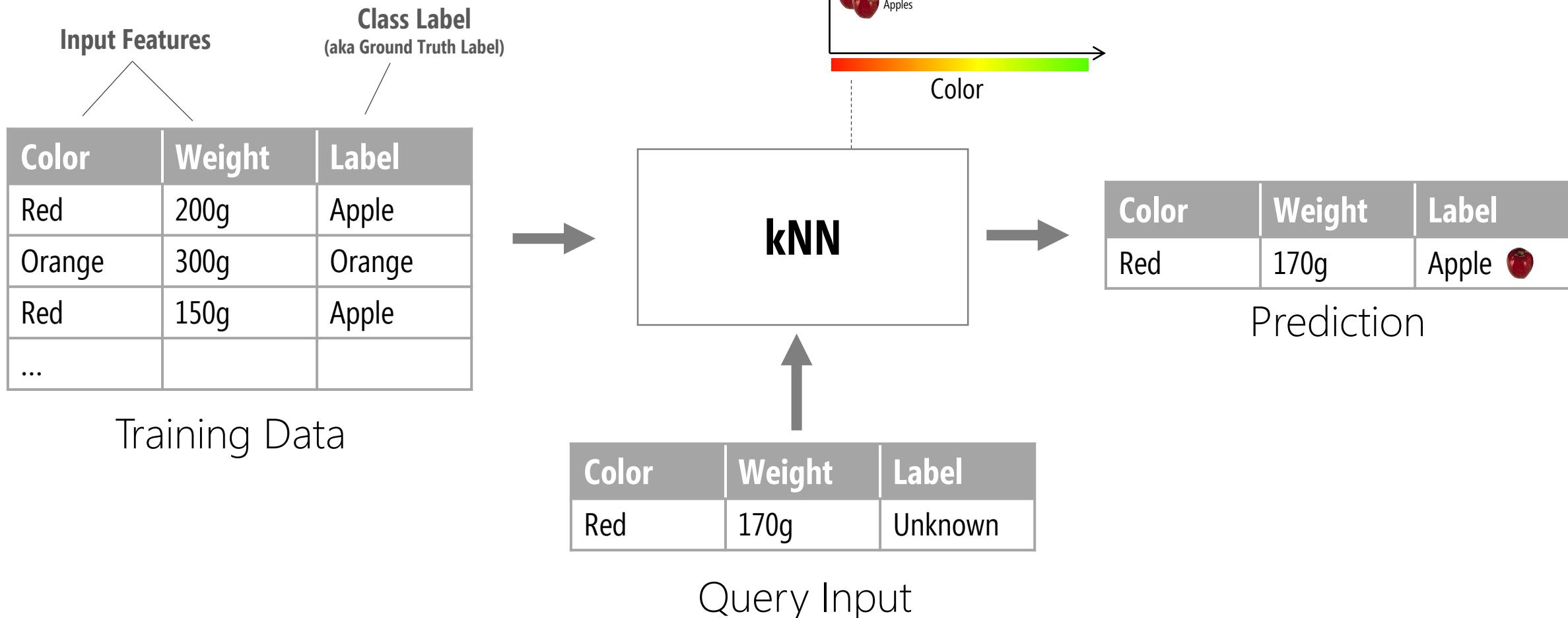
Input Features		Class Label (aka Ground Truth Label)
Color	Weight	Label
Red	200g	Apple
Orange	300g	Orange
Red	150g	Apple
...		

Training Data



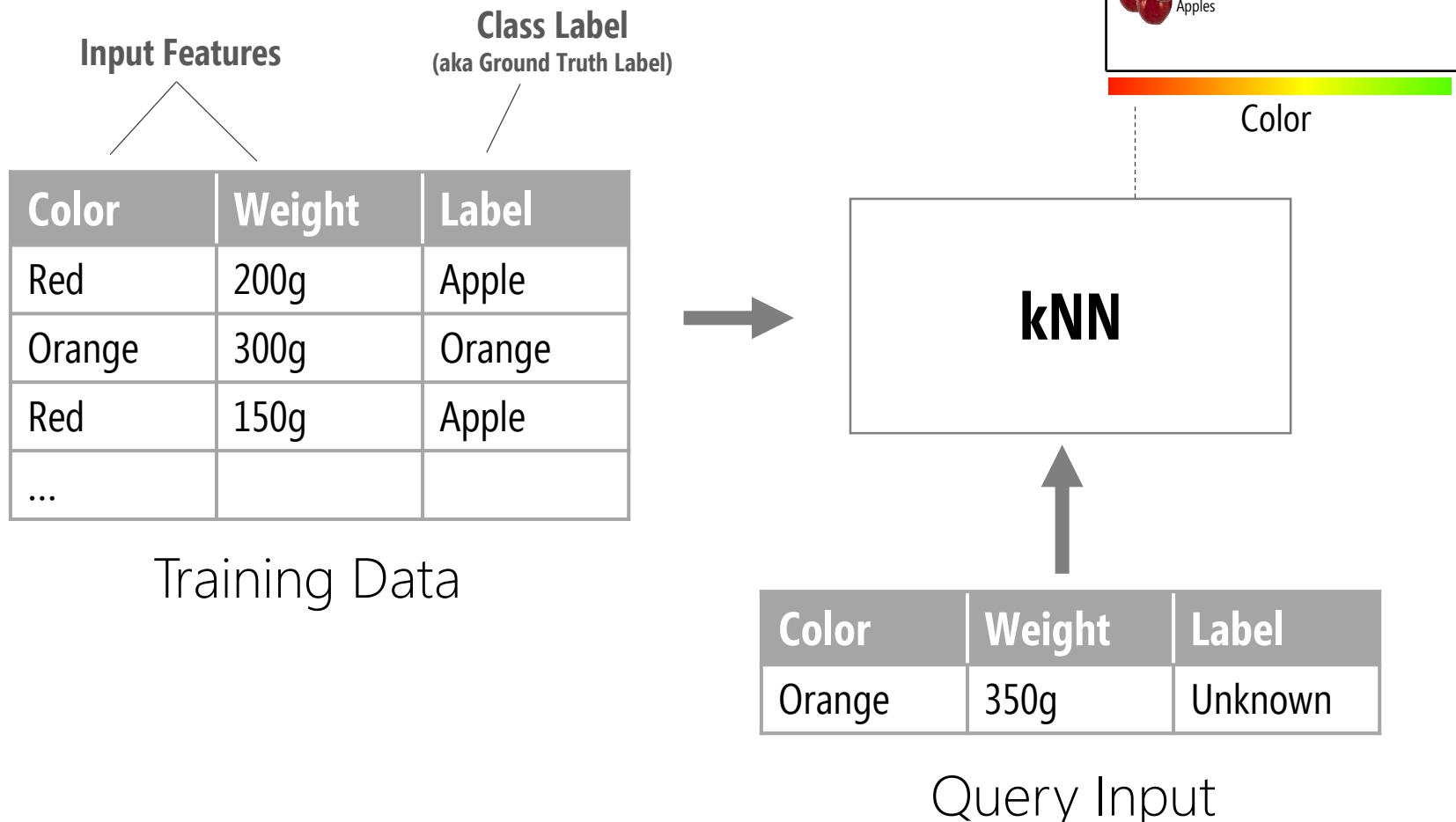
K-NEAREST NEIGHBORS

KNN WHERE K=1



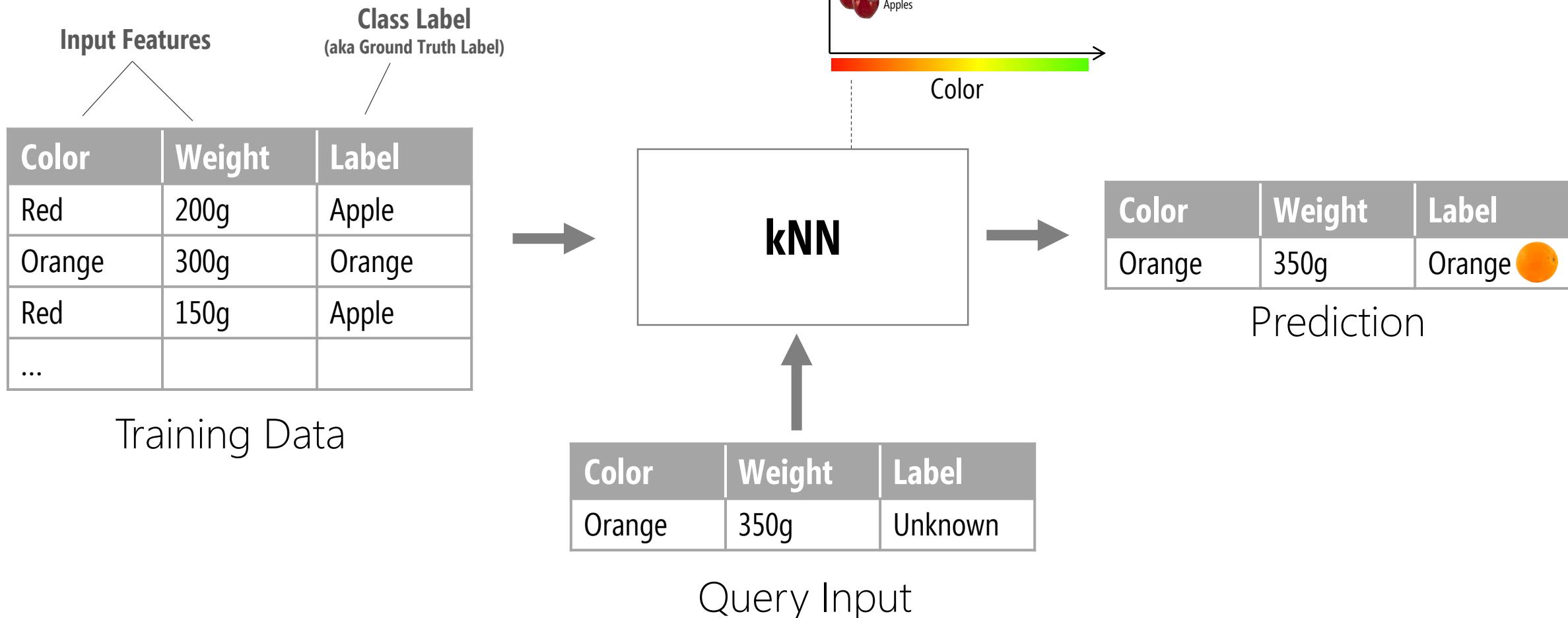
K-NEAREST NEIGHBORS

KNN WHERE K=1

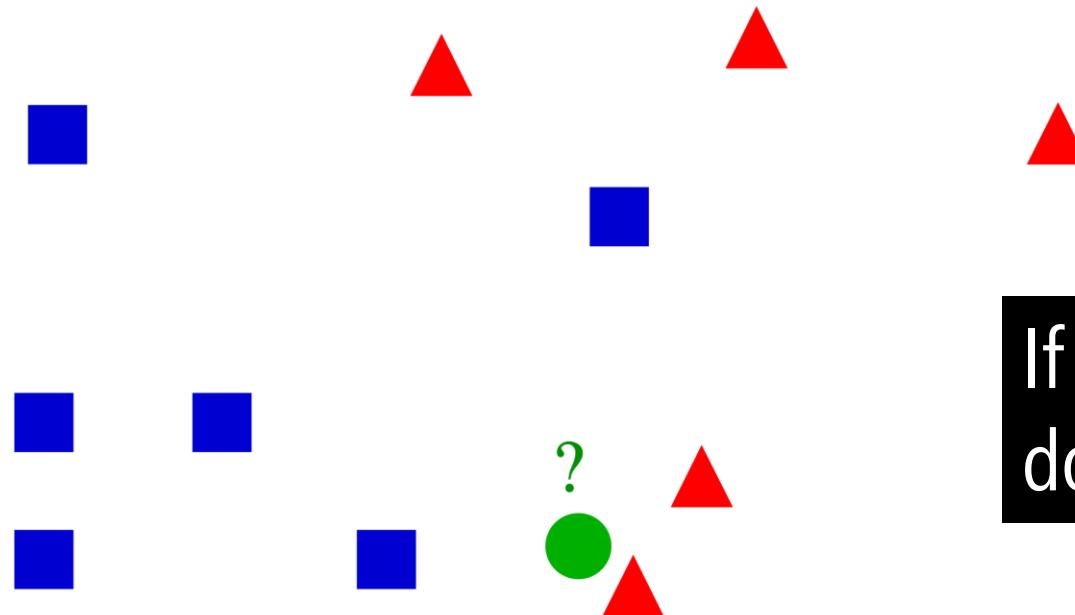


K-NEAREST NEIGHBORS

KNN WHERE K=1

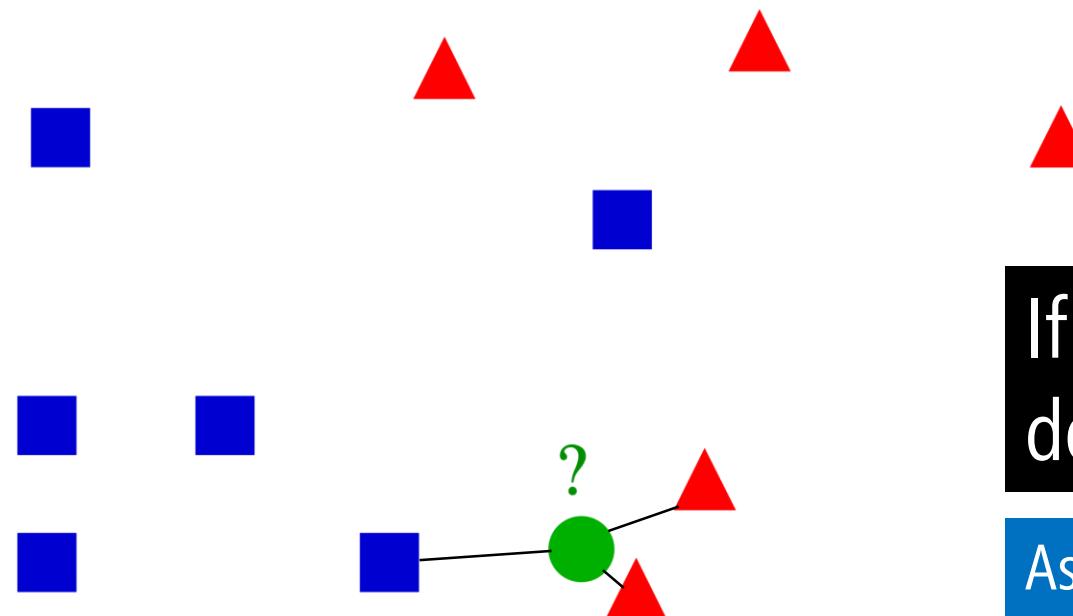


EXAMPLE KNN CLASSIFICATION



If $k=3$, what is the green dot classified as?

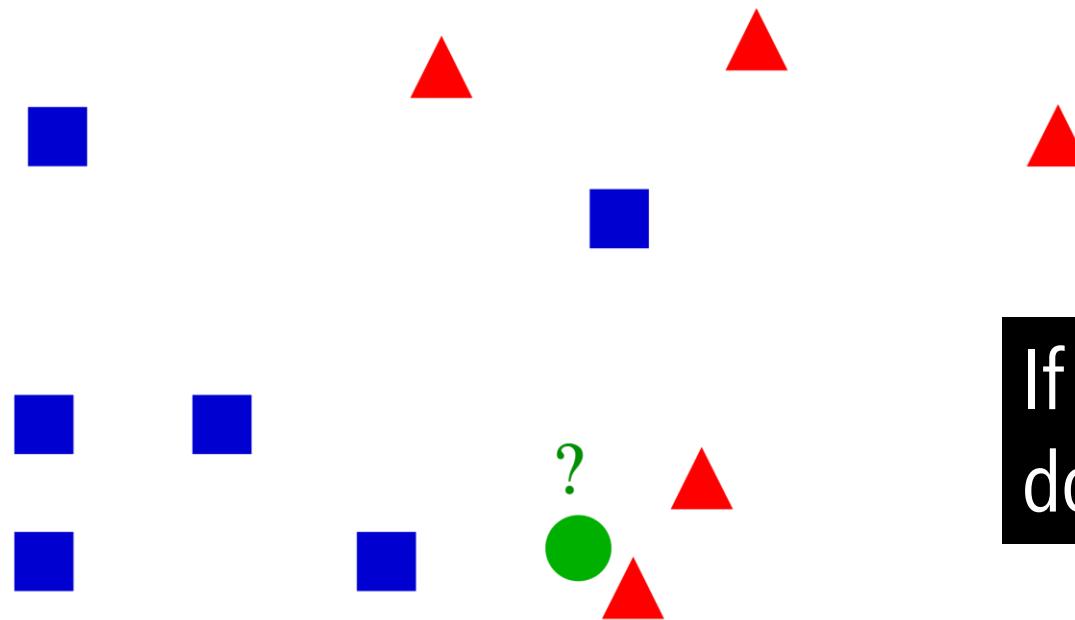
EXAMPLE KNN CLASSIFICATION



If $k=3$, what is the green dot classified as?

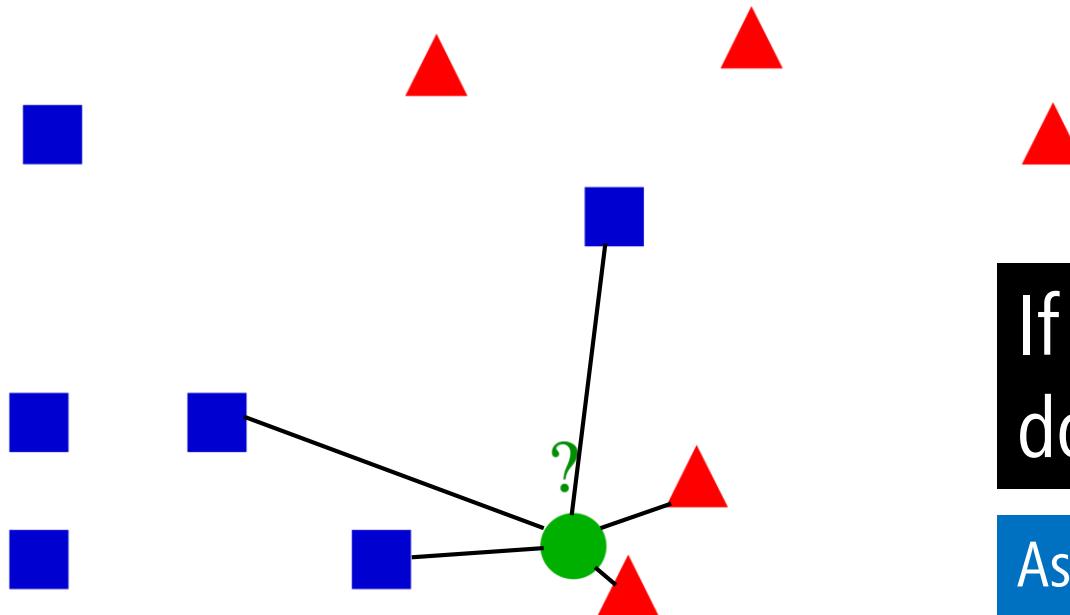
Assigned to the "triangle" class because of the $k=3$ closest neighbors, 2 are triangles and one is a square.

EXAMPLE KNN CLASSIFICATION



If $k=5$, what is the green dot classified as?

EXAMPLE KNN CLASSIFICATION

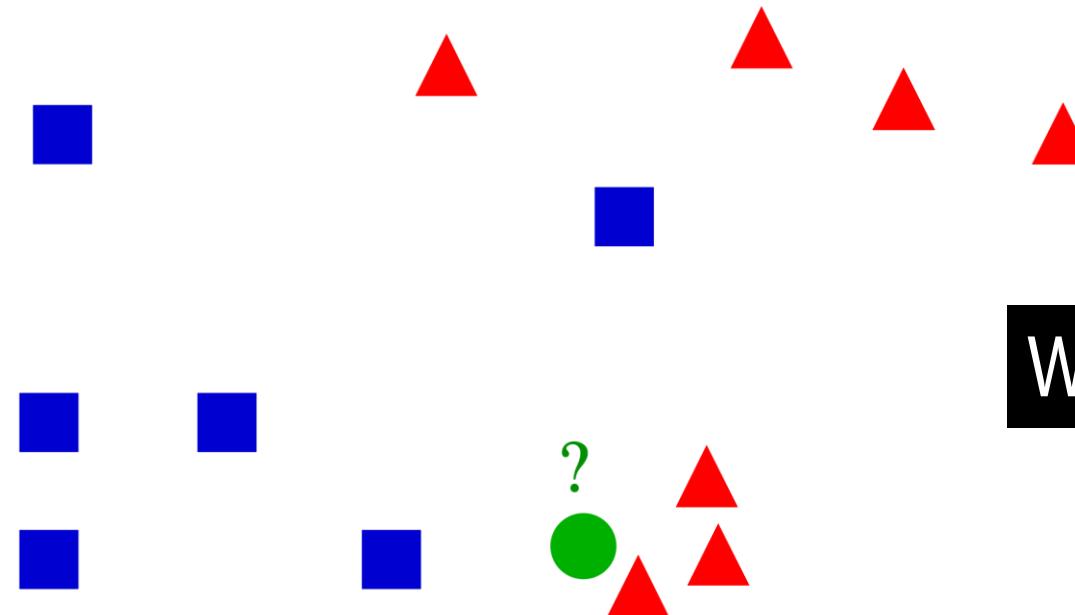


If $k=5$, what is the green dot classified as?

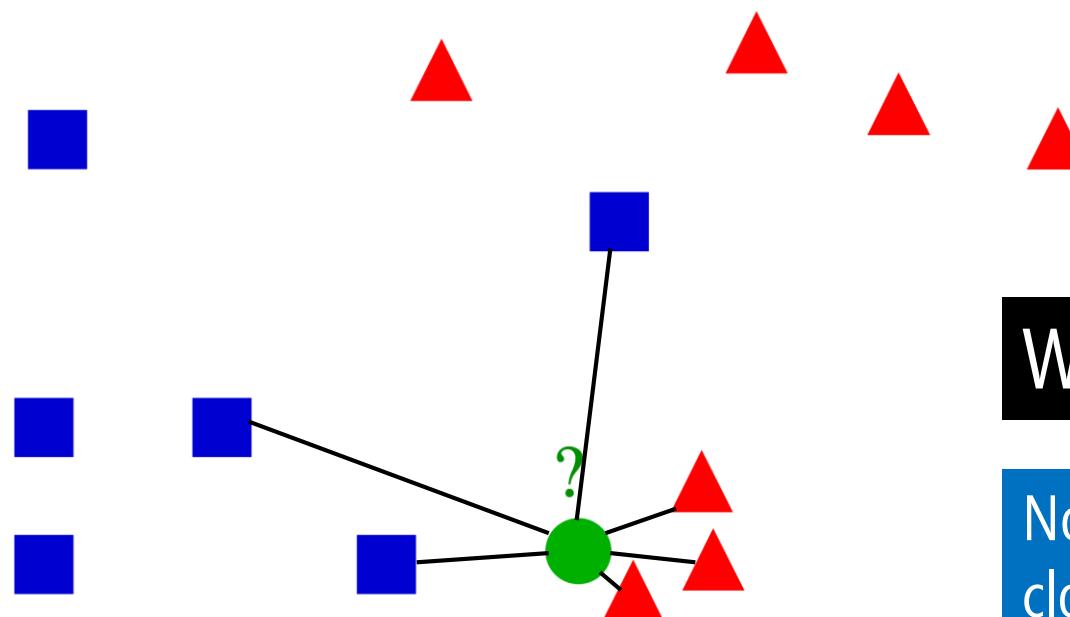
Assigned to the “square” class because of the $k=5$ closest neighbors, 3 are squares and two are triangles.

K-NEAREST NEIGHBORS

EXAMPLE KNN CLASSIFICATION



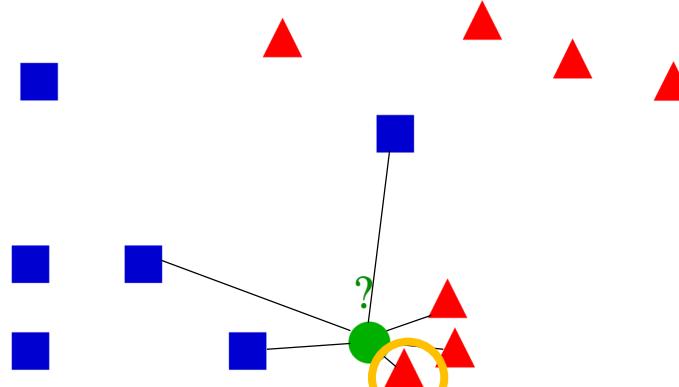
EXAMPLE KNN CLASSIFICATION



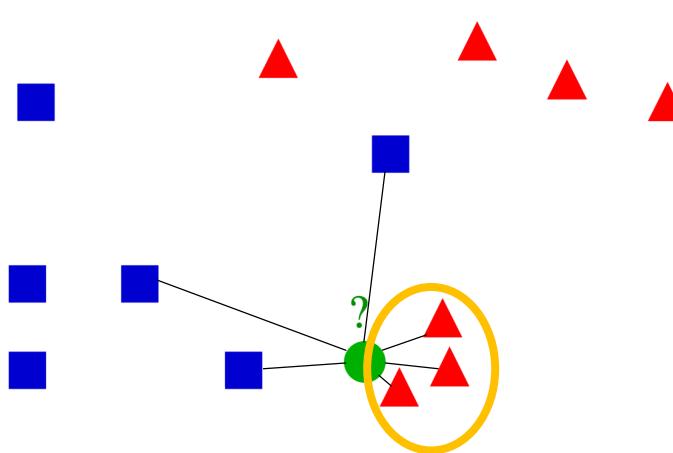
What happens if $k=6$?

Now we have a tie, the six closest neighbors are three blue squares & three blue triangles.

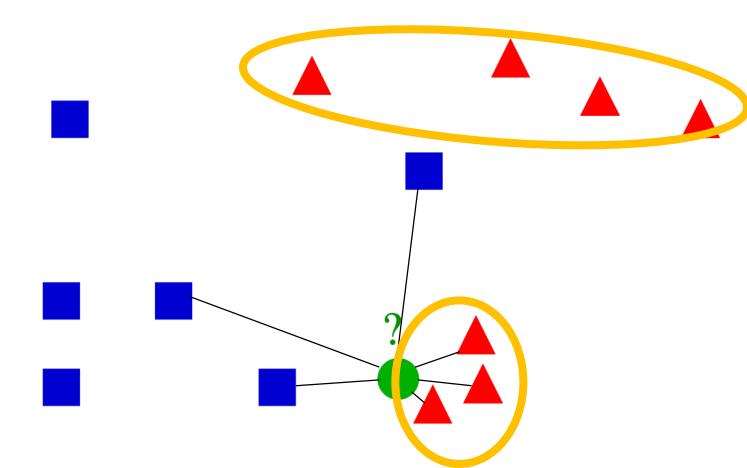
HOW DOES KNN BREAK TIES?



Pick the closest class (*i.e.*, degrade to a k=1)



Pick the class with average lowest distance within the k-neighbor cluster



Pick the class based on some weighting (*e.g.*, class that is observed most in dataset)

STRENGTHS AND WEAKNESSES

Strengths

Model is easy to understand

Often gives reasonable performance for low-dimensionality datasets

Good baseline model to try

Weaknesses

Data needs to be preprocessed (scaled)

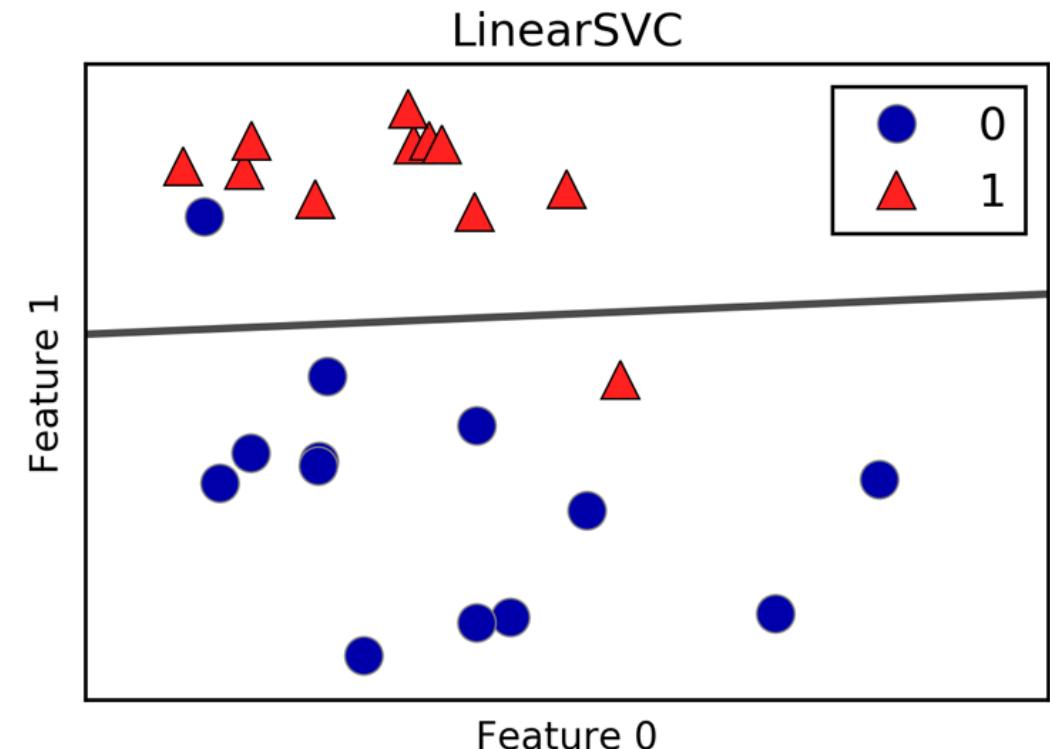
Model building is fast, but prediction can be slow for large datasets

Does not perform well for sparse datasets

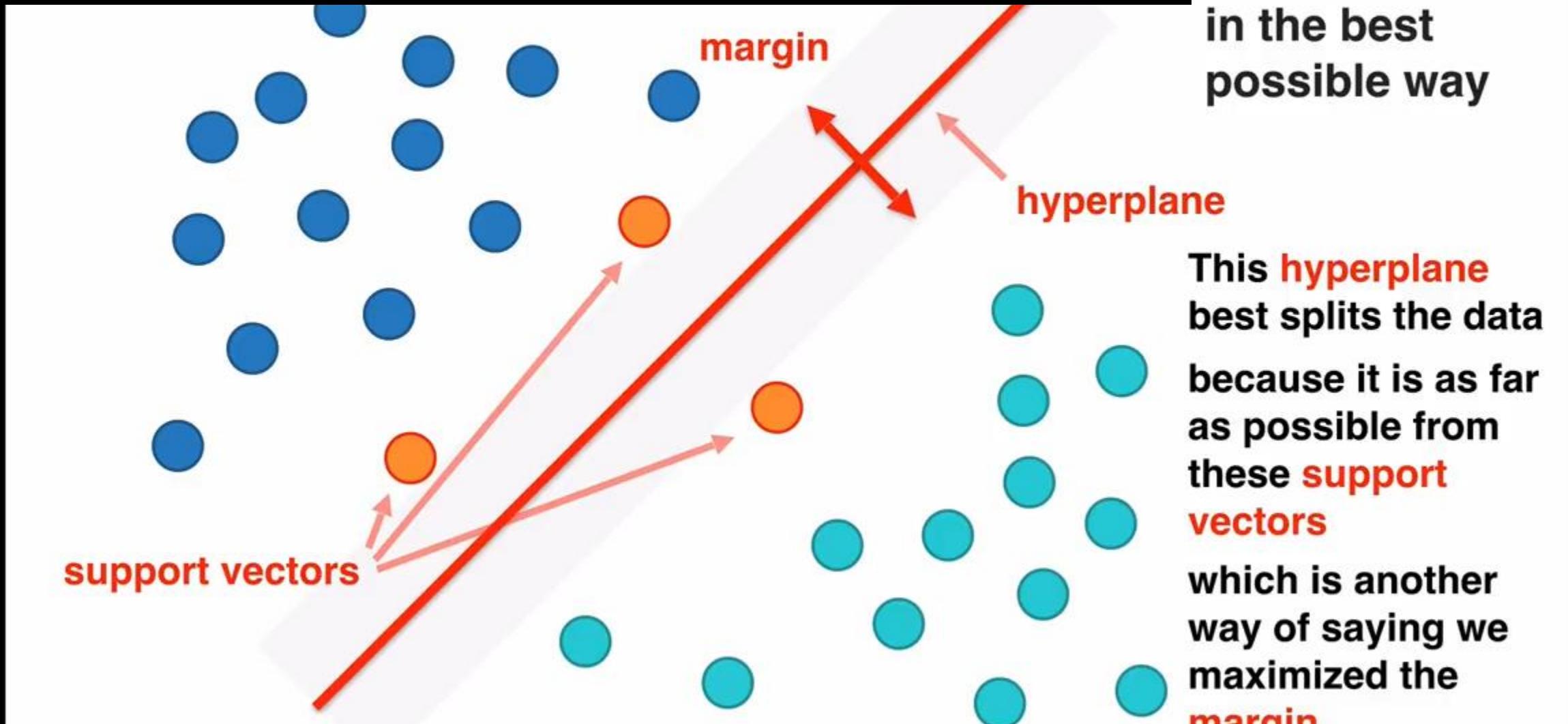
SUPPORT VECTOR MACHINES

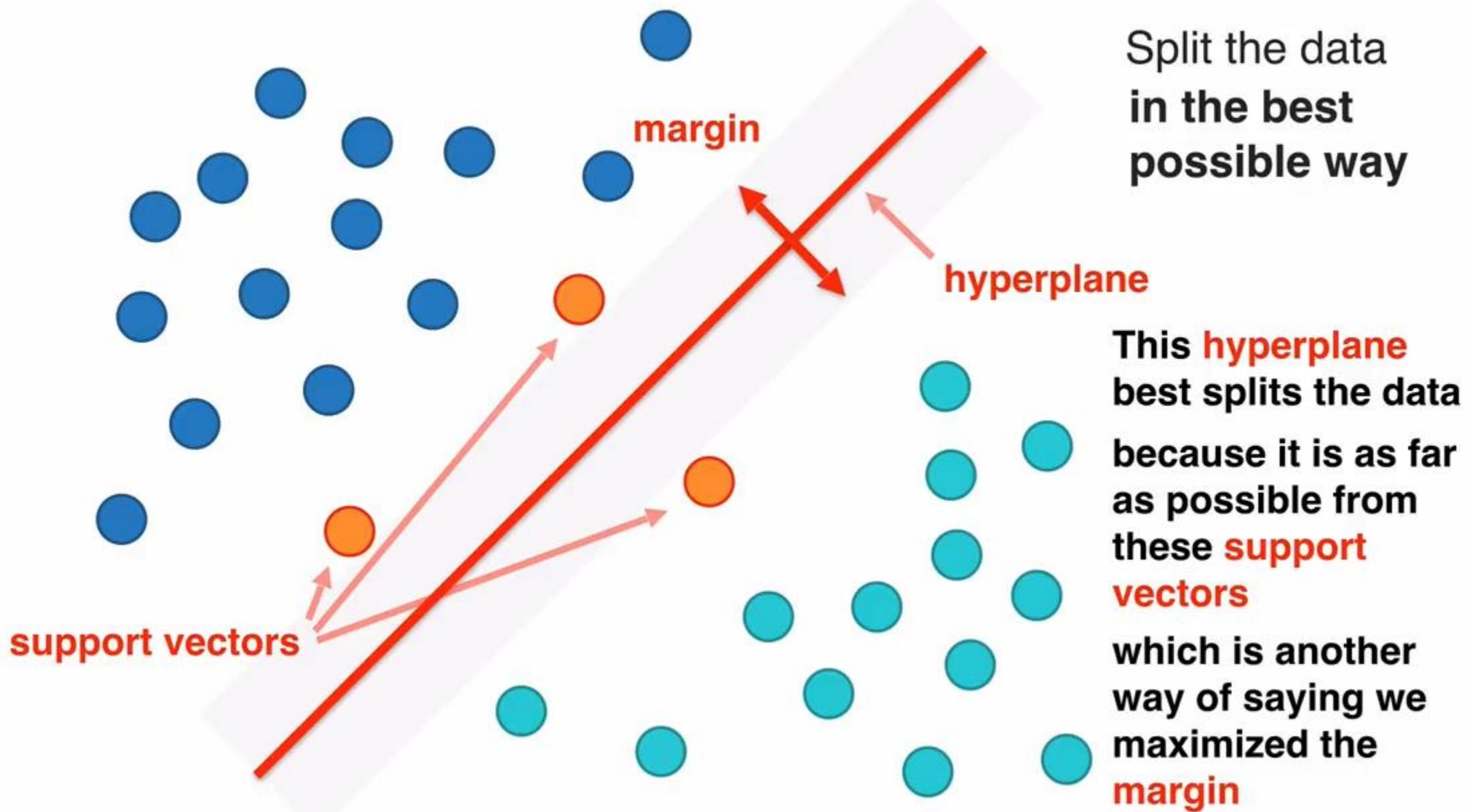
SVMs are a type of linear classifier that separates two classes using a line (in 2D) or a hyperplane (in higher dimensions)

This is called a decision boundary



INTRODUCTION TO SUPPORT VECTOR MACHINES





SVM

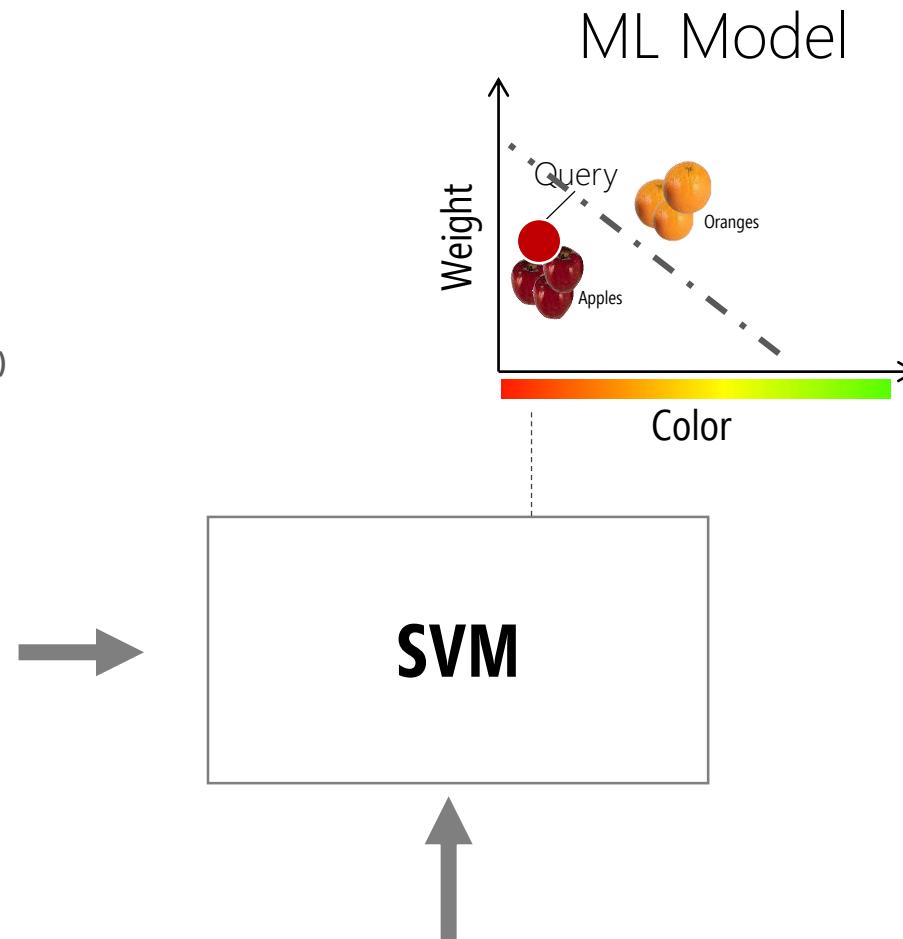
SVM EXAMPLE

Input Features		Class Label (aka Ground Truth Label)
Color	Weight	Label
Red	200g	Apple
Orange	300g	Orange
Red	150g	Apple
...		

Training Data

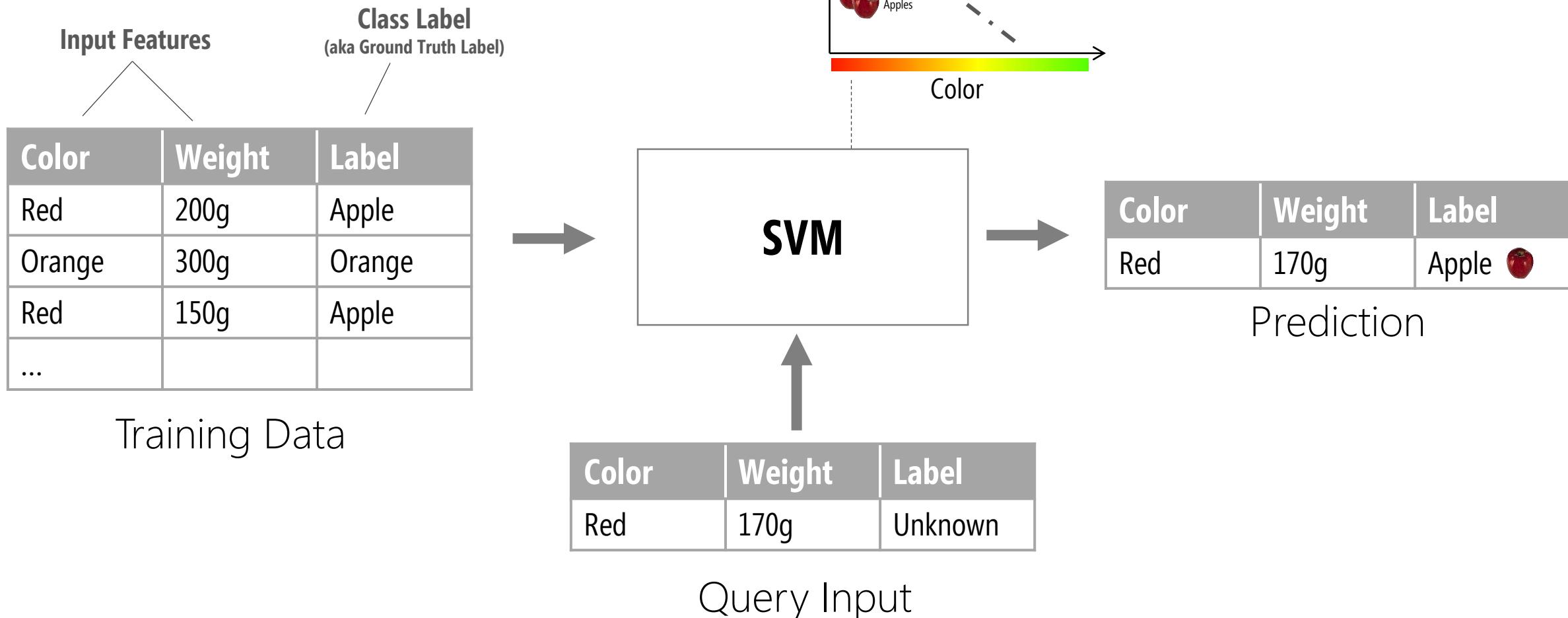
Color	Weight	Label
Red	170g	Unknown

Query Input



K-NEAREST NEIGHBORS

SVM EXAMPLE



K-NEAREST NEIGHBORS

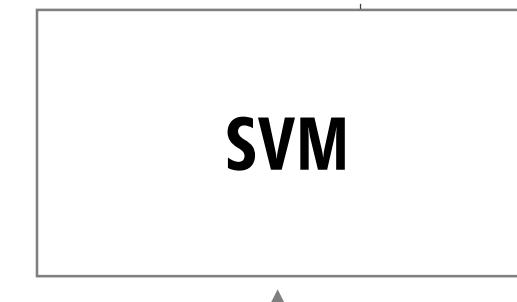
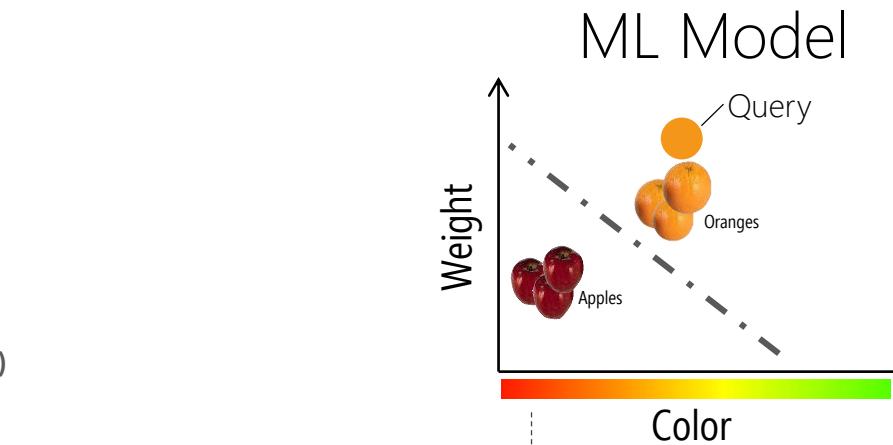
SVM EXAMPLE

Input Features		Class Label (aka Ground Truth Label)
Color	Weight	Label
Red	200g	Apple
Orange	300g	Orange
Red	150g	Apple
...		

Training Data

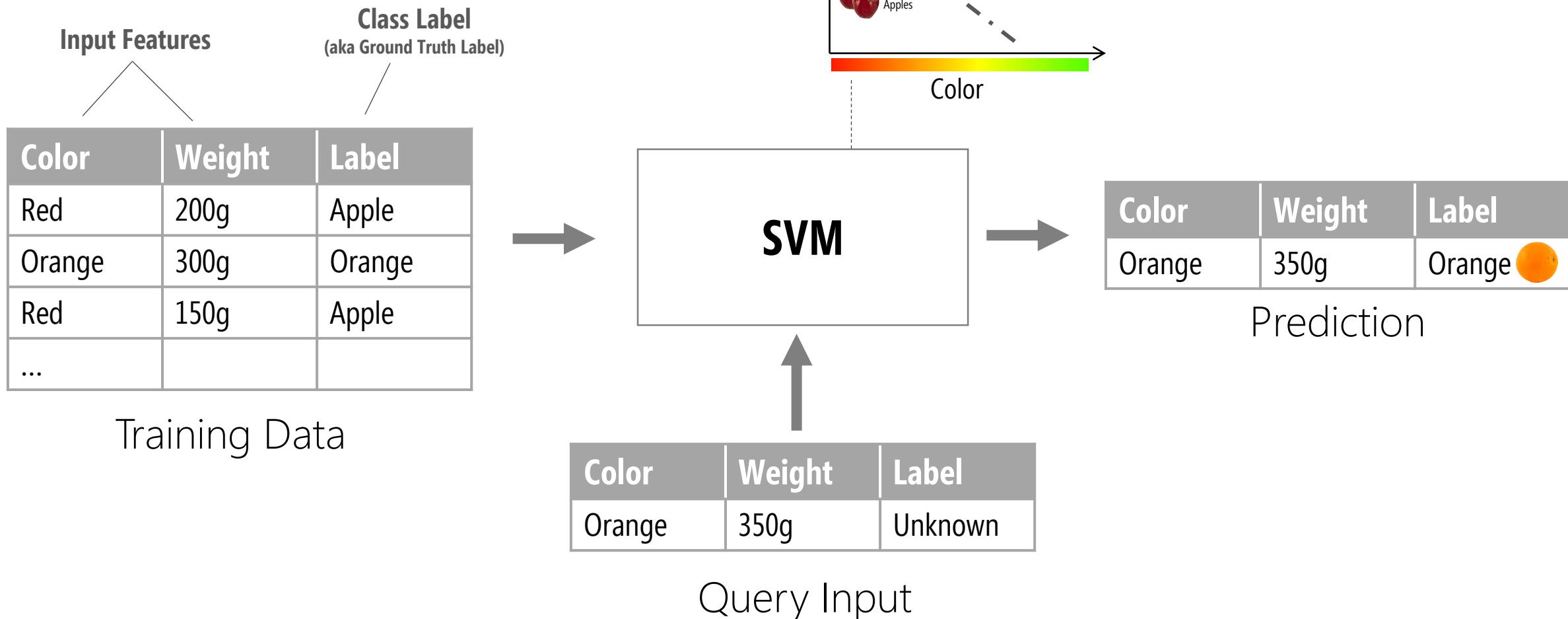
Color	Weight	Label
Orange	350g	Unknown

Query Input

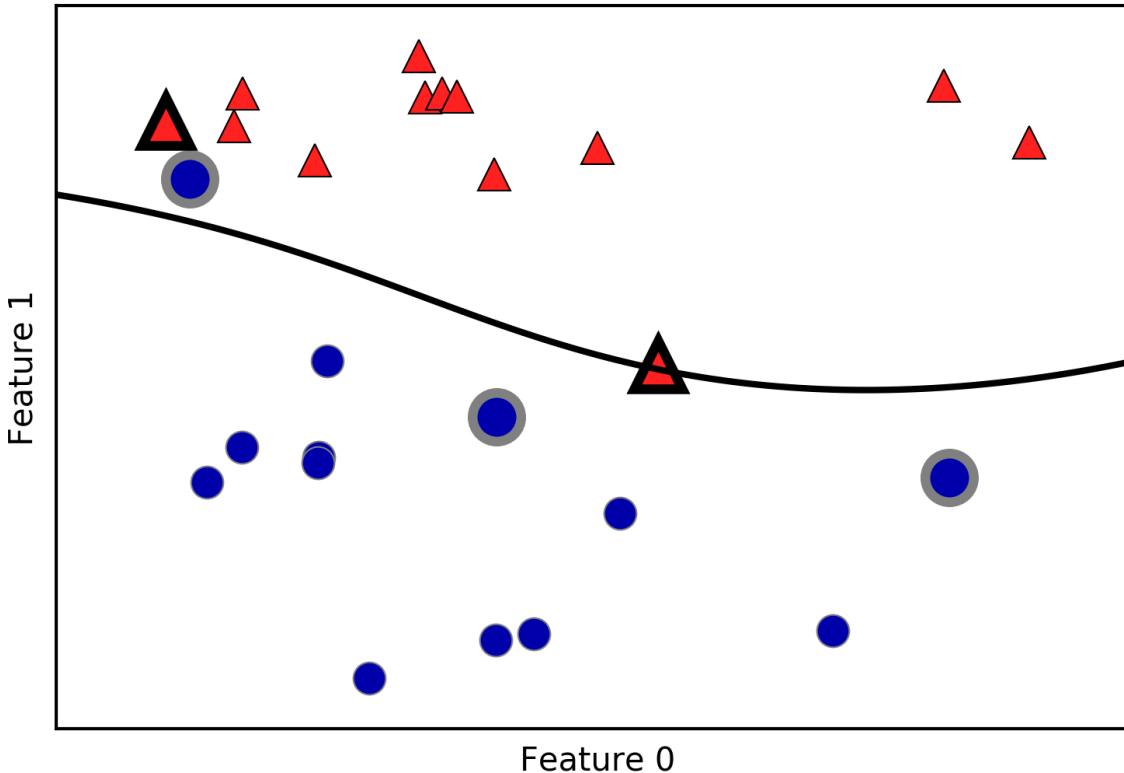


K-NEAREST NEIGHBORS

SVM EXAMPLE

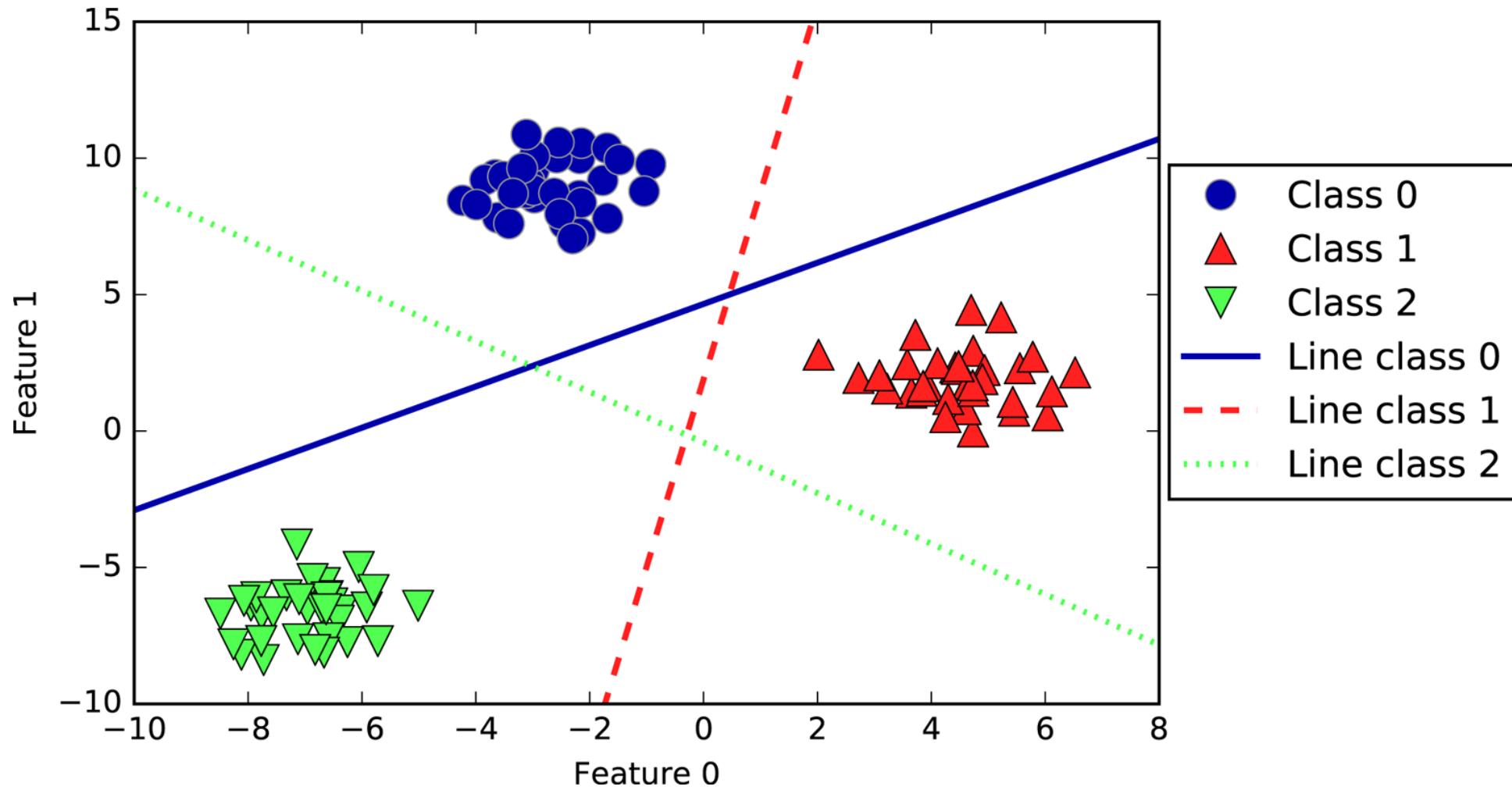


NON-LINEAR KERNEL

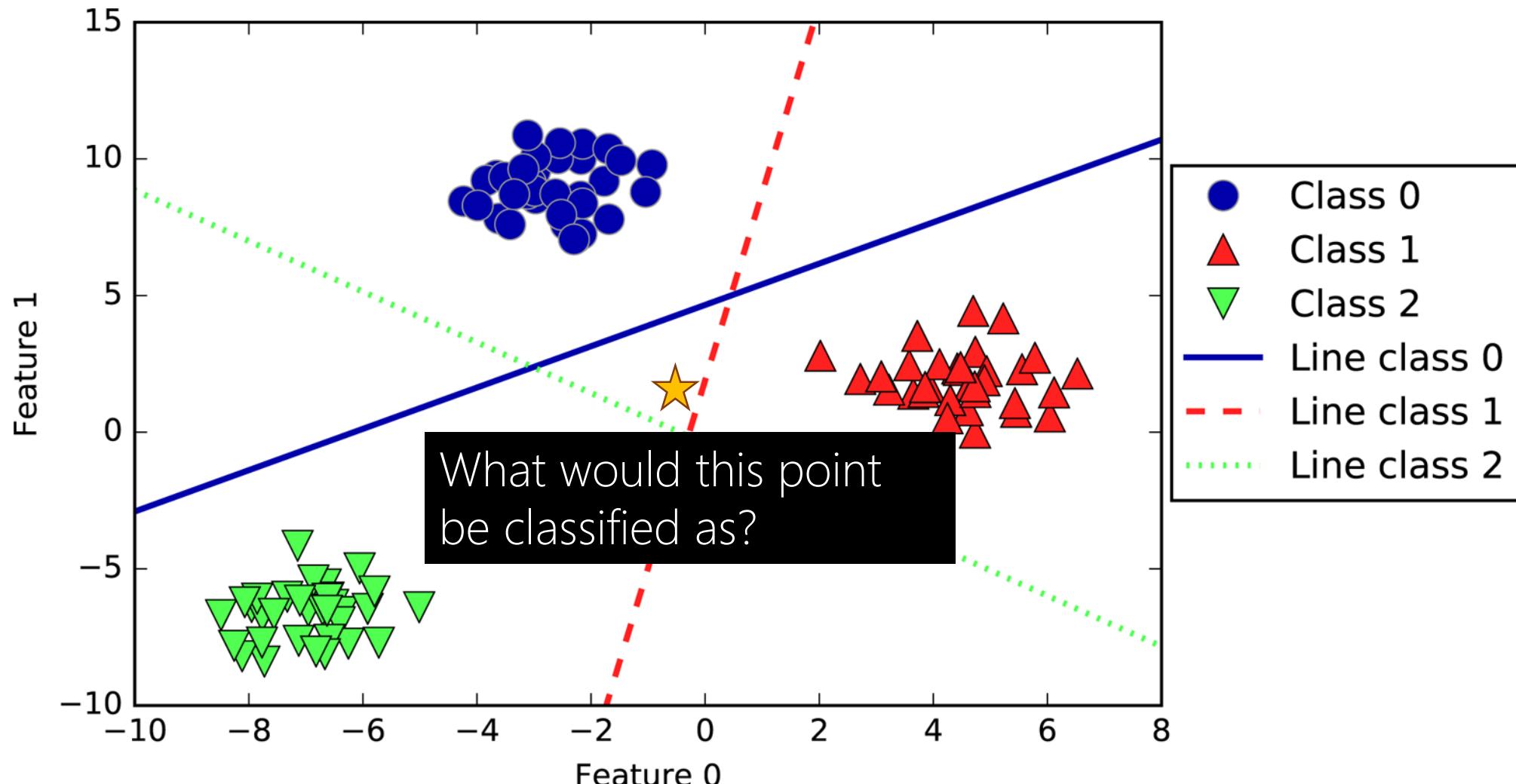


Decision boundary and support vectors found by an SVM with RBF kernel

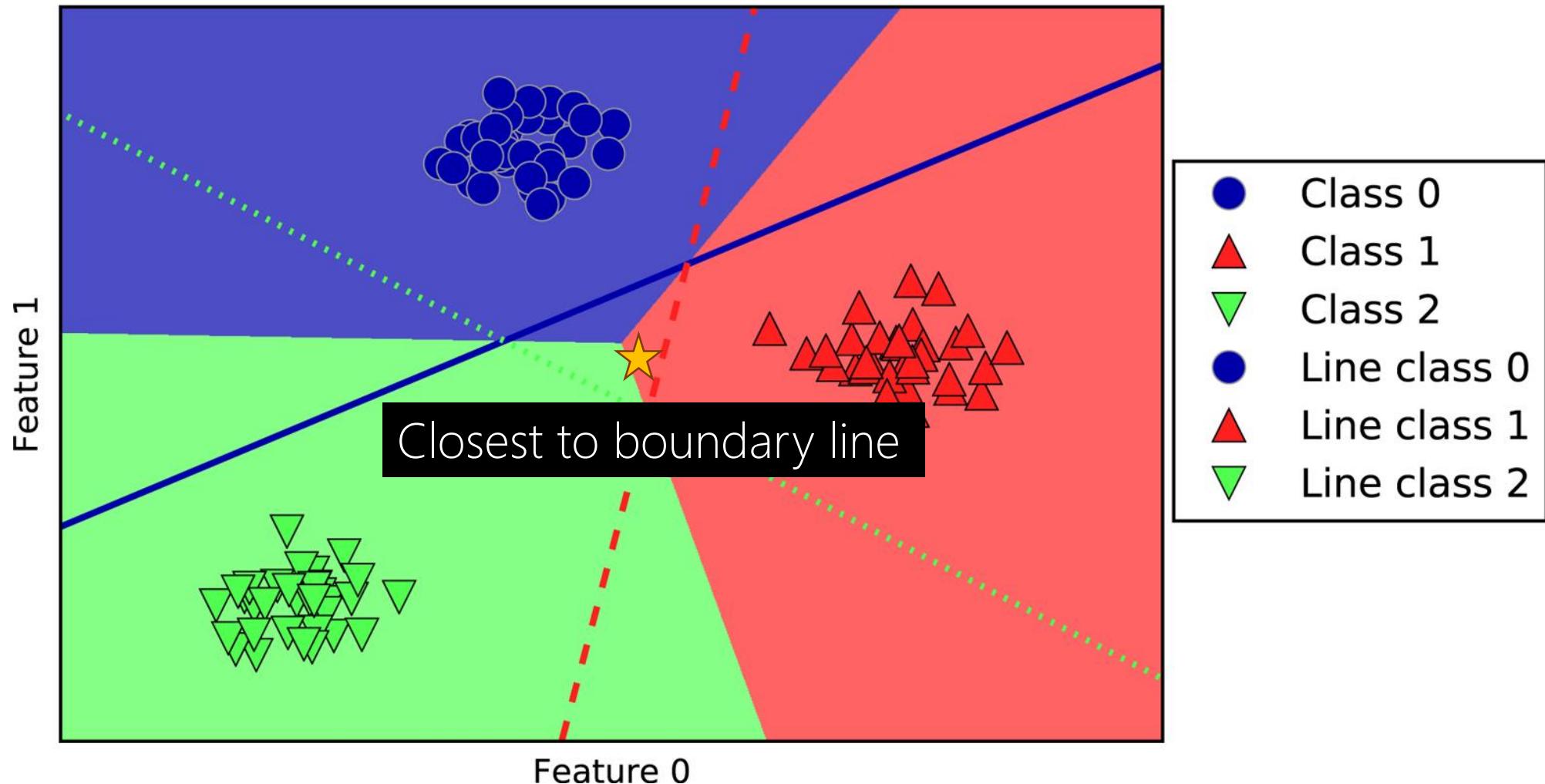
MULTICLASS: ONE VS. REST APPROACH



MULTICLASS: ONE VS. REST APPROACH



MULTICLASS: ONE VS. REST APPROACH



STRENGTHS AND WEAKNESSES

Strengths

Powerful models that perform well on a variety of datasets

Allow for complex decision boundaries, even if the data has only a few features

Works well for both low- and high-dimensional data (*i.e.*, few and many features)

Weaknesses

Require careful preprocessing of data and tuning of parameters (*e.g.*, gamma and C)

Hard to inspect and interpret the model and why a particular prediction was made

ITERATING ON OUR MODEL

Feature selection

Feature processing

Parameter tuning

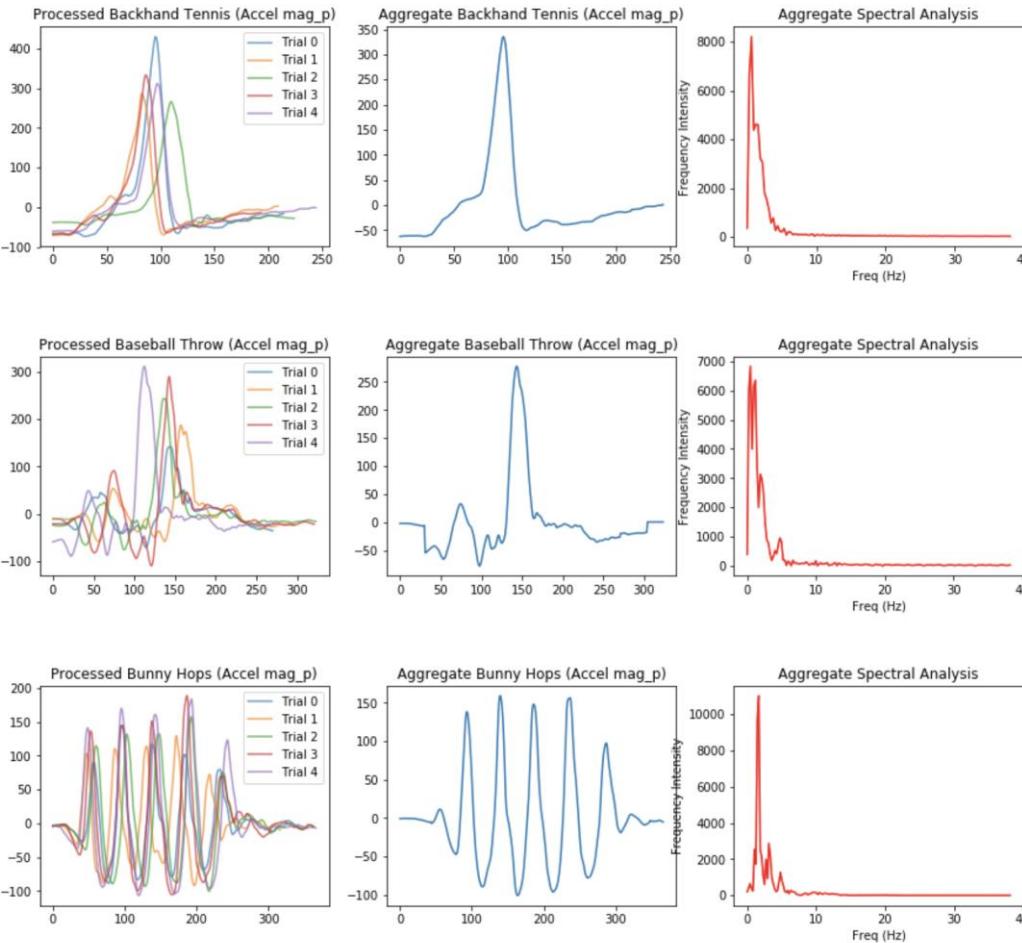
Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

Andrew Ng

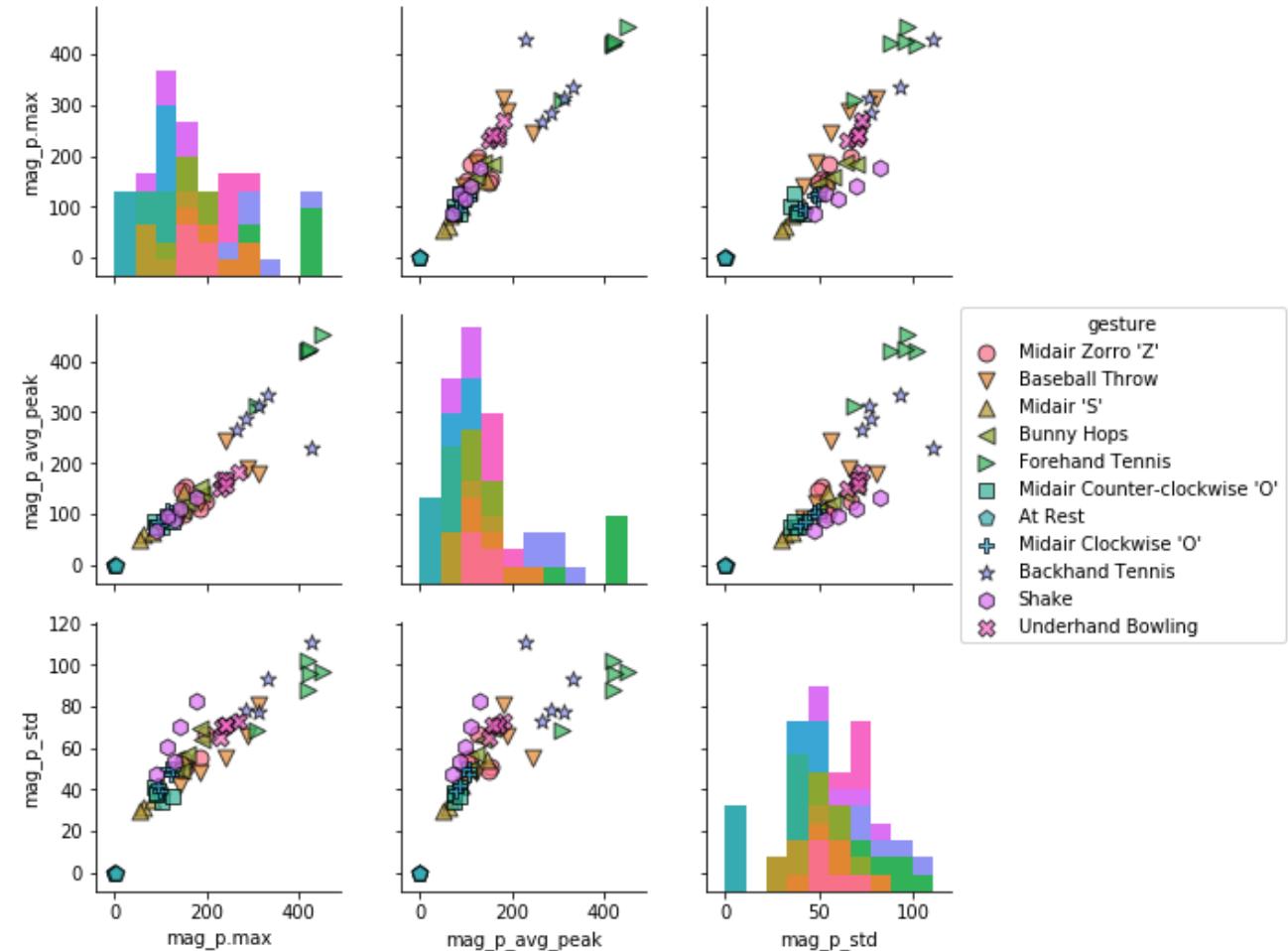
FEATURE SELECTION

SIGNAL EXPLORATIONS

Time-series Explorations



Feature Explorations



HOW DO WE SELECT FEATURES?

Based on **domain expertise** (e.g., a medical doctor helping select features for automatic melanoma recognition in images)

Based on **intuitions, experimentations, and plotting the data**

Feature selection algorithms!

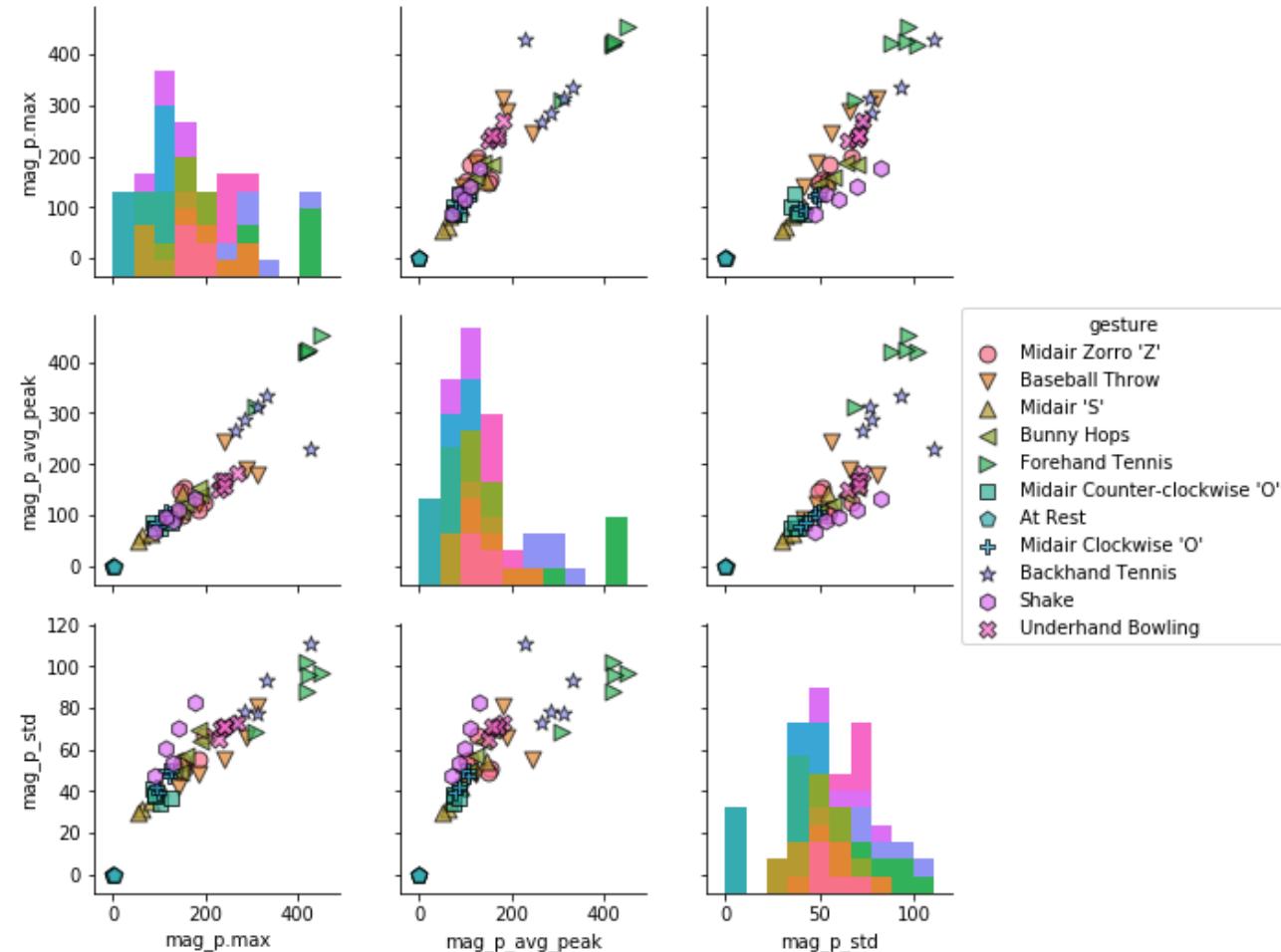
FEATURE SELECTION

PAIR PLOTS

Pair plots formalize some of our 2D explorations of features from last time

The diagonal is filled with histograms of each features (or density plots)

Other cells are 2D combinations

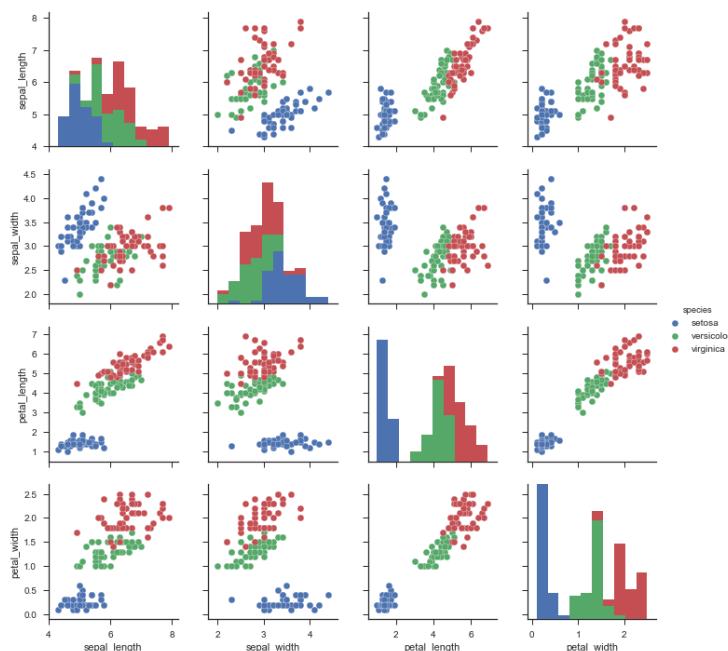


FEATURE SELECTION

PAIR PLOTS: PLAYING WITH ML NOTEBOOK

```
# Draw scatterplots for joint relationships and histograms for univariate distributions
# See: https://seaborn.pydata.org/generated/seaborn.pairplot.html
import seaborn as sns
sns.set(style="ticks", color_codes=True)
iris = sns.load_dataset("iris")
g = sns.pairplot(iris)

# Show different levels of a categorical variable by the color of plot elements:
g = sns.pairplot(iris, hue="species")
```



FEATURE SELECTION

PAIR PLOTS: PLAYING WITH ML NOTEBOOK

```

import pandas as pd
import seaborn as sns

features = dict()
for gesture_name, gesture_trials in selected_gesture_set.map_gestures_to_trials.items():
    for trial in gesture_trials:
        if 'mag_p.max' not in features:
            features['mag_p.max'] = list()
        features['mag_p.max'].append(trial.accel.mag_p.max())

        if 'mag_p_std' not in features:
            features['mag_p_std'] = list()
        features['mag_p_std'].append(np.std(trial.accel.mag_p))

    # peak count
    min_distance_between_peaks = 77 / 3.0 # 77 is the sampling frequency
    peak_indices, peak_properties = sp.signal.find_peaks(trial.accel.mag_p, height=30, distance=min_distance_between_peaks)

    # average of peak values
    if 'mag_p_avg_peak' not in features:
        features['mag_p_avg_peak'] = list()

    mag_p_avg_peak_cnt = 0
    if len(peak_indices) > 0:
        mag_p_avg_peak_cnt = np.average(trial.accel.mag_p[peak_indices])

    features['mag_p_avg_peak'].append(mag_p_avg_peak_cnt)

    if 'gesture' not in features:
        features['gesture'] = list()

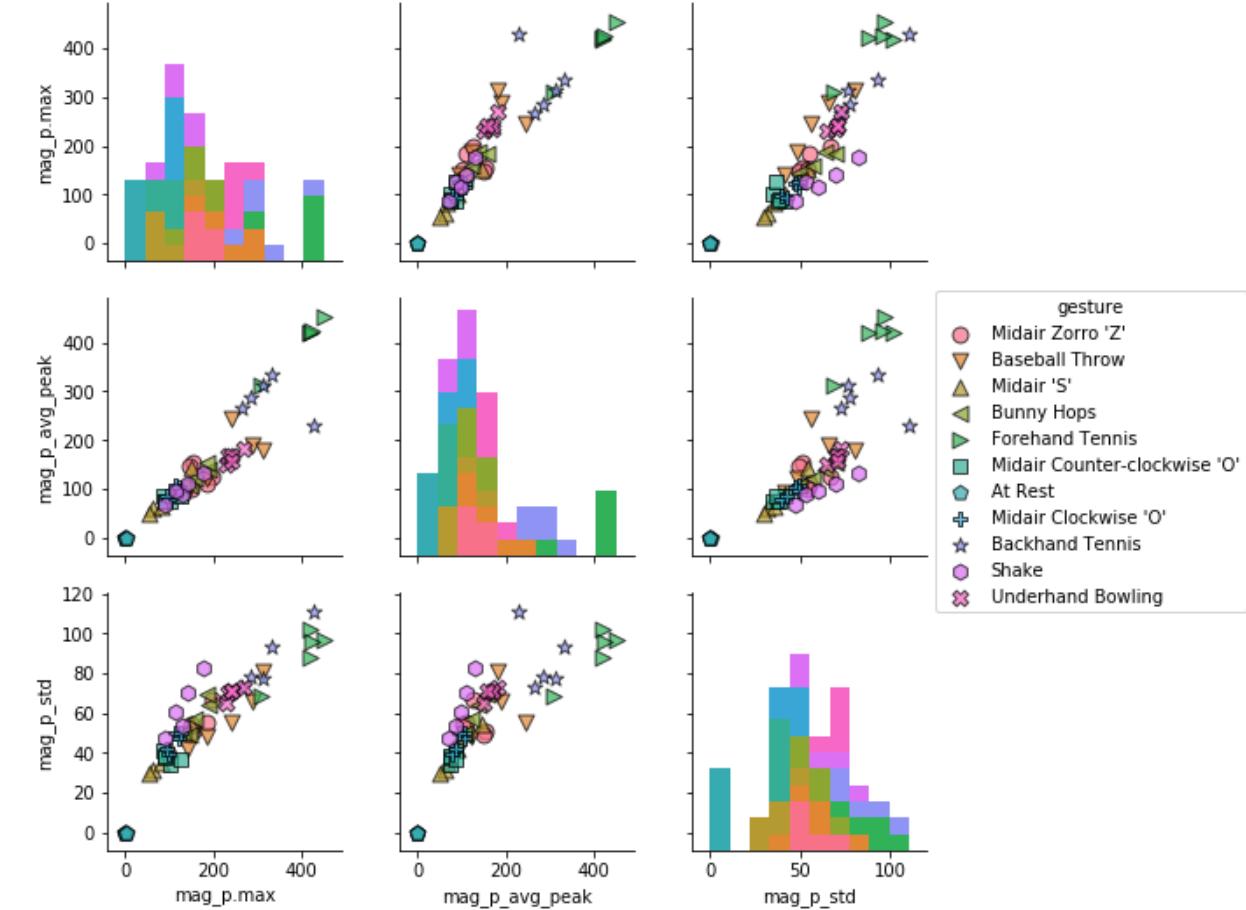
    features['gesture'].append(gesture_name)

df = pd.DataFrame(data=features)

markers = ['o','v','^','<','>','s','p','P','*','h','X','D','d','|','_','0,1,2,3,4,5,6,7,8,9,10','1','2','3','4',' ',' ']
markers = markers[:len(selected_gesture_set.map_gestures_to_trials)]

g = sns.pairplot(df, hue="gesture",
                  plot_kws = {'alpha': 0.7, 's': 80, 'edgecolor': 'k'}, markers = markers)

```



FEATURE SELECTION ALGORITHMS

Given a giant list of input features, a feature selection algorithm can be seen as a search technique for finding the best combination of feature subsets

Brute force: test each possible subset of features finding the one that minimizes the error rate. This is computationally intractable.

Scikit learn and other toolkits have support for this.

https://scikit-learn.org/stable/modules/feature_selection.html

[Previous](#)
1.12.
Multi...
[Next](#)
1.14. Semi-
Supervised
[Up](#)
1.
Supervised...

scikit-learn v0.21.1
[Other versions](#)

Please [cite us](#) if you use
the software.

1.13. Feature selection

- 1.13.1. Removing features with low variance
- 1.13.2. Univariate feature selection
- 1.13.3. Recursive feature elimination
- 1.13.4. Feature selection using SelectFromModel
 - 1.13.4.1. L1-based feature selection
 - 1.13.4.2. Tree-based feature selection
- 1.13.5. Feature selection as part of a pipeline

1.13. Feature selection

The classes in the `sklearn.feature_selection` module can be used for feature selection/dimensionality reduction on sample sets, either to improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets.

1.13.1. Removing features with low variance

`VarianceThreshold` is a simple baseline approach to feature selection. It removes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e. features that have the same value in all samples.

As an example, suppose that we have a dataset with boolean features, and we want to remove all features that are either one or zero (on or off) in more than 80% of the samples. Boolean features are Bernoulli random variables, and the variance of such variables is given by

$$\text{Var}[X] = p(1 - p)$$

so we can select using the threshold `.8 * (1 - .8)`:

```
>>> from sklearn.feature_selection import VarianceThreshold
>>> X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
>>> sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
>>> sel.fit_transform(X)
array([[0, 1],
       [1, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [1, 1]])
```

As expected, `VarianceThreshold` has removed the first column, which has a probability $p = 5/6 > .8$ of containing a zero.

1.13.2. Univariate feature selection

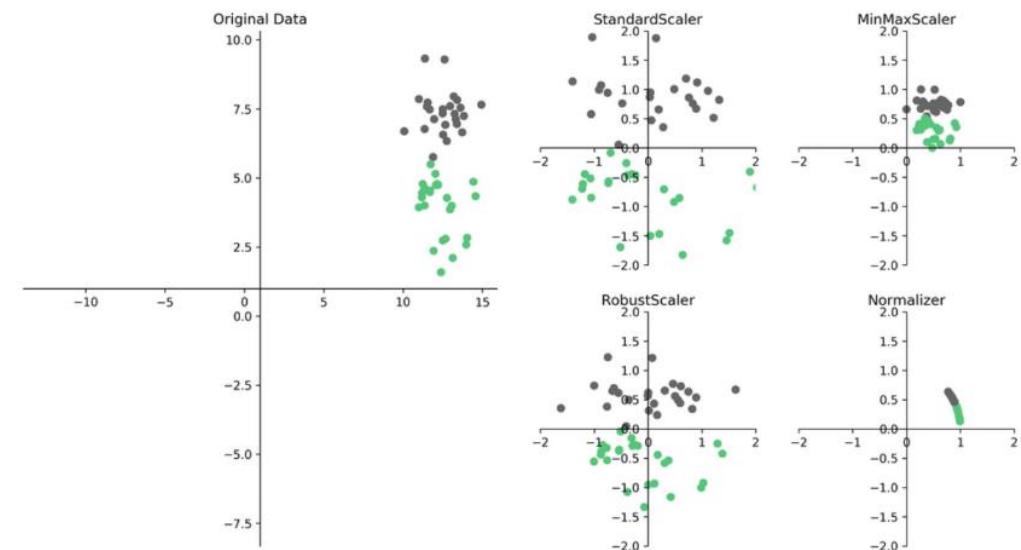
Univariate feature selection works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator. Scikit-learn exposes feature selection routines as objects that implement the `transform` method:

- `SelectKBest` removes all but the k highest scoring features
- `SelectPercentile` removes all but a user-specified highest scoring percentage of features
- using common univariate statistical tests for each feature: false positive rate `SelectFpr`, false discovery rate `SelectFdr`, or family wise error `SelectFwe`.

FEATURE PROCESSING

Need to ensure that the feature vectors are scaled appropriately.

For example, if you have a two-dimensional input vector $[x,y]$ and the x values are orders of magnitude larger than the y -values, the y -values are inconsequential



SCALING DATA IN SCIKIT LEARN

Unscaled

```
trainingData = np.array(trainingData)
clf = svm.SVC(kernel='linear') # kernel='rbf'
clf.fit(trainingData, classLabels)

# make predictions for this test set
for testGestureName, testTrial in testFold.items():
    features = extract_features_example(testTrial)

    svmPrediction = clf.predict([features])
    y_true.append(testGestureName)
    y_pred.append(svmPrediction)

    if testGestureName == svmPrediction[0]:
        mapGestureToCorrectMatches[testGestureName] += 1
```

Scaled

```
trainingData = np.array(trainingData)
scaler = StandardScaler()
scaler.fit(trainingData)
training_data_scaled = scaler.transform(trainingData)

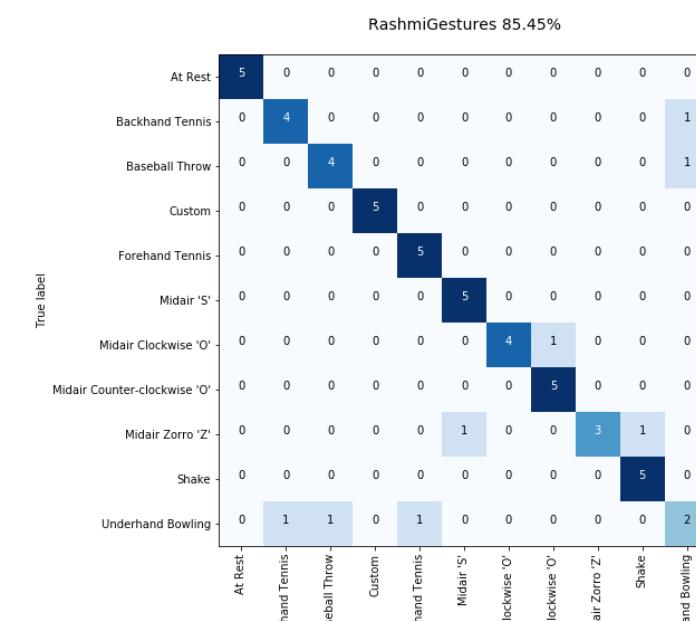
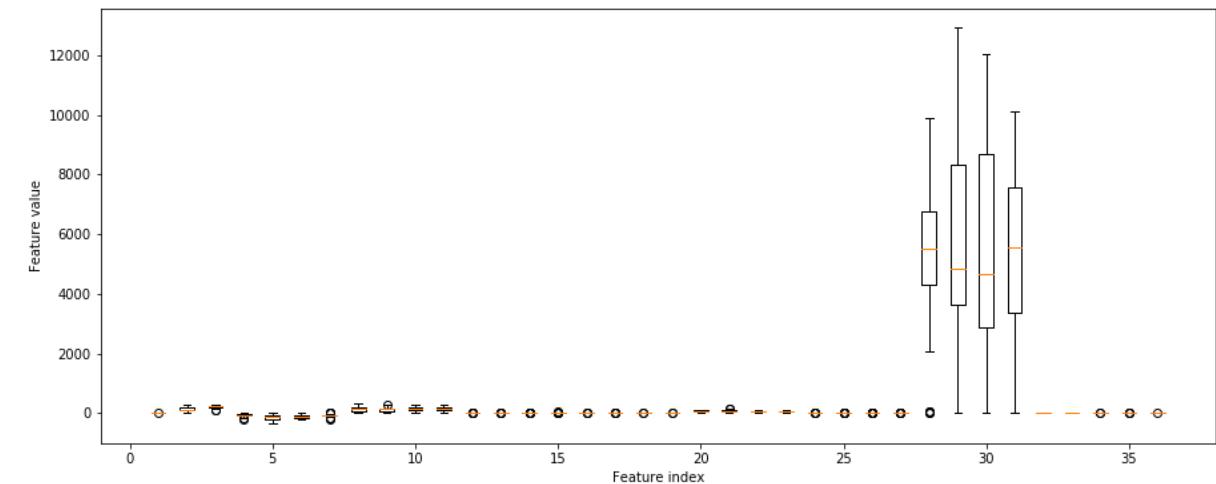
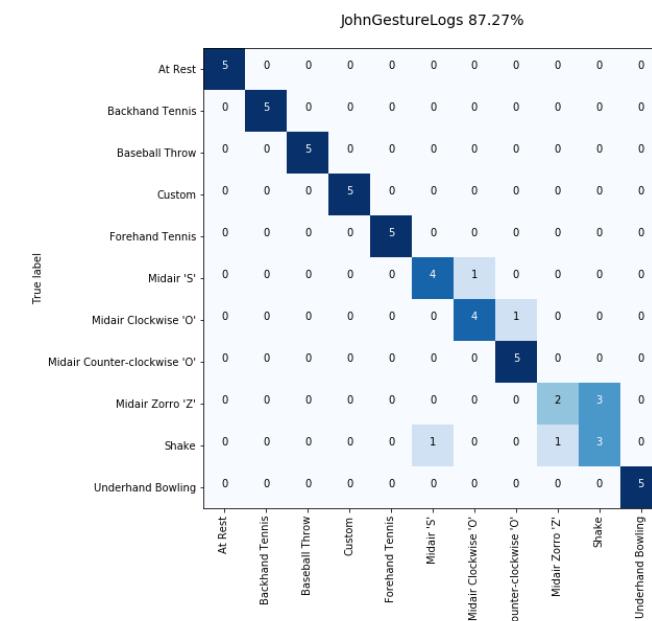
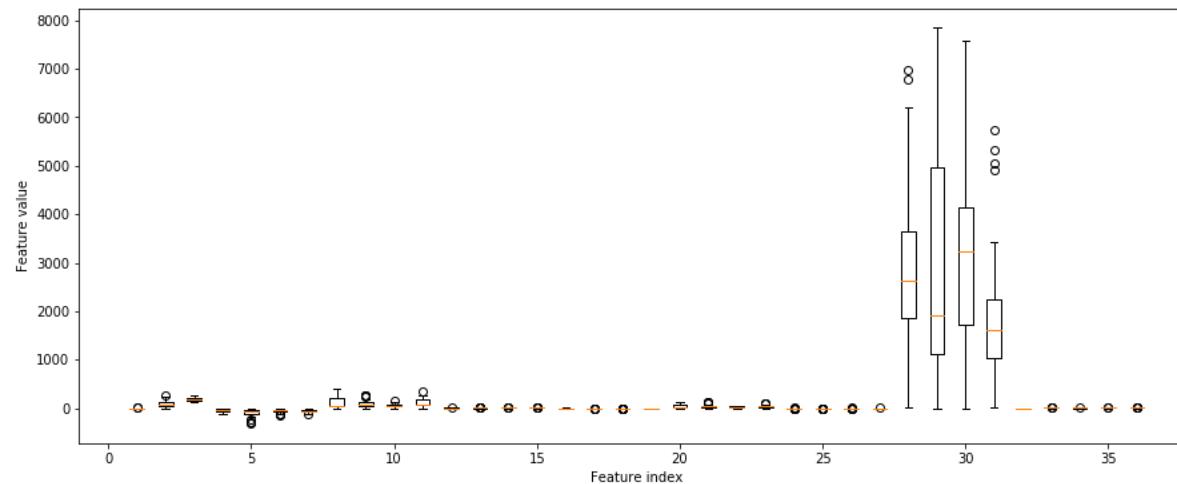
clf = svm.SVC(kernel='linear') # kernel='rbf'
clf.fit(training_data_scaled, classLabels)

# make predictions for this test set
for testGestureName, testTrial in testFold.items():
    features = np.array(extract_features(testTrial))
    features_scaled = scaler.transform([features])
    svmPrediction = clf.predict(features_scaled)
    y_true.append(testGestureName)
    y_pred.append(svmPrediction)

    if testGestureName == svmPrediction[0]:
        mapGestureToCorrectMatches[testGestureName] += 1
```

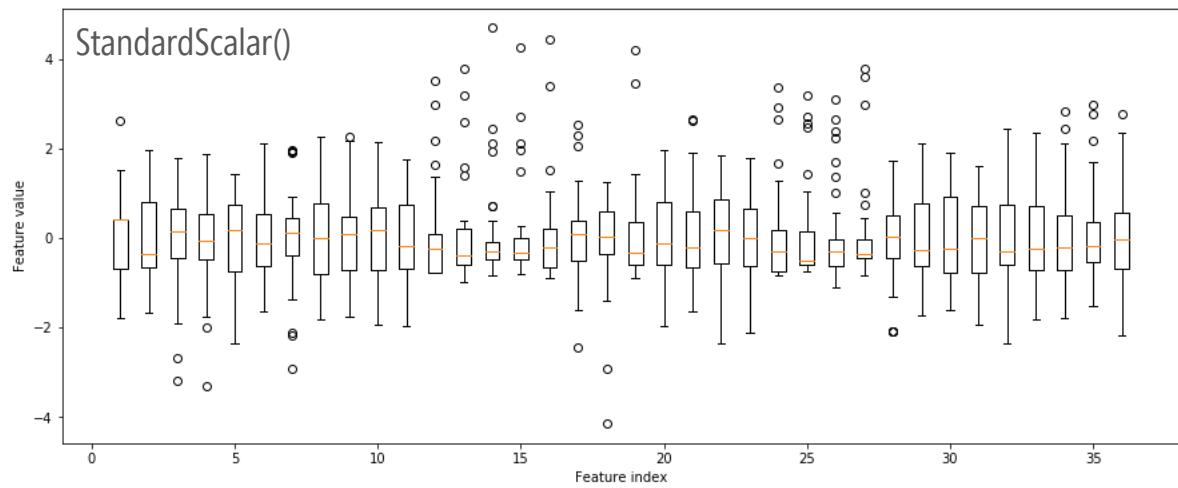
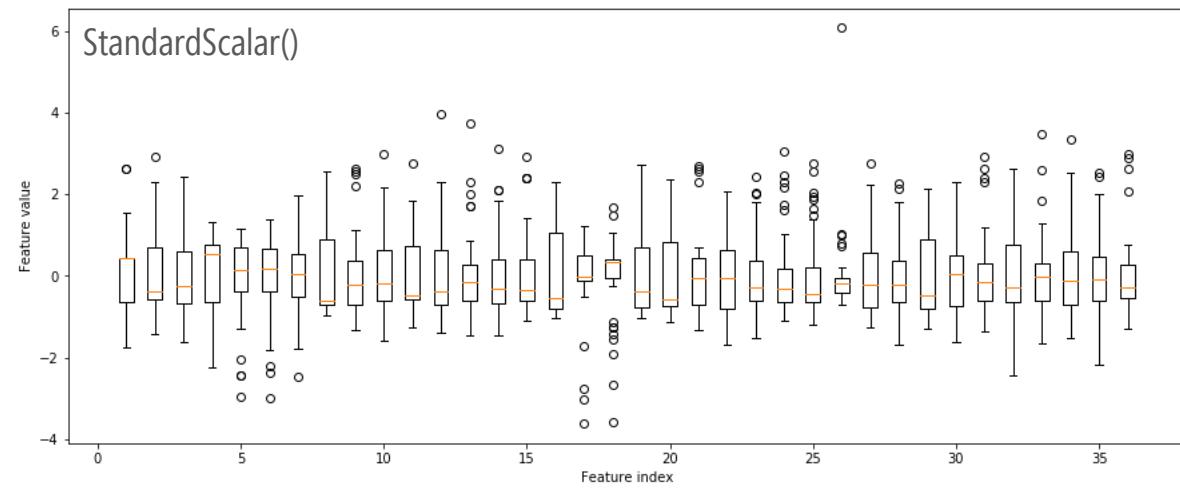
FEATURE SCALING

UNSCALED SVM CLASSIFICATION RESULTS



FEATURE SCALING

SCALED SVM CLASSIFICATION RESULTS

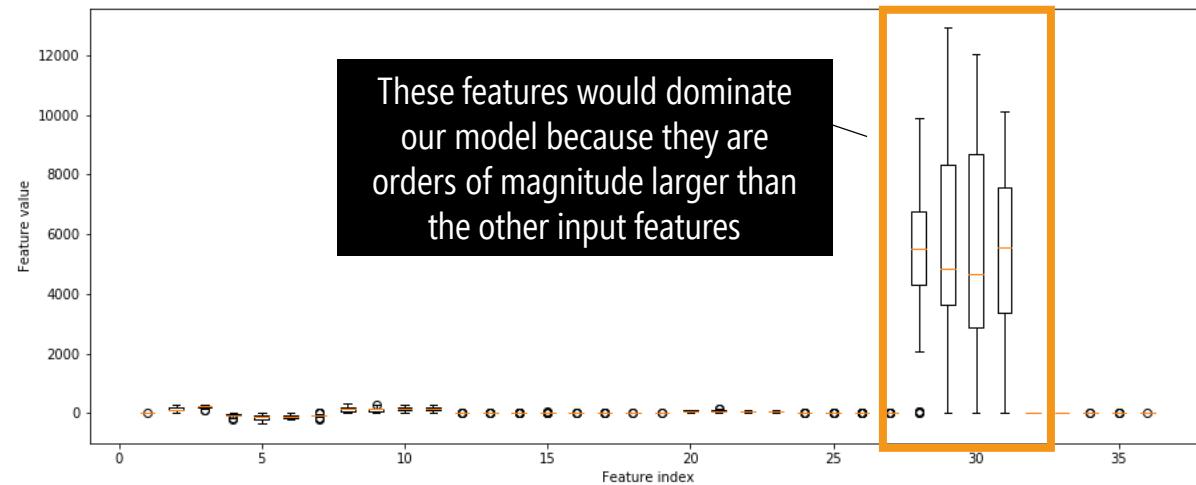


JohnGestureLogs 98.18%

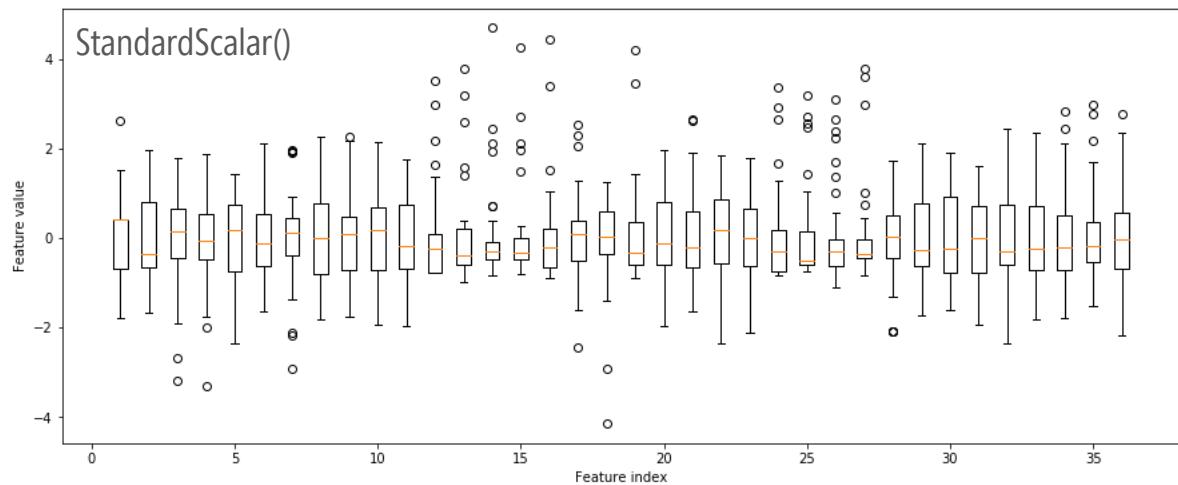
RashmiGestures 100.00%

FEATURE SCALING

COMPARING UNSCALED VS. SCALED



These features would dominate our model because they are orders of magnitude larger than the other input features



RashmiGestures 85.45%

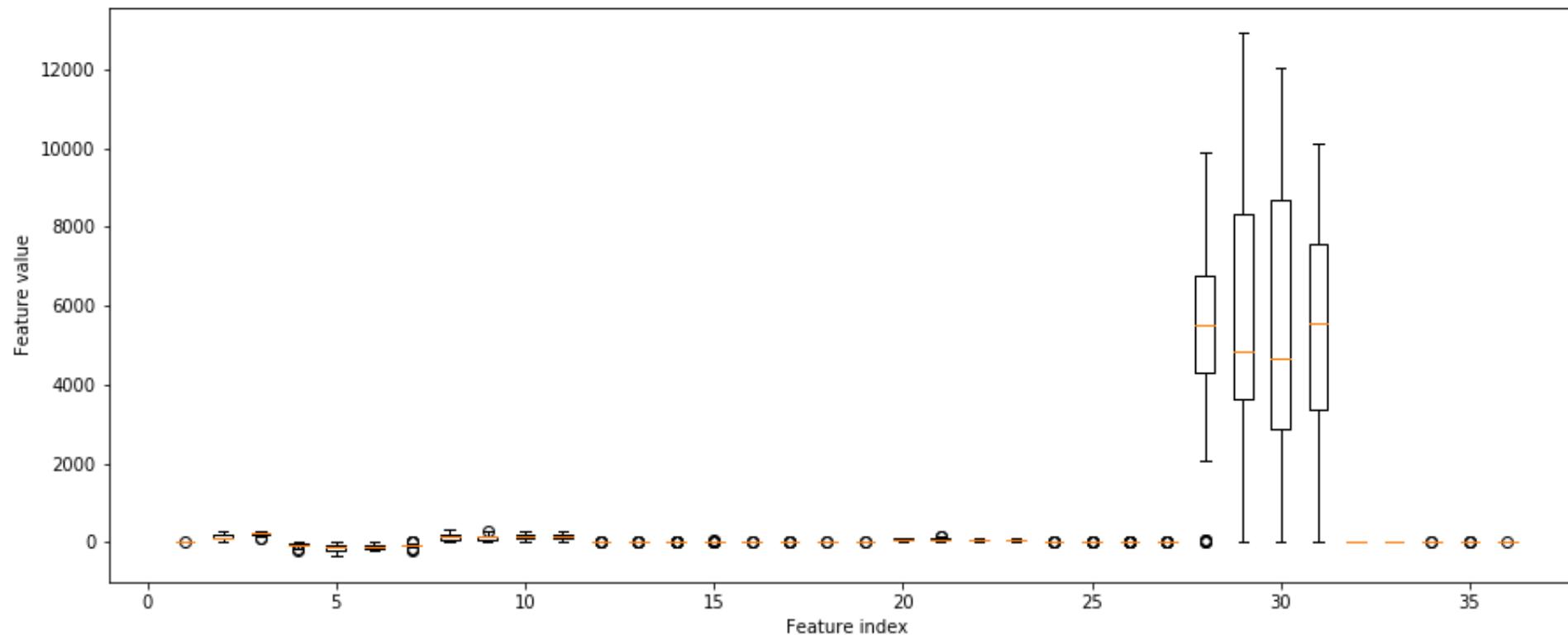
	1	2	3	4	5	6	7	8	9	10	11
At Rest	5	0	0	0	0	0	0	0	0	0	0
Backhand Tennis	0	4	0	0	0	0	0	0	0	0	1
Baseball Throw	0	0	4	0	0	0	0	0	0	0	1
Custom	0	0	0	5	0	0	0	0	0	0	0
Forehand Tennis	0	0	0	0	5	0	0	0	0	0	0
Midair 'S'	0	0	0	0	0	5	0	0	0	0	0
Midair Clockwise 'O'	0	0	0	0	0	0	4	1	0	0	0
Midair Counter-clockwise 'O'	0	0	0	0	0	0	0	5	0	0	0
Midair Zorro 'Z'	0	0	0	0	0	1	0	0	3	1	0
Shake	0	0	0	0	0	0	0	0	0	5	0
Underhand Bowling	0	1	1	0	1	0	0	0	0	0	2

Scaling our features with StandardScalar() leads to a dramatic improvement in accuracy.

RashmiGestures 100.00%

AS AN ASIDE: GRAPHING FEATURE BOX PLOTS

```
plt.figure(figsize=(15, 6))
plt.boxplot(training_data, manage_xticks=False)
plt.xlabel("Feature index")
plt.ylabel("Feature value")
```



PARAMETER TUNING

Most ML algorithms have parameters to set

When you train a model, you must provide values for those parameters and the efficacy of the model depends on those parameters

The optimal set of parameters is known as *model selection*

Scikit learn and other frameworks also support this...

[Previous
3.1. Cross-
va...](#)[Next
3.2.4.1.1.
sk...](#)[Up
3. Model
sele...](#)

scikit-learn v0.21.1
[Other versions](#)

Please [cite us](#) if you use
the software.

3.2. Tuning the hyper-parameters of an estimator

[3.2.1. Exhaustive Grid Search](#)

[3.2.2. Randomized Parameter Optimization](#)

[3.2.3. Tips for parameter search](#)

- [3.2.3.1. Specifying an objective metric](#)

- [3.2.3.2. Specifying multiple metrics for evaluation](#)

- [3.2.3.3. Composite estimators and parameter spaces](#)

- [3.2.3.4. Model selection: development and evaluation](#)

- [3.2.3.5. Parallelism](#)

- [3.2.3.6. Robustness to failure](#)

[3.2.4. Alternatives to brute force parameter search](#)

- [3.2.4.1. Model specific cross-validation](#)

- [3.2.4.2. Information Criterion](#)

- [3.2.4.3. Out of Bag Estimates](#)

3.2. Tuning the hyper-parameters of an estimator

Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes. Typical examples include `c`, `kernel` and `gamma` for Support Vector Classifier, `alpha` for Lasso, etc.

It is possible and recommended to search the hyper-parameter space for the best [cross validation](#) score.

Any parameter provided when constructing an estimator may be optimized in this manner. Specifically, to find the names and current values for all parameters for a given estimator, use:

```
estimator.get_params()
```

A search consists of:

- an estimator (regressor or classifier such as `sklearn.svm.SVC()`);
- a parameter space;
- a method for searching or sampling candidates;
- a cross-validation scheme; and
- a [score function](#).

Some models allow for specialized, efficient parameter search strategies, outlined below. Two generic approaches to sampling search candidates are provided in scikit-learn: for given values, `GridSearchCV` exhaustively considers all parameter combinations, while `RandomizedSearchCV` can sample a given number of candidates from a parameter space with a specified distribution. After describing these tools we detail [best practice](#) applicable to both approaches.

Note that it is common that a small subset of those parameters can have a large impact on the predictive or computation performance of the model while others can be left to their default values. It is recommended to read the docstring of the estimator class to get a finer understanding of their expected behavior, possibly by reading the enclosed reference to the literature.

3.2.1. Exhaustive Grid Search

The grid search provided by `GridSearchCV` exhaustively generates candidates from a grid of parameter values specified with the `param_grid` parameter. For instance, the following `param_grid`:

```
param_grid = [  
    {'C': [1, 10, 100, 1000], 'kernel': ['linear']},  
    {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']}  
]
```

specifies that two grids should be explored: one with a linear kernel and C values in [1, 10, 100, 1000], and the second one with an RBF kernel, and the cross-product of C values ranging in [1, 10, 100, 1000] and gamma values in [0.001, 0.0001].

PARAMETER TUNING PROCESS

1. Define the parameter space. For a given ML model, decide what range of parameter values you want to consider.
2. Define the evaluation method (*e.g.*, cross-validation)
3. Define your key metrics (*e.g.*, accuracy, precision, recall, root-mean squared error)
4. Train and evaluate. For each unique combination of the parameter values, cross-validation performed and your key metrics produced.
5. Compare output and choose the best-performing parameters for your model.

PARAMETER TUNING

PARAMETER TUNING: DECISION TREE EXAMPLE ON AZURE

Most ML toolkits offer parameter tuning functionality. This example is from Microsoft Azure's ML toolkit.

Two-Class Boosted Decision Tree

Create trainer mode
Single Parameter

Maximum number of leaves per tree
20

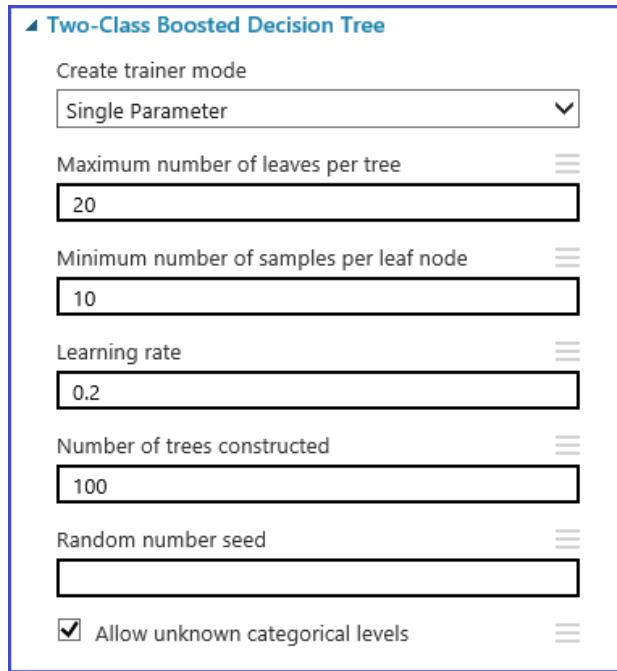
Minimum number of samples per leaf node
10

Learning rate
0.2

Number of trees constructed
100

Random number seed

Allow unknown categorical levels



Two-Class Boosted Decision Tree

Create trainer mode
Parameter Range

Maximum number of leaves per tree
Use Range Builder
Parameter Range : 101 - 900

Number of points : 3
Log Scale

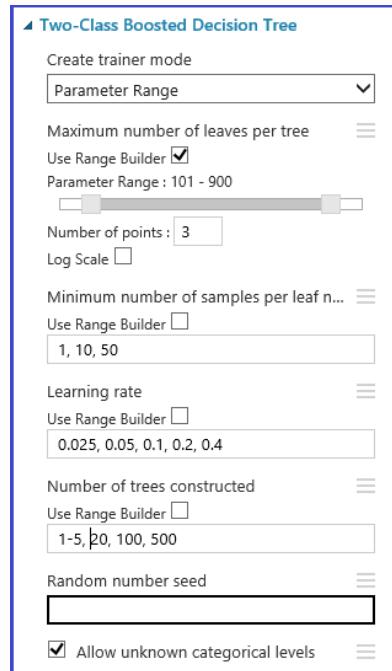
Minimum number of samples per leaf n...
Use Range Builder
1, 10, 50

Learning rate
Use Range Builder
0.025, 0.05, 0.1, 0.2, 0.4

Number of trees constructed
Use Range Builder
1-5, 20, 100, 500

Random number seed

Allow unknown categorical levels



Partition and Sample

Partition or sample mode
Assign to Folds

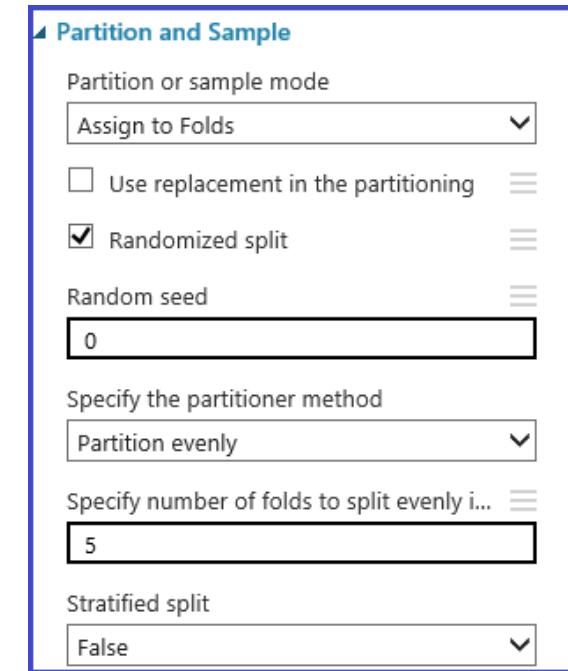
Use replacement in the partitioning
 Randomized split

Random seed
0

Specify the partitioner method
Partition evenly

Specify number of folds to split evenly i...
5

Stratified split
False



Sweep Parameters

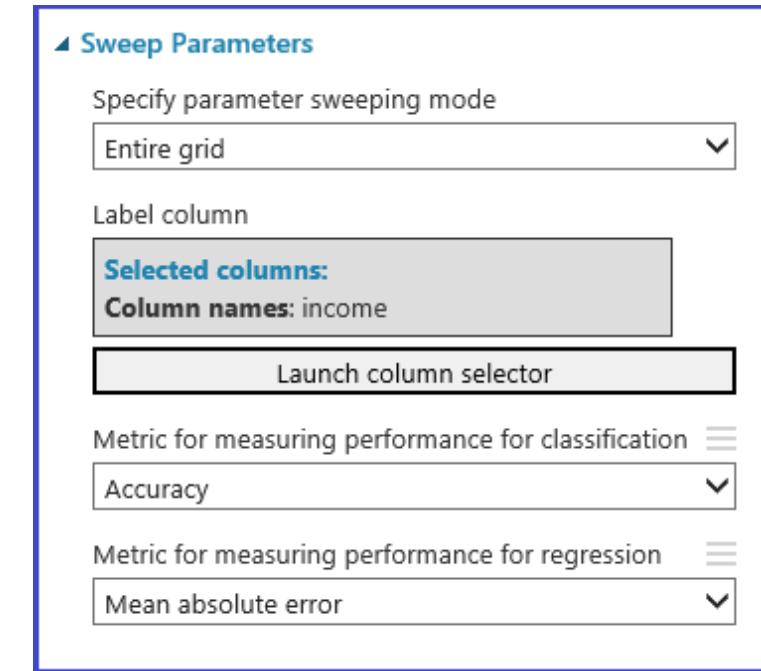
Specify parameter sweeping mode
Entire grid

Label column
Selected columns:
Column names: income

Launch column selector

Metric for measuring performance for classification
Accuracy

Metric for measuring performance for regression
Mean absolute error



1. Choose and Setup Model

Select an ML model to tune

2. Parameter Ranges

Decide what range of values to input and compare

3. Setup Evaluation

Decide how many folds in cross-validation and key error metrics.

PARAMETER TUNING

PARAMETER TUNING: DECISION TREE EXAMPLE ON AZURE

Most ML toolkits offer parameter tuning functionality. This example is from Microsoft Azure's ML toolkit.

The screenshot shows the 'Sweep Parameters' configuration panel. It includes fields for specifying the parameter sweeping mode ('Entire grid'), selecting the label column ('Selected columns: Column names: income'), and choosing metrics for classification ('Accuracy') and regression ('Mean absolute error'). The background shows other parts of the Azure ML studio interface.

ation
y folds in cross-validation and key error metrics.

rows	columns	Number of leaves	Minimum leaf instances	Learning rate	Number of trees	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
180	11	8	10	0.1	500	0.866128	0.75785	0.652595	0.701295	0.922837	0.290561	47.363286
		8	1	0.05	500	0.866097	0.762796	0.644306	0.698562	0.922672	0.289389	47.575474
		8	1	0.2	100	0.865944	0.763813	0.641755	0.697484	0.921469	0.291591	47.176649
		32	1	0.05	100	0.865913	0.759367	0.648769	0.699725	0.921503	0.29273	46.970281
		8	10	0.2	100	0.865882	0.763821	0.641372	0.697262	0.922052	0.290024	47.46048
		32	1	0.025	500	0.865668	0.749604	0.663946	0.70418	0.923187	0.292484	47.014813
		8	10	0.05	500	0.865514	0.761322	0.643158	0.697269	0.922803	0.288929	47.658898
		32	1	0.1	100	0.865453	0.751234	0.659737	0.702519	0.922703	0.292779	46.961499
		8	1	0.025	500	0.865391	0.764778	0.636909	0.695011	0.921386	0.29142	47.207519
		8	50	0.05	500	0.865115	0.760932	0.641372	0.696055	0.92261	0.288991	47.647701
		32	10	0.025	500	0.865115	0.748308	0.662798	0.702962	0.923435	0.29145	47.202173
		8	50	0.025	500	0.865084	0.764336	0.635761	0.694145	0.921226	0.291554	47.183315
		32	10	0.05	100	0.865084	0.756395	0.648642	0.698387	0.92155	0.29239	47.031913
		8	10	0.025	500	0.864992	0.764066	0.635633	0.693957	0.921674	0.290844	47.312033
		32	10	0.2	20	0.864961	0.757939	0.645326	0.697114	0.920488	0.294732	46.607674
		8	1	0.1	500	0.864808	0.752756	0.653105	0.699399	0.922545	0.291764	47.145278
		32	10	0.1	100	0.864777	0.748626	0.66012	0.701593	0.92295	0.291274	47.234005
		8	1	0.1	100	0.864715	0.763255	0.635251	0.693395	0.920514	0.293092	46.904731
		8	10	0.1	100	0.864715	0.763416	0.634996	0.693309	0.920639	0.292815	46.954929
		8	10	0.4	100	0.864623	0.754863	0.648387	0.697585	0.921302	0.292915	46.936722
		32	1	0.2	20	0.864439	0.757089	0.64354	0.695712	0.920124	0.295428	46.481506

4. Train, Evaluate, and compare

You can see the parameters in left-hand columns and the output produced.

ACTIVITY: TRY DIFFERENT MODELS, INPUT FEATURES, AND SCALINGS

```
clf = svm.SVC(kernel='linear')
clf.fit(training_data, class_labels)
svm_prediction = clf.predict(test_features)
```

```
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(training_data, class_labels)
knn_prediction = clf.predict(test_features)
```

```
clf = DecisionTreeClassifier(random_state=0)
clf.fit(training_data, class_labels)
tree_prediction = clf.predict(test_features)
```

```
clf = MLPClassifier(solver='lbfgs', random_state=0, hidden_layer_sizes=[100])
clf.fit(training_data, class_labels)
tree_prediction = clf.predict(test_features)
```

SCIKIT LEARN

CSE 599 Prototyping Interactive Systems | Lecture 15 | May 21

Jon Froehlich • Jasper O'Leary (TA)