1. Welcome to the Nanodegree
   1. Welcome to the MLND Program
      1. Welcome to MLND
      2. Announcement
      3. Program Readiness
      4. What is Machine Learning
      5. Machine Learning vs. Traditional Coding
      6. Quiz: Applications of Machine Learning
      7. Connections to the GA Tech
      8. Program Outline
   2. What is Machine Learning
      1. Introduction to Machine Learning
      2. Decision Trees
      3. Decision Trees Quiz
      4. Decision Trees Answer
      5. Naïve Bayes
      6. Naïve Bayes Quiz
      7. Naïve Bayes Answer
      8. Gradient Descent
      9. Linear Regression Quiz
      10. Linear Regression Answer
      11. Logistic Regression Quiz
      12. Logistic Regression Answer
      13. Support Vector Machines
      14. Support Vector Machines Quiz
      15. Support Vector Machines Answer
      16. Neural Networks
      17. Kernel Method
      18. Kernel Method Quiz
      19. Kernel Method Answer
      20. Recap and Challenge
      21. K-means Clustering
      22. Hierarchical Clustering
      23. Practice Project: Detect Spam
          1. https://github.com/udacity/machine-learning/blob/master/projects/practice\_projects/naive\_bayes\_tutorial/Naive\_Bayes\_tutorial.ipynb
      24. Summary
   3. MLND Program Orientation
      1. Before the Program Orientation
      2. Introduction
      3. Projects and Progress
      4. Connecting with your community

## Connecting with Your Community

Your Nanodegree community will play a huge role in supporting you when you get stuck and in helping you deepen your learning. Getting to know your fellow students will also make your experience a lot more fun!

**To ask and answer questions, and to contribute to discussions, head to your**[**program forum**](http://discussions.udacity.com/categories)**. You can get there by clicking the Discussion link in the classroom and in the Resources tab in your Udacity Home.** You can search to see if someone has already asked a question related to yours, or you can make a new post if no one has. Chances are, someone else is wondering about the same thing you are, so don’t be shy!

In addition, students may connect with one another through [**Slack**](https://slack.com/), a team-oriented chat program. You can join the MLND Slack student community by following [**this link**](http://br-slack.udacity.com/) and registering your email. There are many content-related channels where you can speak with students about a particular concept. In addition, you can talk with **MLND graduates and alumni** to get a live perspective on the program in the #mlnd channel! You can find the student-run community wiki [**here**](https://github.com/machinelearningnanodegree/MLND/wiki).

* + 1. Support from the Udacity Team

## Support from the Udacity Team

The Udacity team is here to help you reach your Nanodegree program goals! You can interact with us in the following ways:

* **Forums**: Along with your student community, the Udacity team maintains a strong presence in the forum to help make sure your questions get answered and to connect you with other useful resources.
* **Project Reviews**: During the project submission process, your submissions will be reviewed by a qualified member of our support team, who will provide comments and helpful feedback on where your submission is strongest, and where your submission needs improvement. The reviews team will support your submissions all the way up to meeting specifications!
* **By email**: You can always contact the Machine Learning team with support-related questions using [**machine-support@udacity.com**](mailto:machine-support@udacity.com). Please make sure that you have exhausted all other options before doing so!

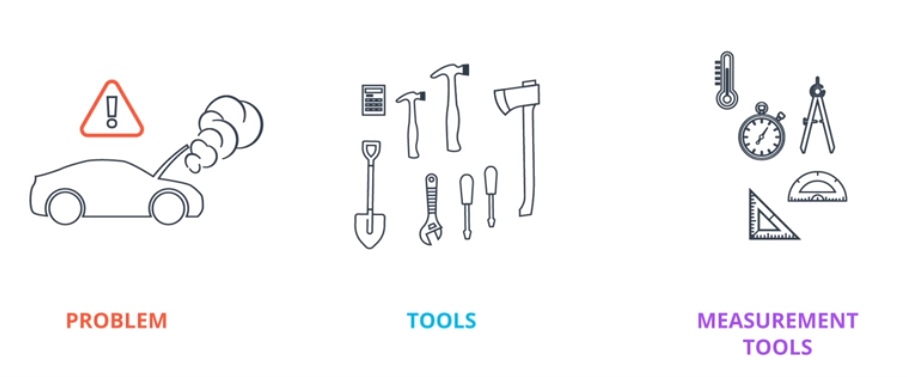
Find out more about the support we offer using the **Resources** tab in your Udacity Nanodegree Home.

* + 1. How does Project Submission Work?
    2. Integrity and Mindset
    3. Final Tips
    4. Wrapping up the Program Orientation

1. Evaluation and Validation
   1. Training Models
      1. Intro

How well is my model doing? Learn metrics to tell us if it is good or not. The second question is: How do we improve the model based on these metrics?

* + 1. Outline

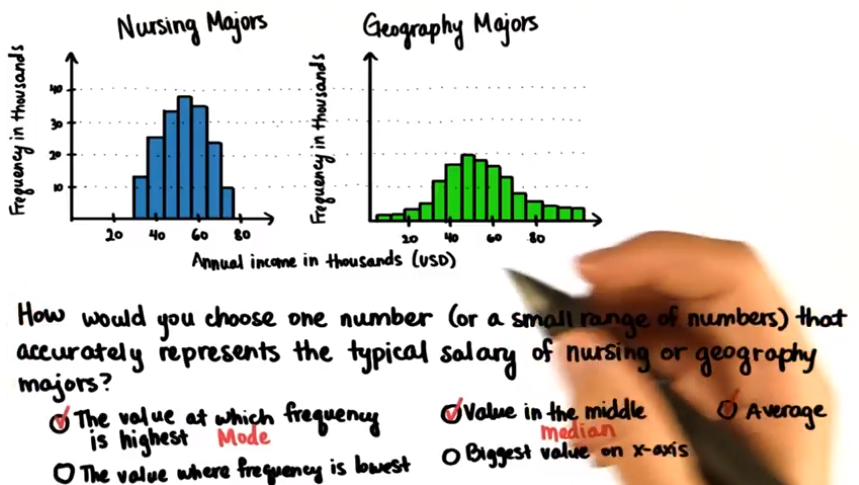


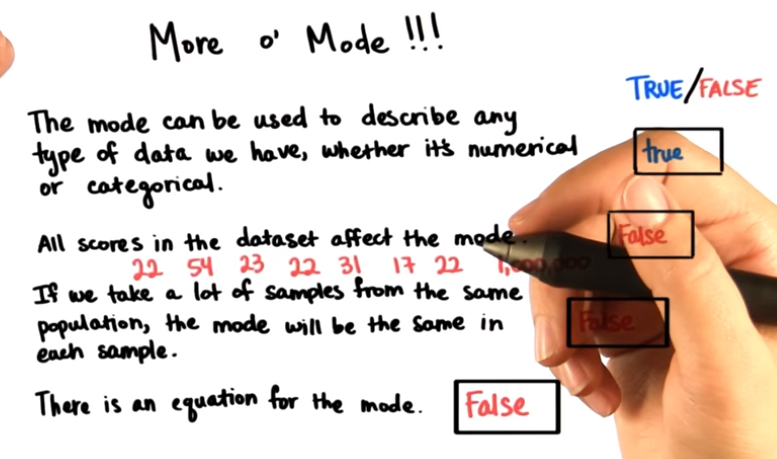
* + 1. Stats Refresher

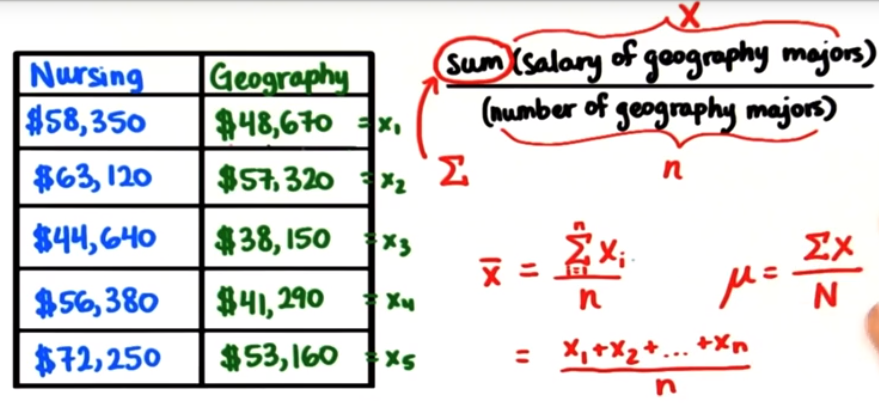
# Statistics Refresher

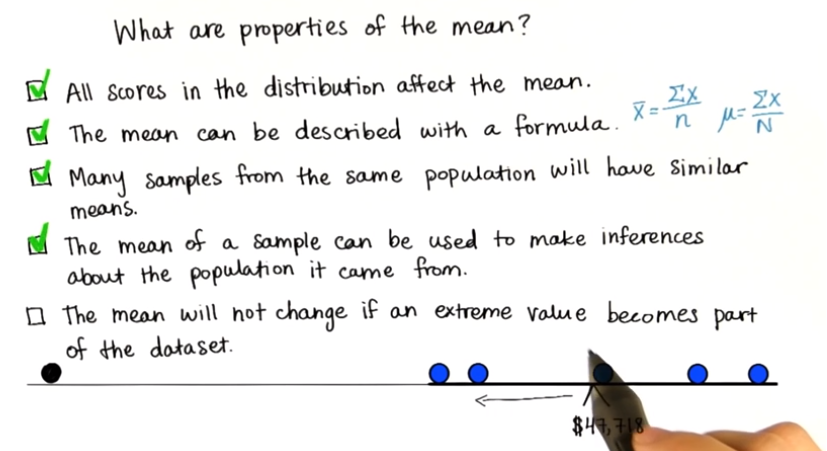
In this class, we'll assume familiarity with concepts in statistics such as mean, median, variance, etc. If it's been a while since you learned these and you feel that you need a refresher, here are some videos and quizzes in the **Extracurricular Section** which we encourage you to check out.

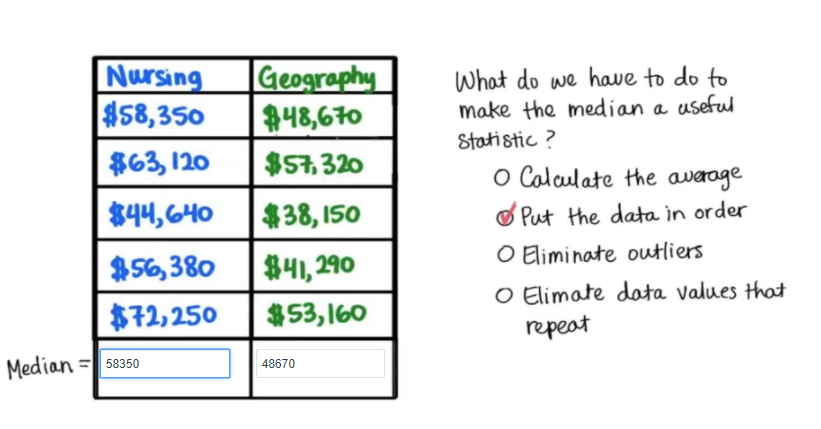
* [**Measures of Central Tendency**](https://classroom.udacity.com/nanodegrees/nd009-br/parts/d2fec38c-e52a-4370-a633-963baf638733/modules/6bebd83e-e02e-44dd-97ec-34e1129d32db/lessons/5422370497/concepts/1f585461-1429-401d-bc2a-238567a3e255): Mean, Median, Mode.
* [**Variability of Data**](https://classroom.udacity.com/nanodegrees/nd009-br/parts/d2fec38c-e52a-4370-a633-963baf638733/modules/6bebd83e-e02e-44dd-97ec-34e1129d32db/lessons/5452179865/concepts/edebdc1f-699d-46f7-a7f9-3a396c9ece9a): Inter Quartile Range, Outliers, Standard Deviation, Bessel's Correction.

****

****

****

****

****

* + 1. Numpy and Pandas

# Numpy, Pandas, and scikit-learn Refresher

Over the next few sections, you'll learn the basics in **Numpy**, **Pandas**, and **scikit-learn**, in order to train and test your machine learning models. However, if you feel that you need a more thorough introduction, here are some videos and quizzes in the **Extracurricular Section** of the course that we encourage you to check out the following content in elective part.

* [**Numpy and Pandas Tutorial**](https://classroom.udacity.com/nanodegrees/nd009-br/parts/d2fec38c-e52a-4370-a633-963baf638733/modules/6bebd83e-e02e-44dd-97ec-34e1129d32db/lessons/5454078888/concepts/54758106780923)
* [**Scikit Learn Tutorial**](https://classroom.udacity.com/nanodegrees/nd009-br/parts/d2fec38c-e52a-4370-a633-963baf638733/modules/6bebd83e-e02e-44dd-97ec-34e1129d32db/lessons/5430830847/concepts/6a282140-f39d-4b43-8c91-539571a30fad)
  + 1. Tuning Parameters Automatically

**How to train, test, evaluate and validate models in order to make the best decisions with our data.**

* + 1. Data Refresher

# Data Refresher

If you have previously played with data, and have an intuition about it, you can click on next. However, if you'd like to get some practice into what types of data there are, and get a hands-on exercise working on real data, here are some links in the **Extracurricular Section** that we encourage you to look at.

* [**Nature of Data**](https://classroom.udacity.com/nanodegrees/nd009-br/parts/d2fec38c-e52a-4370-a633-963baf638733/modules/6bebd83e-e02e-44dd-97ec-34e1129d32db/lessons/8034555385/concepts/6785689350923): Types of data (numerical, categorical, time series, etc.)
* Data types 1 – Numeric Data
  + Discrete or continuous
* Data types 2 – Categorical Data
  + Represents characteristics, can take on numerical values, but they don’t have mathematical meaning (1 – male, 2 – female)
  + Ordinal data (very low, low, average, high, very high)
* Data types 3 – Time Series Data
  + Collection of observations obtained through repeated measurements over time. Pretty similar to numerical data, but rather than having a bunch of numerical values which don’t have any time ordering, time series data does have some implied ordering. There’s a first data point collected and a last data point collected.
* Treatment of categorical data

Many algorithms assume that input data is numerical. For categorical data, this often means converting categorical data into numerical data that represents the same patterns.

One standard way of doing this is with [**one-hot encoding**](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html). There are built-in methods for this in scikit-learn.

Essentially, a categorical feature with 3 possible values is converted into three binary features corresponding to the values. The new feature corresponding to the category a datum belongs to has value 1, while the other new features have value 0.

For example, in a dataset on baseball players, one feature might be "Handedness" which can take values "left" or "right". Then the data:

* Joe
  + Handedness: right
* Jim
  + Handedness: left

Would become:

* Joe
  + Handedness/right: 1
  + Handedness/left: 0
* Jim
  + Handedness/right: 0
  + Handedness/left: 1

For ordinal data, it often makes sense to simply assign the values to integers. So the following data:

* Joe
  + Skill: low
* Jim
  + Skill: medium
* Jane
  + Skill: high

Would become:

* Joe
  + Skill: 0
* Jim
  + Skill: 1
* Jane
  + Skill: 2

These approaches are not all that is possible. However in general these simple approaches suffice, and if there is a reason to use another encoding that will be subject to the nature of the data.

* Encoding using sklearn

## Encoding using sklearn

Encoding in sklearn is done using the **[preprocessing module](http://scikit-learn.org/stable/modules/classes.html" \l "module-sklearn.preprocessing" \t "_blank)** which comes with a variety of options of manipulating data before going into the analysis of data. We will focus on two forms of encoding for now, the **[LabelEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html" \t "_blank)** and the **[OneHotEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html" \t "_blank)**.

### Label Encoder

First, we have to import the preprocessing library.

**from** sklearn **import** preprocessing

Let's create a dummy dataframe named data with a column whose values we want to transform from categories to integers.

*# creating sample data*

sample\_data = {'name': ['Ray', 'Adam', 'Jason', 'Varun', 'Xiao'],

'health':['fit', 'slim', 'obese', 'fit', 'slim']}

*# storing sample data in the form of a dataframe*

data = pandas.DataFrame(sample\_data, columns = ['name', 'health'])

We have 3 different labels that we are looking to categorize: slim, fit, obese. To do this, we will call LabelEncoder() and fit it to the column we are looking to categorize.

**label\_encoder** = preprocessing.**LabelEncoder**()

**label\_encoder**.fit(**data**['health'])

Once you have fit the label encoder to the column you want to encode, you can then transform that column to integer data based on the categories found in that column. That can be done as follows:

label\_encoder.transform(data['health'])

This will give you the output:

array([0, 2, 1, 0, 2])

You can combine the fit and transform statements above by using label\_encoder.fit\_transform(data['health']).

The string categorical health data has been mapped as follows:

fit : 0

obese: 1

slim: 2

One thing to keep in mind when encoding data is the fact that you do not want to skew your analysis because of the numbers that are assigned to your categories. For example, in the above example, slim is assigned a value 2 and obese a value 1. This is not to say that the intention here is to have slim be a value that is empirically twice is likely to affect your analysis as compared to obese. In such situations it is better to one-hot encode your data as all categories are assigned a 0 or a 1 value thereby removing any unwanted biases that may creep in if you simply label encode your data.

### One-hot Encoder

If we were to apply the one-hot transformation to the same example we had above, we'd do it in Pandas using **[get\_dummies](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html" \t "_blank)** as follows:

pandas.get\_dummies(data['health'])

We could do this in sklearn on the label encoded data using **[OneHotEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html" \t "_blank)** as follows:

**ohe** = preprocessing.OneHotEncoder() *# creating OneHotEncoder object*

label\_encoded\_data = label\_encoder.fit\_transform(data['health'])

ohe.fit\_transform(label\_encoded\_data.reshape(-1,1))

* Quiz: One-Hot Encoding

**# In this exercise we'll load the titanic data (from Project 0)**

**# And then perform one-hot encoding on the feature names**

**import numpy as np**

**import pandas as pd**

**# Load the dataset**

**X = pd.read\_csv('titanic\_data.csv')**

**# Limit to categorical data**

**X = X.select\_dtypes(include=[object])**

**from sklearn.preprocessing import LabelEncoder**

**from sklearn.preprocessing import OneHotEncoder**

**# TODO: Create a LabelEncoder object, which will turn all labels present in**

**# in each feature to numbers.**

**# HINT: Use LabelEncoder()**

**le = LabelEncoder()**

**# TODO: For each feature in X, apply the LabelEncoder's fit\_transform**

**# function, which will first learn the labels for the feature (fit)**

**# and then change the labels to numbers (transform).**

**for feature in X:**

**# HINT: use fit\_transform on X[feature] using the LabelEncoder() object**

**X[feature] = le.fit\_transform(X[feature])**

**# TODO: Create a OneHotEncoder object, which will create a feature for each**

**# label present in the data.**

**# HINT: Use OneHotEncoder()**

**ohe = OneHotEncoder()**

**# TODO: Apply the OneHotEncoder's fit\_transform function to all of X, which will**

**# first learn of all the (now numerical) labels in the data (fit), and then**

**# change the data to one-hot encoded entries (transform).**

**# HINT: Use fit\_transform on X using the OneHotEncoder() object**

**xt = ohe.fit\_transform(X)**

**onehotlabels = xt**

* Time series data leakage

When dealing with time-series data, it can be tempting to simply disregard the timing structure and simply treat it as the appropriate form of categorical or numerical data.

One important concern, however, is that if you are building a predictive project looking at forecasting future data points. In this case, it is important **NOT** to use the future as a source of information! Since "hindsight is 20/20" and retrodictions are much easier than predictions, in predictive tasks it's generally a good idea to use a training set made up of data from before a certain point, a validation set of data from some dates beyond that, and testing data leading up to the present. This way your algorithm won't overfit by learning future trends.

* A hands-on example

In the next section we'll explore the famous [**Enron Email Dataset**](https://www.cs.cmu.edu/~./enron/) which was the focus of much of the Introduction to Machine Learning course.

While this specific dataset will play a less central role in this Nanodegree program, we will return to it a few times as an example to get practice with various techniques as they are introduced.

You can download our copy of the dataset [**here**](https://github.com/udacity/ud120-projects), along with the starting code for a variety of mini-projects. None of these mini-projects are required for completing the Nanodegree program, but they are great practice!

* [**Datasets and Questions**](https://classroom.udacity.com/nanodegrees/nd009-br/parts/d2fec38c-e52a-4370-a633-963baf638733/modules/6bebd83e-e02e-44dd-97ec-34e1129d32db/lessons/2291728537/concepts/28772285450923): A thorough exercise on Enron Data where you can apply the previously learned concepts.
  + 1. Loading data into Pandas

# Loading data into Pandas

In this section we'll overview the basics on how to open a dataset in Pandas. If you want to go more in detail, please check the [**Pandas Documentation**](http://pandas.pydata.org/pandas-docs/stable/).

For this, we'll be using the dataset called 2\_class\_data.csv, which looks as follows:

**[[](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/0fdcf21e-19f2-4e1d-a9b9-e8f73c8cbda4)](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/0fdcf21e-19f2-4e1d-a9b9-e8f73c8cbda4)**

To open this dataset, which comes as a csv file, we'll use the command read\_csv in Pandas:

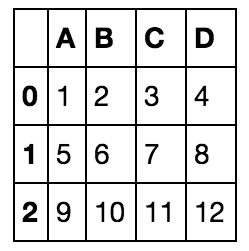
**import** pandas

data = pandas.read\_csv("file\_name.csv")

* + 1. Numpy Arrays

# Numpy Arrays

Now that we've loaded the data in Pandas, we need to split the input and output into numpy arrays, in order to apply the classifiers in scikit learn. This is done in the following way: Say we have a pandas dataframe called df, like the following, with four columns labeled A, B, C, D:

**[[](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/4c706530-12a9-46ca-889f-92941b0883e5)](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/4c706530-12a9-46ca-889f-92941b0883e5)**

If we want to extract column A, we do the following:

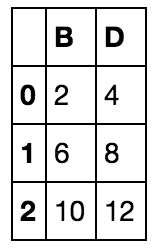
>> df['A']

0 1

1 5

2 9

Name: A, dtype: int64

**[[](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/4c706530-12a9-46ca-889f-92941b0883e5)](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/4c706530-12a9-46ca-889f-92941b0883e5)**

Now, try it yourself! Working with the same dataframe that we loaded in pandas previously, split it into the features X, and the labels y.

**import pandas as pd**

**import numpy as np**

**data = pd.read\_csv("data.csv")**

**# TODO: Separate the features and the labels into arrays called X and y**

**X = np.array(data[['x1', 'x2']])**

**y = np.array(data['y'])**

* + 1. Training models in sklearn

# Training Models in scikit learn

In this section, we'll still be working with the dataset of the previous sections.

**[[](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/9ed98cab-aa00-43a3-b8dc-ef7ed0bee156)](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/9ed98cab-aa00-43a3-b8dc-ef7ed0bee156)**

In the last section, we learned most of the most important classification algorithms in Machine Learning, including the following:

* Logistic Regression
* Neural Networks
* Decision Trees
* Support Vector Machines

Now, we'll have the chance to use them in real data! In sklearn, this is very easy, all we do is define our classifier, and then use the following line to fit the classifier to the data (which we call X, y):

classifier.fit(X,y)

Here are the main classifiers we define, together with the package we must import:

### Logistic Regression

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression()

### Neural Networks

(note: This is only available on versions 0.18 or higher of scikit-learn)

from sklearn.neural\_network import MLPClassifier

classifier = MLPClassifier()

### Decision Trees

from sklearn.ensemble import GradientBoostingClassifier

classifier = GradientBoostingClassifier()

### Support Vector Machines

from sklearn.svm import SVC

classifier = SVC()

### Example: Logistic Regression

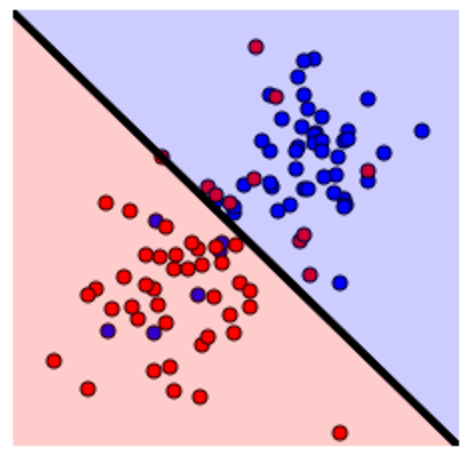
Let's do an end-to-end example on how to read data and train our classifier. Let's say we carry our X and y from the previous section. Then, the following commands will train the Logistic Regression classifier:

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression()

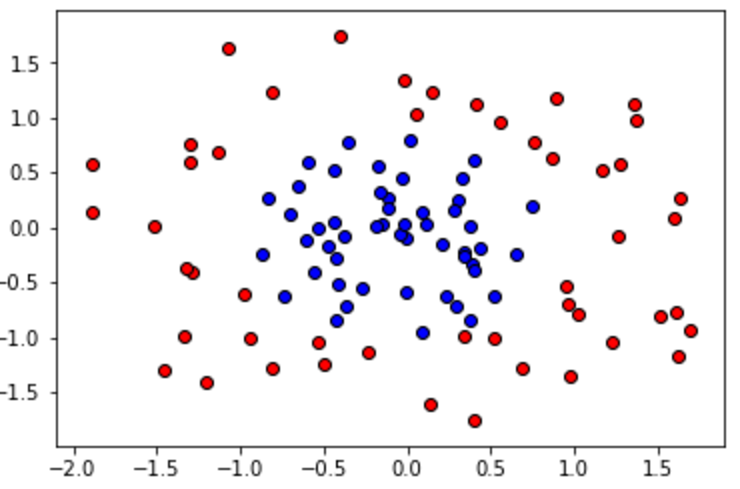
classifier.fit(X,y)

This gives us the following boundary:

**[[](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/9ed98cab-aa00-43a3-b8dc-ef7ed0bee156)](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/9ed98cab-aa00-43a3-b8dc-ef7ed0bee156)**

# Quiz: Train your own model

Now, it's your turn to shine! In the quiz below, we'll work with the following dataset:

**[[](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/9ed98cab-aa00-43a3-b8dc-ef7ed0bee156)](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/9ed98cab-aa00-43a3-b8dc-ef7ed0bee156)**

Your goal is to use one of the classifiers above, between Logistic Regression, Decision Trees, or Support Vector Machines (sorry, Neural Networks are still not available in this version of sklearn, but we will be upgrading soon!), to see which one will fit the data better. Click on Test Run to see the graphical output of your classifier, and in the quiz underneath this, enter the classifier that you think fit the data better!

import pandas

import numpy

# Read the data

data = pandas.read\_csv('data.csv')

# Split the data into X and y

X = numpy.array(data[['x1', 'x2']])

y = numpy.array(data['y'])

# import statements for the classification algorithms

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.svm import SVC

# TODO: Pick an algorithm from the list:

# - Logistic Regression

# - Decision Trees

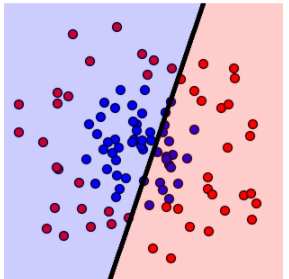
# - Support Vector Machines

# Define a classifier (bonus: Specify some parameters!)

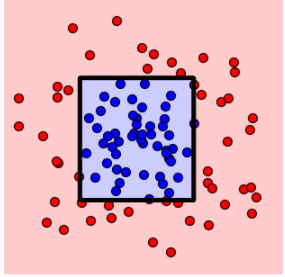
# and use it to fit the data

# Click on `Test Run` to see how your algorithm fit the data!

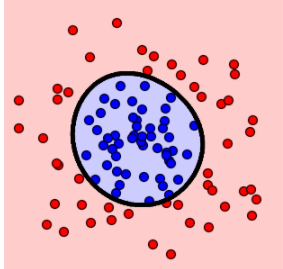
classifier = LogisticRegression()



#classifier = GradientBoostingClassifier()



classifier = SVC()

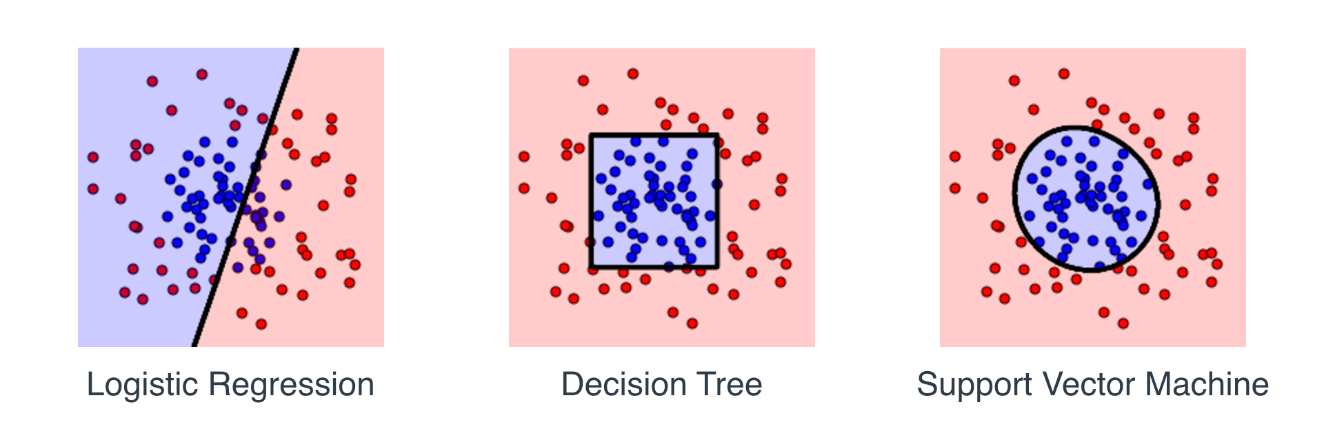


classifier.fit(X,y)

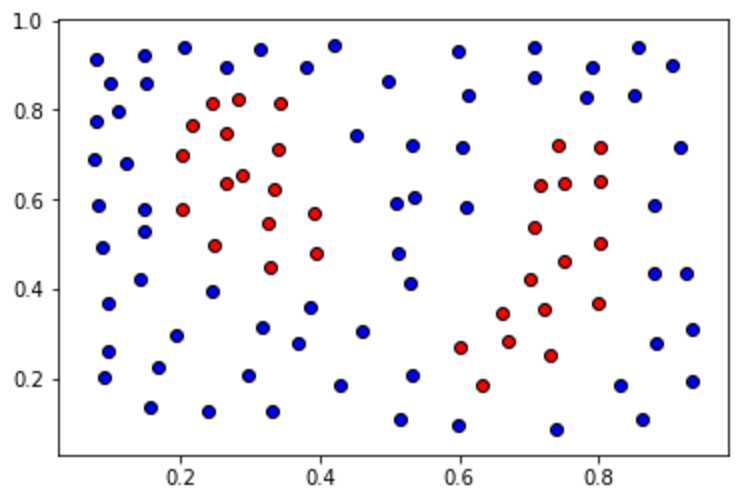
* + 1. Tuning parameters manually

# Tuning Parameters

So it looks like 2 out of 3 algorithms worked well last time, right? This are the graphs you probably got:

**[[](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/d7f3ec93-ab4e-463b-947d-47779ab4bbed)](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/d7f3ec93-ab4e-463b-947d-47779ab4bbed)**

It seems that Logistic Regression didn't do so well, as it's a linear algorithm. Decision Trees managed to bound the data well (question: Why does the area bounded by a decision tree look like that?), and the SVM also did pretty well. Now, let's try a slightly harder dataset, as follows:

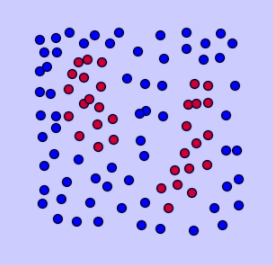
**[[](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/d7f3ec93-ab4e-463b-947d-47779ab4bbed)](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/89711201-5ae5-403d-a1fc-fd40f412606b/concepts/d7f3ec93-ab4e-463b-947d-47779ab4bbed)**

Let's try to fit this data with an SVM Classifier, as follows:

**classifier** = SVC()

classifier.fit(X,y)

If we do this, it will fail (you'll have the chance to try below).



However, it seems that maybe we're not exploring all the power of an SVM Classifier. For starters, are we using the right kernel? We can use, for example, a polynomial kernel of degree 2, as follows:

classifier = SVC(kernel = 'poly', degree = 2)

Let's try it ourselves, let's play with some of these parameters. We'll learn more about these later, but here are some values you can play with (These are parameters that we'll learn later. For now, we can use them as a black box.):

* kernel: linear, poly, rbf.
* degree (integer): This is the degree of the polynomial kernel, if that's the kernel you picked.
* gamma (integer): The gamma parameter.
* C (integer): The C parameter.

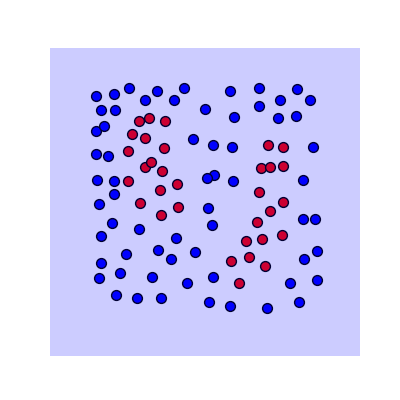
In the quiz below, you can play with these parameters. Try to tune them in such a way that they bound the desired area!

**classifier = SVC(kernel = 'poly', degree = 2)**

**classifier = SVC(kernel = 'poly', degree = 3)**

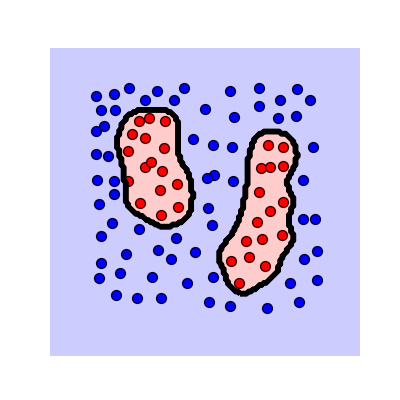
**classifier = SVC(kernel = 'rbf', degree = 1)**

**classifier = SVC(kernel = 'rbf', degree = 2)**

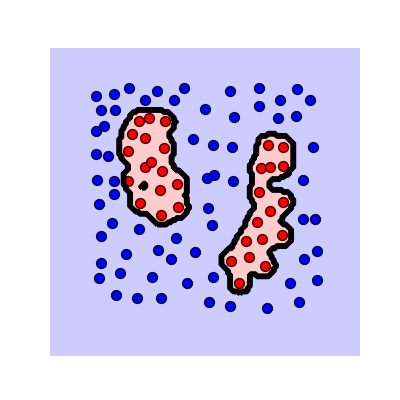


Solução:

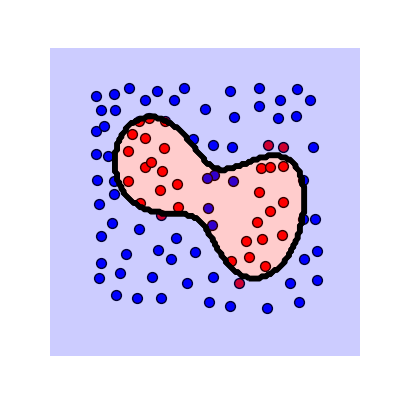
**classifier = SVC(kernel = 'rbf', gamma = 200)**



**Teste: classifier = SVC(kernel = 'rbf', gamma = 500)**

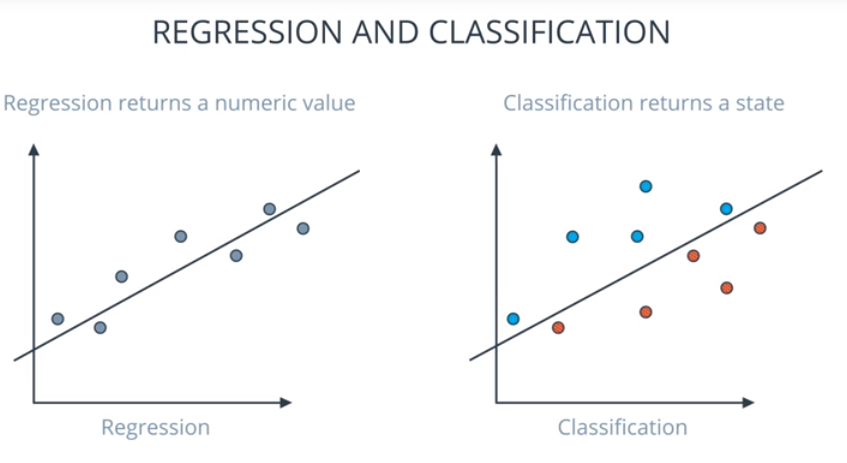


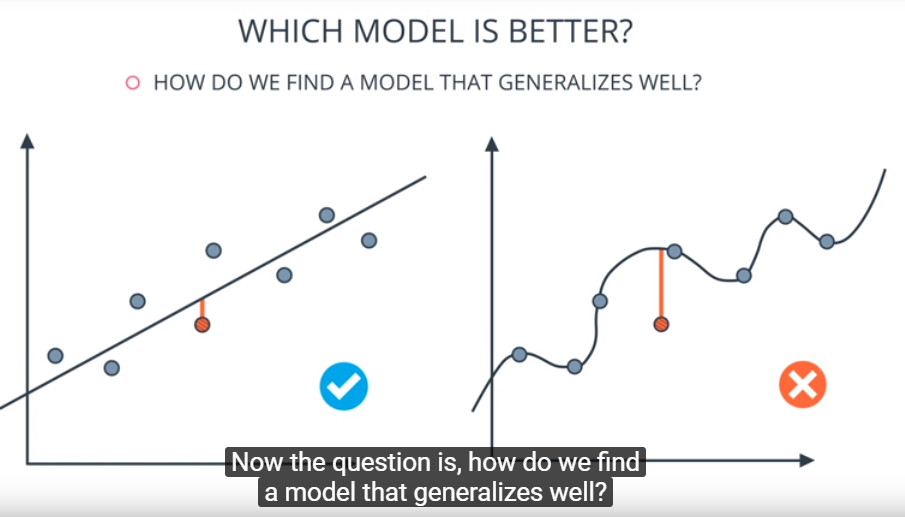
**Teste: classifier = SVC(kernel = 'rbf', gamma = 10)**

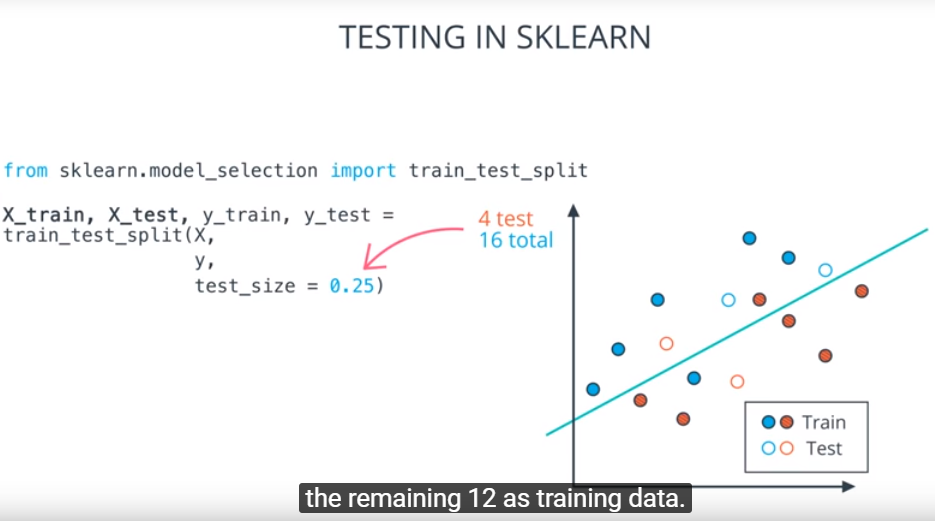


Note: The quiz is not graded, but if you want to see a solution that works, hit Submit.

* 1. Testing Models
     1. Testing your models







* + 1. Quiz: Testing in sklearn

Splitting a dataset into training and testing data is very easy with sklearn. All we need is the command train\_test\_split. The function takes as inputs X and y, and returns four things:

* X\_train: The training input
* X\_test: The testing input
* y\_train: The training labels
* y\_test: The testing labels

The call to the function looks as follows:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25)

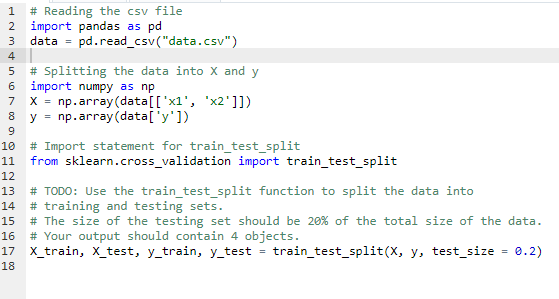
The last parameter, test\_size is the percentage of the points that we want to use as testing. In the above call, we are using 25% of our points for testing, and 75% for training.

Let's practice! We'll again use the dataset from the previous section:

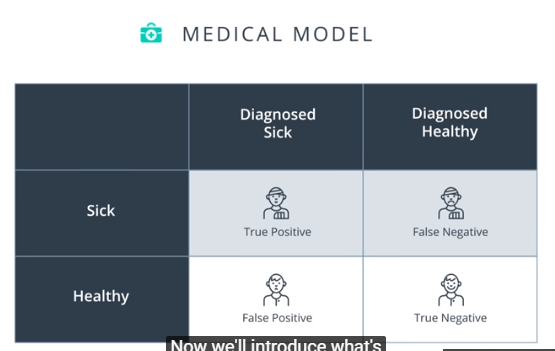
**[[](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/e7deb147-edf7-4673-84a9-056dc13f69ac/concepts/0ebede2d-a9a4-4191-86b6-81c1feb016a8)](https://classroom.udacity.com/nanodegrees/nd009-br/parts/8f0114b4-e778-4fce-ba89-e9a10a2026e0/modules/6d3c5e76-3df0-48e7-9ae5-05acc603b10f/lessons/e7deb147-edf7-4673-84a9-056dc13f69ac/concepts/0ebede2d-a9a4-4191-86b6-81c1feb016a8)**

In the following quiz, use the train\_test\_split function to split the dataset into training and testing sets. The size of the testing set must be **20%** of the total size of the data. Call your training sets X\_train and y\_train, and your testing sets X\_test and y\_test.

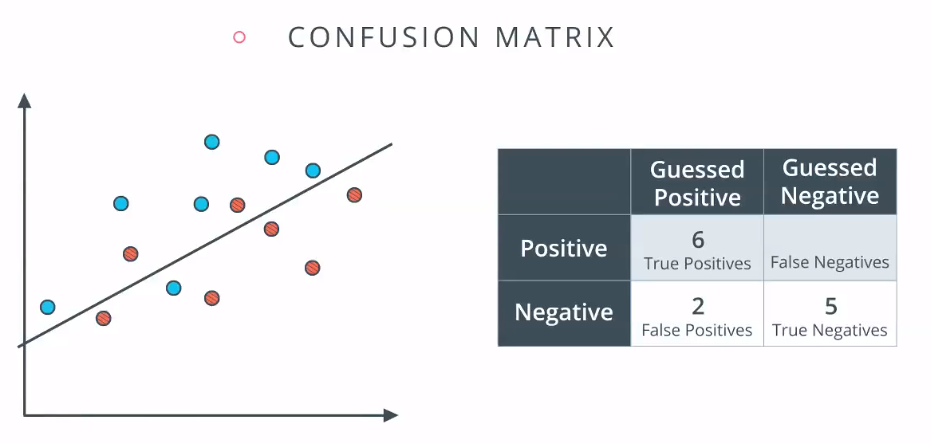
Click on Test Run to see a visualization of the results, where the training set will be drawn as circles, and the testing set as squares. Then when you're done, click on Submit to check your code!



* 1. Evaluation Metrics
     1. Confusion Matrix

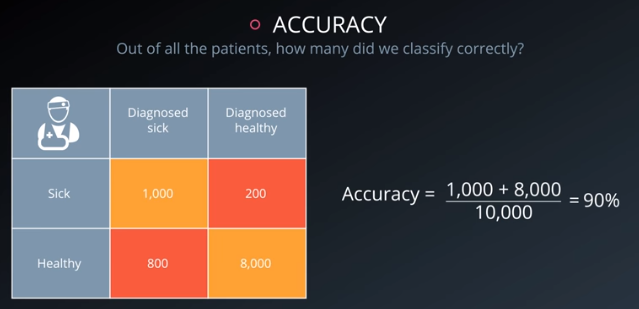
****

* + 1. Confusion Matrix 2



* + 1. Accuracy

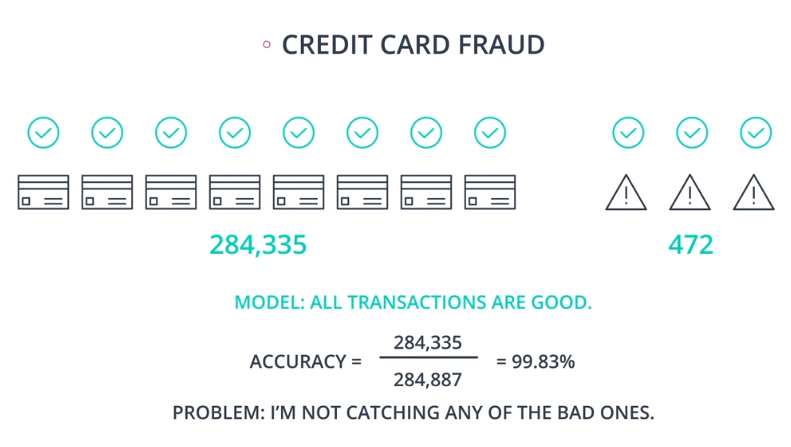
Out of all the patients, how many did we classify correctly? Accuracy = Number of correctly classified points/number of total points

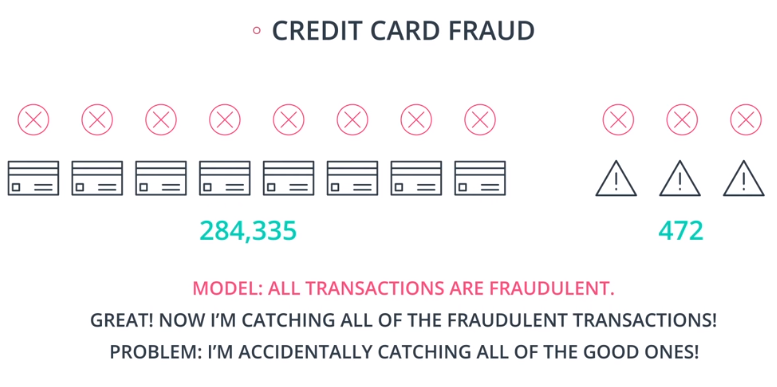


* + 1. Accuracy 2



* + 1. When accuracy won’t work



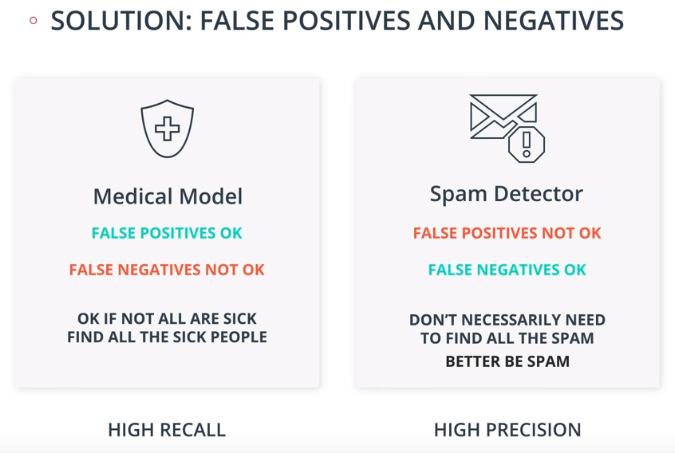


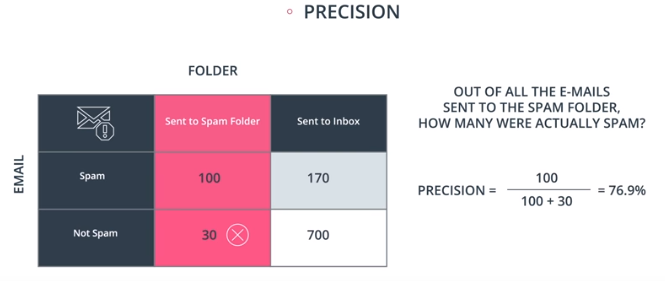
* + 1. False negatives and positives

In the medical example, which is worse? A false positive (diagnosed as sick, but healthy person) or a false negative (diagnosed as not sick, but actually sick): **false negative!** Correct! A False Positive implies sending a healthy person to get more tests. This is slightly inconvenient, but ok. A False Negative implies sending a sick person home, which can be disastrous!

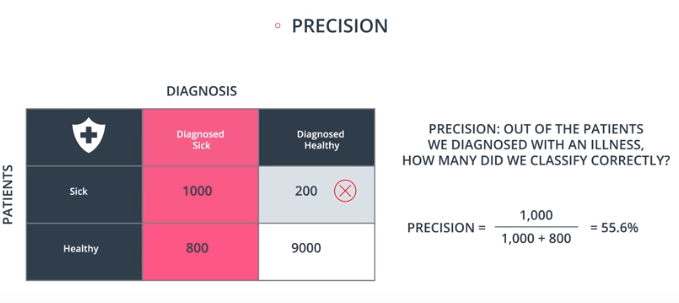
In the spam example, which is worse? A false positive (sent to spam, but not spam) or false negative (not sent to spam, but actually spam): **false positive!** Correct! A False Negative implies a spam message will make its way into your inbox. This is slightly inconvenient, but ok. A False Positive implies missing an e-mail from your dear grandma, which can be disastrous!

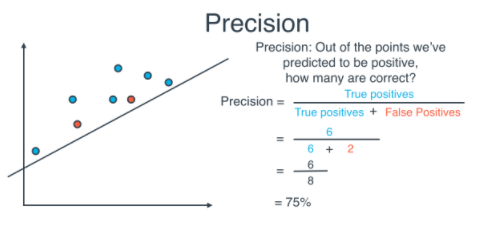
* + 1. Precision and Recall



* + 1. Precision

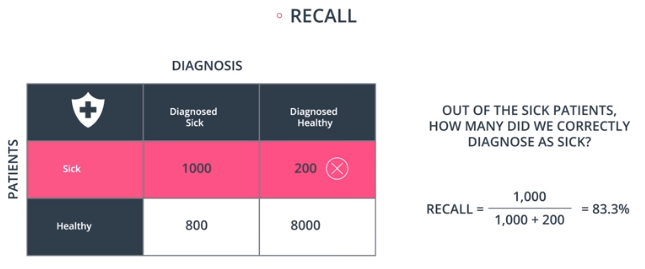
In the medical model, it’s ok that this model has a low precision.



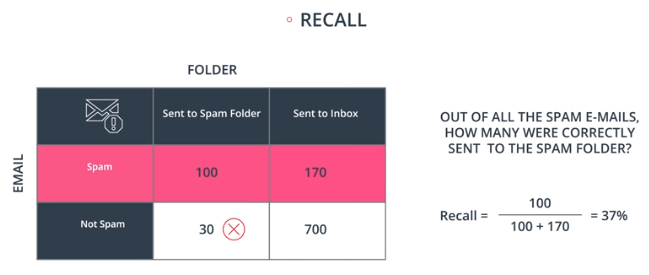


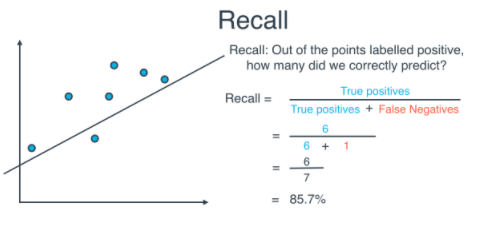
* + 1. Recall

Out of the sick patients, how many did we correctly diagnose as sick?



In the spam example, it’s ok that this model has a low recall.

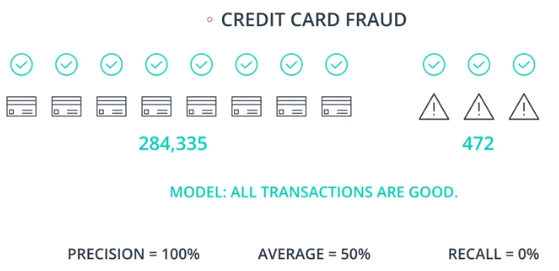


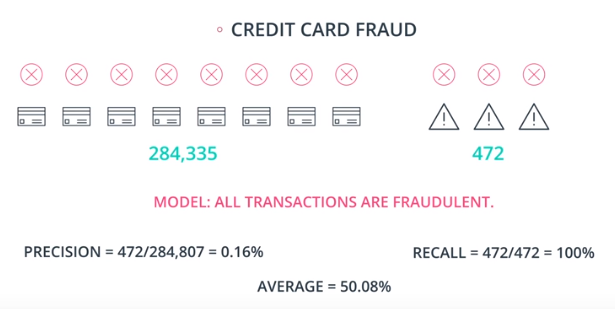


* + 1. F1 Score

We only want to have one score, not carrying 2 or more scores to make decisions. How to combine these two ones into one?

Bad idea to use the average as a metric:

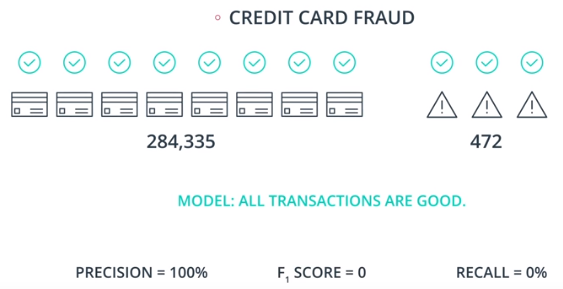




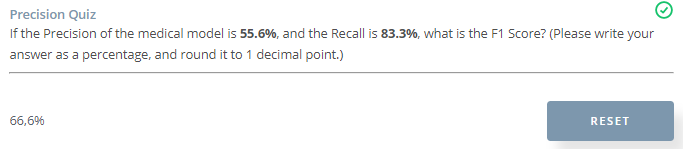
Another type of average: **harmonic mean**. Instead of average mean, where if one is good but the other is bad, then the average is ok. But the F1 score raises the flag is one of them is small.



The F1 score in the credit card example:

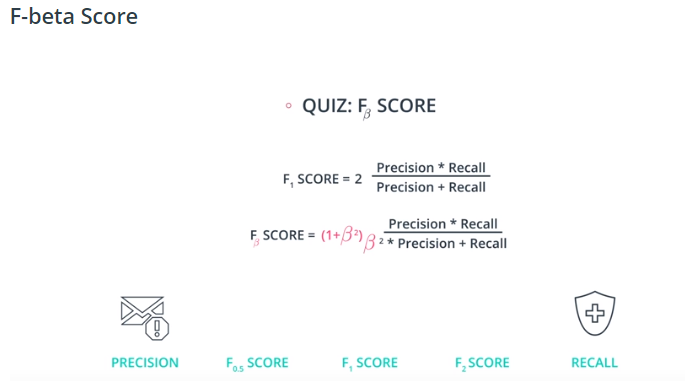


Quiz

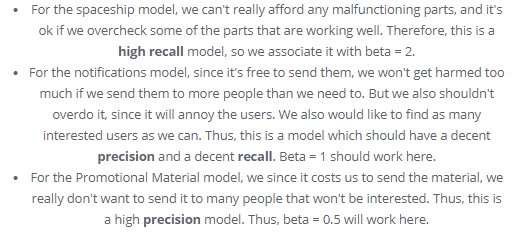


* + 1. F-beta Score

Let’s say our model cares a bit more about precision than recall. Then we want something more skewed towards precision.

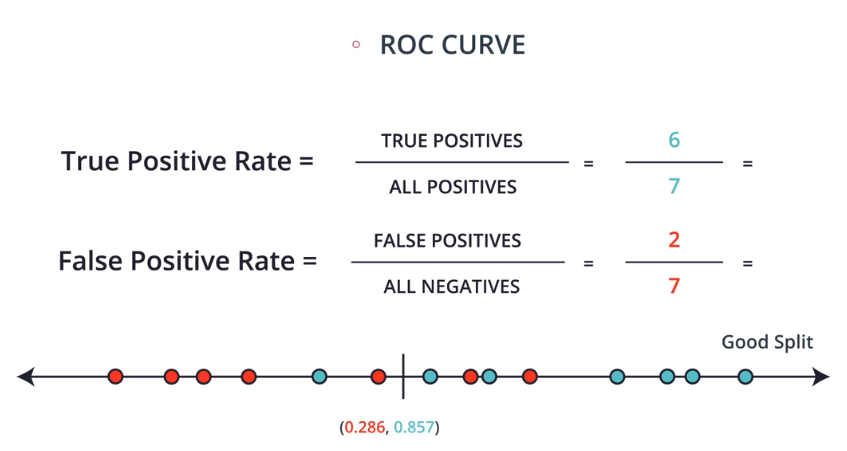


If we have the fraud detection example, which beta should we use? I would initially think that this needs to be a high recall model since we need to catch all the fraud cases and it’s okay if we accidentally detect and investigate something that are not fraud. If beta is too close to recall, then we’re sacrificing too much precision and we accidentally send their customers too many notifications about their transactions we taught them being fraudulent and they’re starting to get annoyed. Finding a good value of beta requires a lot of intuition of your data and a lot of experimentation.



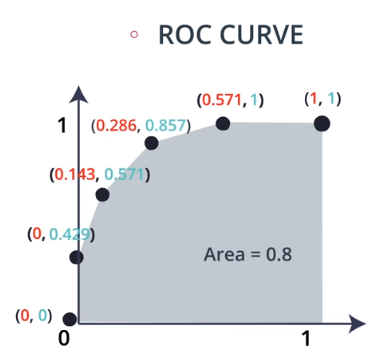
* + 1. ROC Curve

Another metric to evaluate model, receiver operating characteristic curve.

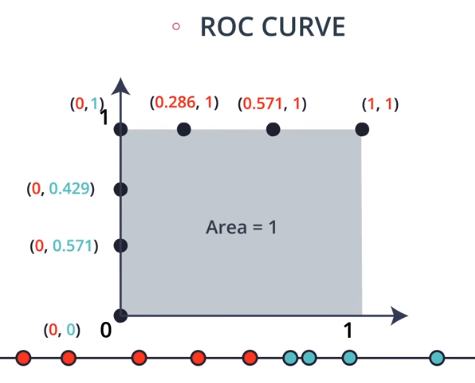




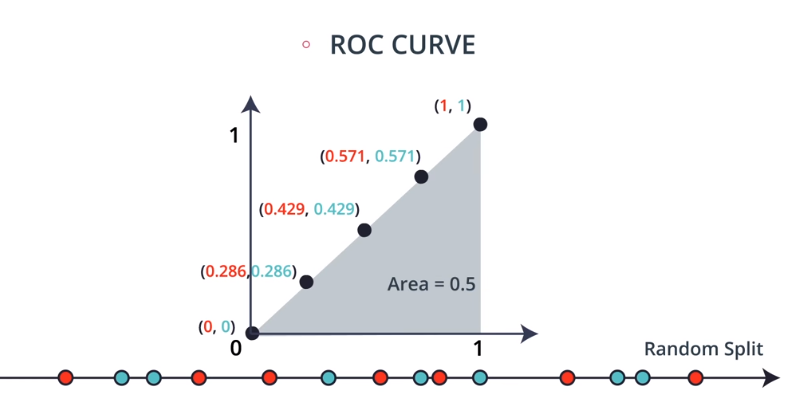
The ROC curve for the good split

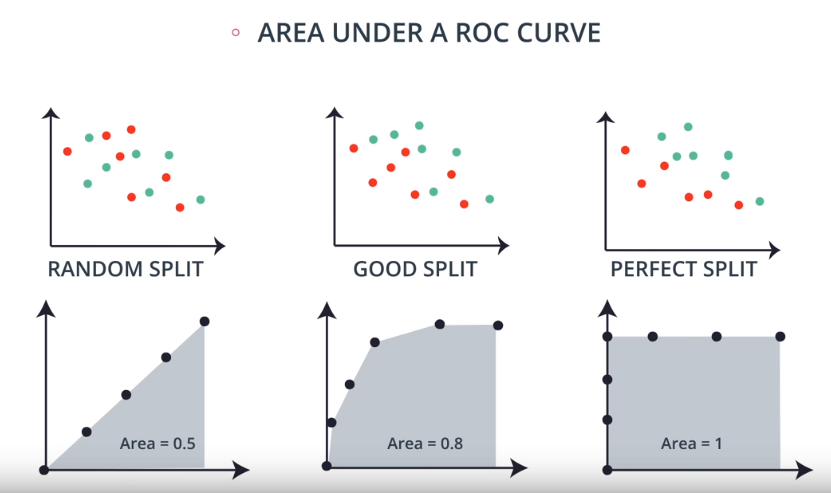


The ROC curve for the perfect split



The ROC curve for the random split

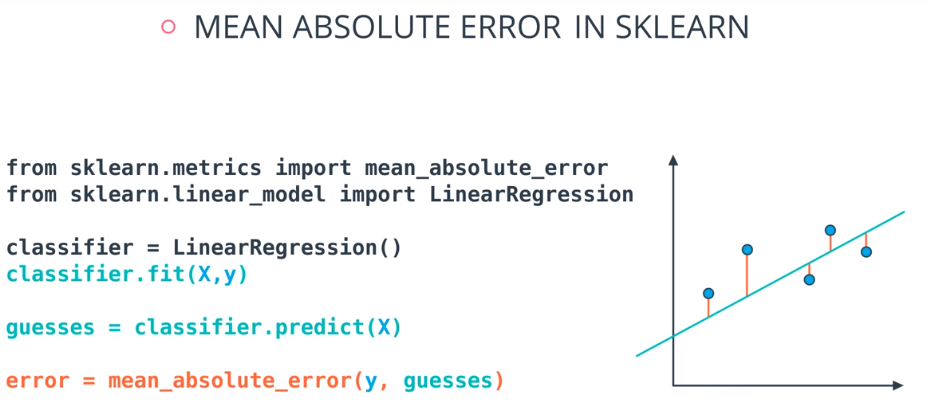




* + 1. Regression Metrics

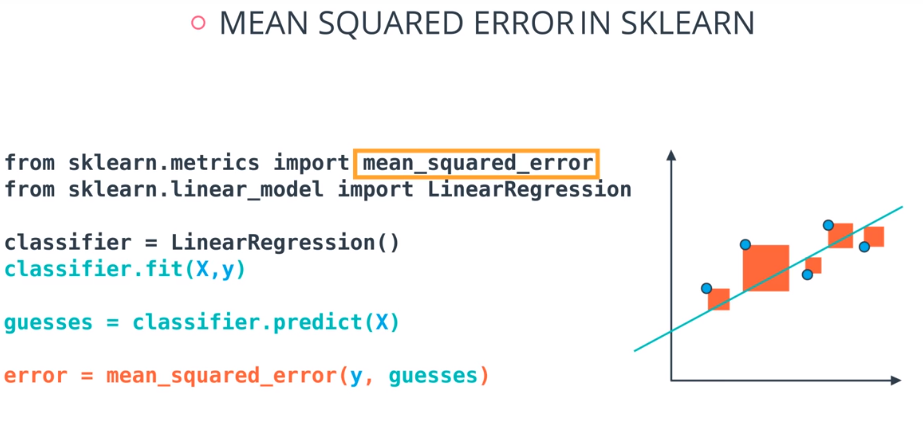
Now we’ll study metrics to evaluate regression models. The first is

* Mean Absolute Error

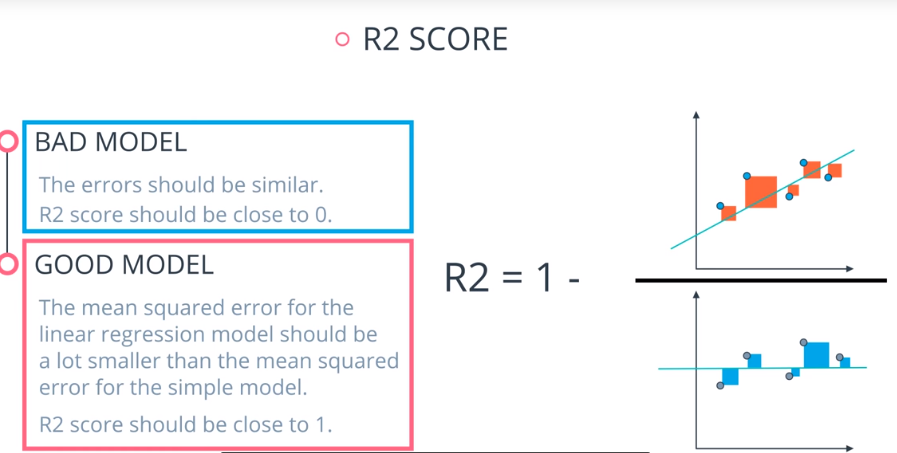


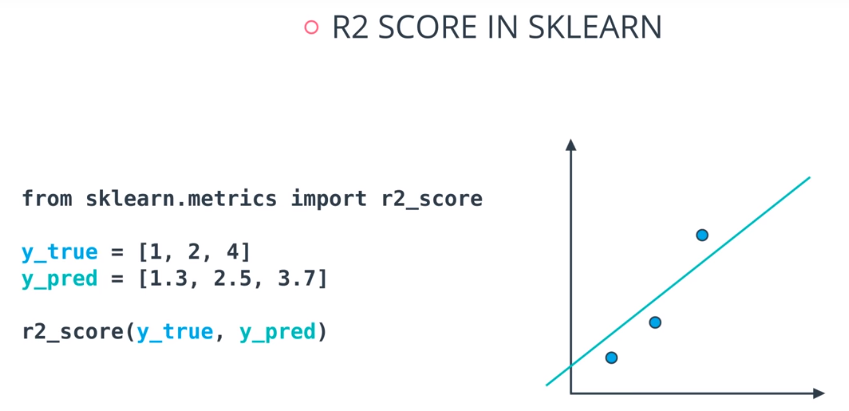
Mean absolute error has a problem, the absolute value function is not differentiable and this may not be good if we want to use methods such as gradient descent.

* To solve this problem, we use the more common **mean squared error**, for this metric, we add the squares of the distances between the points and the line.



* Very common regression metric called R2 Score. It’s based on comparing our model to the simplest possible model. The simplest model would be get the average and draw a horizontal line through them and then we can calculate the mean squared error for this model. Then we subtract from 1 the division of MSE(model) per MSE(simplest model)



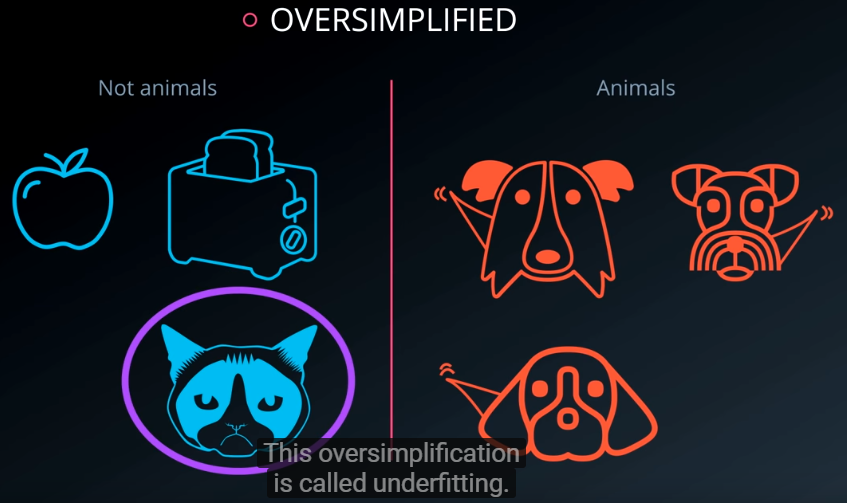


* 1. Detecting Errors
     1. Types of Errors
* Overfitting

Model is too specific. It does well in the training set but horrible in testing.

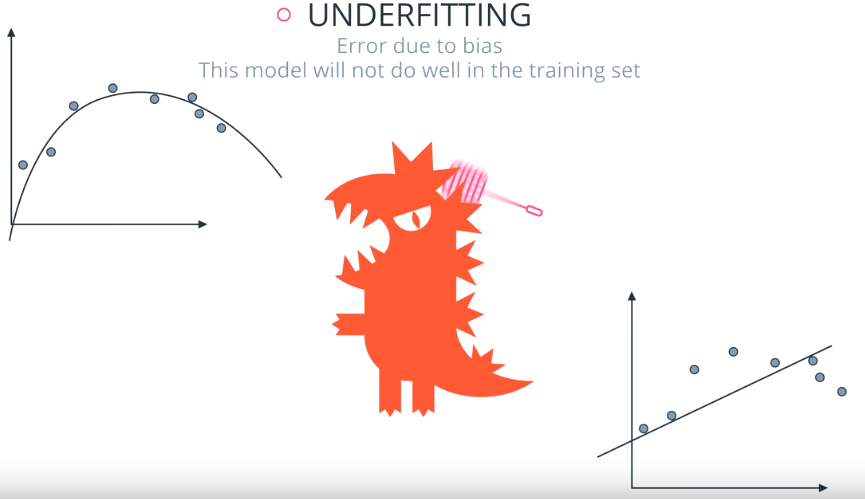


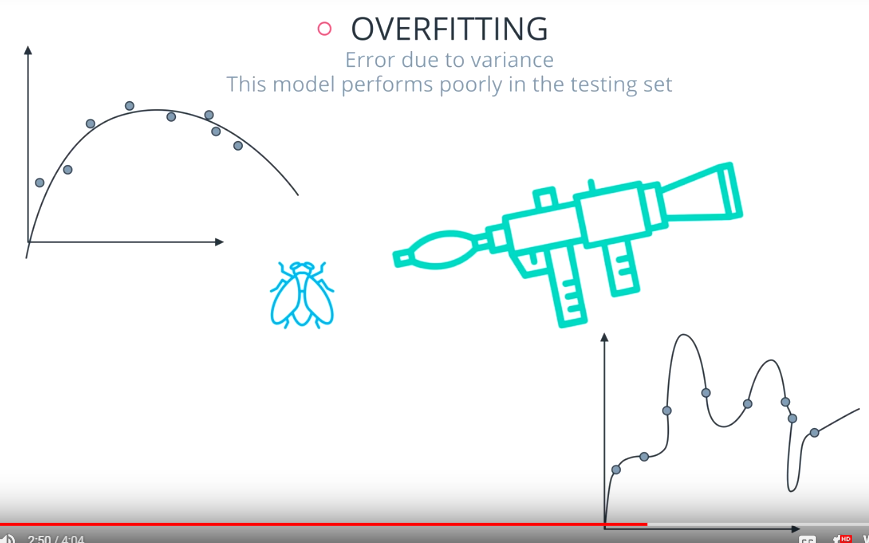
* Underfitting

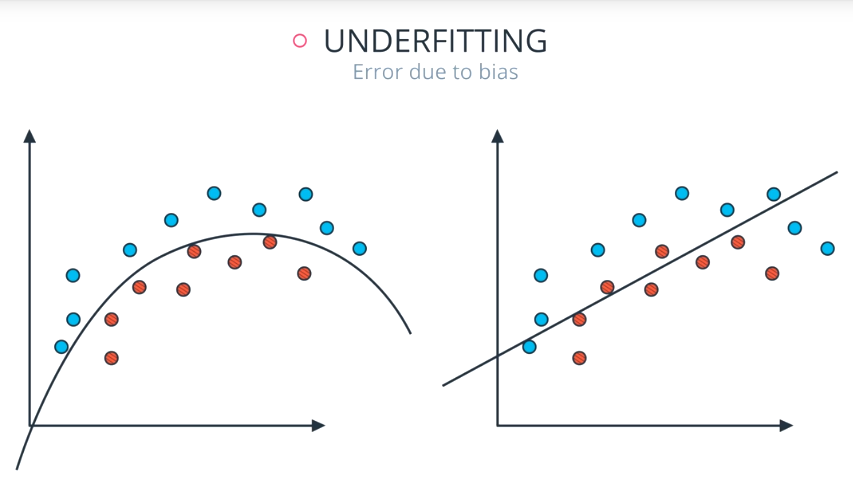


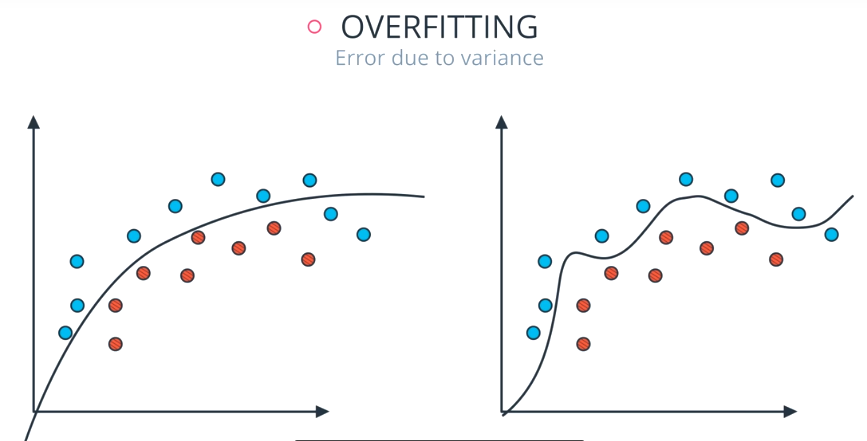


More technically saying:

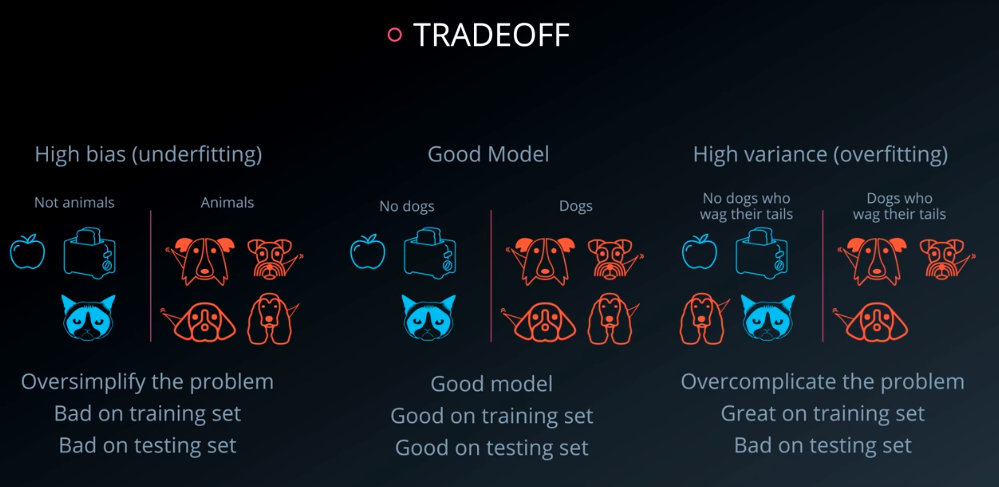








Summary:



* + 1. Model Complexity Graph
    2. Cross Validation
    3. K-Fold Cross Validation
    4. Learning Curves
    5. Detecting Overfitting and Underfitting with Learning Curves
    6. Solution: Detecting Overfitting and Underfitting
  1. Putting it all together
     1. Grid Search
     2. Summary
     3. Outro
  2. Predicting Boston Housing Prices

1. Supervised Learning
2. Unsupervised Learning
3. Machine Learning Capstone