**WELCOME TO MACHINE LEARNING**

Meet with Sebastian and Katie to discuss machine learning.

<http://jefflirion.github.io/udacity/>

Supervised Classification Problem

Um monte de exemplos e você sabe a resposta certa para esses exemplos, mostrando ao carro qual o comportamento correto. Assim como se aprende a dirigir (pela observação). Dê um monte de exemplos e ele vai descobrir por conta o que está acontecendo. No deserto, se você correr muito corre o risco de capotar o carro. Dirigindo lentamente, o carro aprendeu a pegar o comportamento com milhares de quilometros de treino.

Applications: Self-Driving Car (Google).

Álbum com fotos taggeadas e reconhecer alguém na foto (Facebook).

Mostrar preferências musicais e um monte de recursos dessa música (gênero etc), recomendar uma nova música. (Pandora, Netflix)

Unsupervised Classification

Analisar dados bancários em busca de transações estranhas e flagrar essas por fraude

\* Não dá para definir o que é uma transação estranha, não há exemplo do que isso possa significar

Reunir estudantes da Udacity em tipos baseados nos estilos de aprendizado.

\* Clustering, não sabemos quais grupos existem

Features e Labels

soaring

|

| o o o o -> like

|

| x x x x -> don't like

-------------

light relaxed fast

gráficos de dispersão (scatter plots). Machine Learning define a superfície de decisão (decision surface) data -> decision surface

**LESSON 1: NAÏVE BAYES**

Learn about classification, training and testing, and run a Naïve Bayes classifier using scikit-learn.

-- 20 - calculando acurácia do NB --

def NBAccuracy(features\_train, labels\_train, features\_test, labels\_test):

""" compute the accuracy of your Naive Bayes classifier """

### import the sklearn module for GaussianNB

from sklearn.naive\_bayes import GaussianNB

### create classifier

clf = GaussianNB()

### fit the classifier on the training features and labels

clf.fit(features\_train, labels\_train)

### use the trained classifier to predict labels for the test features

pred = clf.predict(features\_test)

### calculate and return the accuracy on the test data

### this is slightly different than the example,

### where we just print the accuracy

### you might need to import an sklearn module

accuracy = clf.score(features\_test, labels\_test)

return accuracy

--------------------- base rule e naive base -------------

Bayes Rule (incorpora uma evidência de teste a uma probabilidade anterior)

Teste de Câncer

P(C) = 0.01 (probabilidade de ter o câncer C é 1%)

90% dos casos dão POSITIVOS se você tem C (sensitivity)

90% dos casos dão NEGATIVOS se você NÃO tem C (specitivity)

Se você fizer o teste e der POSITIVO, qual a % de estar correto?

8,3333% (desenhar diagrama)

Bayes Rule -> prior probabilidade + test evidence = posterior probability

prior P(C) = 0.01 (probabilidade de ter o câncer C é 1%) P(-C) = 0.99 (99%)

P(Pos|C) = 0.9 (90%) sensitivity

P(Neg|-C) = 0.9 (90%) specitivity P(Pos|-C) = 0.1 (10%)

posterior P(C|Pos) (probabilidade Câncer, dado que teste deu positivo) = P(C) \* P(Pos|C)

P(-C|Pos) = P(-C) \* P(Pos|-C)

Text Learning - Naive Bayes (exemplo de E-mail, Chris usa .8 'deal', .1 'love' e .1 life, Sara .5 'love', .2 'deal' e .3 'life')

Quem mandou o e-mail Life Deal?

Priori: P(Chris) = 0.5 X (.1 \* .8 \* .5) = 0.04

P(Sara) = 0.5 (.3 \* .2 \* .5) = 0.03

P(Chris| "Life Deal") = 0.04 / (0.04 + 0.03) = 0.57

P(Sara | "Life Deal") = 0.03 / (0.04 + 0.03) = 0.43

E para "Love Deal"?

P(Chris) = 0.5 X (.1 \* .8 \* .5) = 0.04

P(Sara) = 0.5 (.5 \* .2 \* .5) = 0.05

P(Chris| "Life Deal") = 0.04 / (0.04 + 0.05) = 0.44

P(Sara | "Life Deal") = 0.05 / (0.04 + 0.05) = 0.55

Naive Bayes é chamado de naive porque ignora as ordens das palavras. Pontos fracos e fortes:

- Fácil de implementar

- Google no começo confundiu "Chicago Bulls" com touros de Chicago. A ordem importava.

**LESSON 2: SVM**

Build an intuition about how support vector machines (SVMs) work and implement one using scikit-learn.

maximizar a margem (que separa dois grupos distintos), classifica corretamente e maxima a margem.

Quando não consegue achar a linha, deixa um 'outlier' no grupo. SVMs podem ser não lineares e desenhar formas complexas.

Há grupos que não podem ser classificados, mas se adicionar um novo atributo, pode (|x|, x2+y2) e aí dá para desenhar a decision boundary.

Mas há um truque: uso de kernels. central tricks in all of the machine learning

from sklearn import svm

clf = SVC(kernel="linear")

clf.fit(features\_train, labels\_train)

pred = clf.predict(features\_test)

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape=None, degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

Parameters for SVM: kernel, C and gamma (linear dividiu melhor e RBF deixou ilhas)

C: controls the tradeoff between smooth decision boundary and classifying training points correctly.

The larger C is, less smooth is the decision boundary and more correct points are correctly classified

Gamma: defines how far the influence of a single training example reaches.

Low values -> far (decision boundary more flat)

High values -> close (decision boundary close to overfit, smoother)

Overfitting: common phenomena in machine learning that you have to be aware every time you do machine learning.

One of the ways we can avoid overfitting is through the parameter of the algorithm. All three parameters (C, gamma and kernel)

affect overfitting. So this is a lot of the artistry of machine learning: to tune these parameters. There are some ways to detect overfitting

Pontos fracos e fortes do SVM:

- Forte: funciona muito bem em domínios onde há uma margem clara de separação

- Fraco: não funciona muito bem em datasets muito grandes, porque o tempo de treinamento é cúbico em relação ao tamanho do dataset

- Fraco: também não funcionam bem com muitos ruídos (noise), overlapping classes, e aí onde Naive Bayes entra melhor.

- Fraco: se tiver muitas features e grande dataset, SVMs podem ser lentos e podem ser propensos a overfitting dos ruídos dos dados

**LESSON 3: DECISION TREES**

Learn about how the decision tree algorithm works, including the concepts of entropy and information gain.

To windsurf with sun and wind conditions

sun

| x x x x o o o o o

|x x x x o o oo o o

| x x x x o o o o o

|x x x x o o o oo

| x xx x x o o oo

| x x xx x x x xx x x

| x x x x x x x xx

|x x x x x x x x x x

---------------------- wind

podem ser utilizados tanto para classificação quanto para regressão

no scikit -> Decision Tree Classifier

-- exemplo --

#!/usr/bin/python

""" lecture and example code for decision tree unit """

import sys

from class\_vis import prettyPicture, output\_image

from prep\_terrain\_data import makeTerrainData

import matplotlib.pyplot as plt

import numpy as np

import pylab as pl

from classifyDT import classify

features\_train, labels\_train, features\_test, labels\_test = makeTerrainData()

### the classify() function in classifyDT is where the magic

### happens--fill in this function in the file 'classifyDT.py'!

clf = classify(features\_train, labels\_train)

#### grader code, do not modify below this line

prettyPicture(clf, features\_test, labels\_test)

output\_image("test.png", "png", open("test.png", "rb").read())

--

def classify(features\_train, labels\_train):

### your code goes here--should return a trained decision tree classifer

from sklearn import tree

clf = tree.DecisionTreeClassifier()

clf.fit(features\_train, labels\_train)

return clf

O overfitting acontece quando, por exemplo, min\_split\_sample = 2 (default), gerando muitos nós complexos na árvore (com várias linhas e cortes para pegar apenas 2 pontos)

------------------------------------------------------------------------------

Entropy <- controls how a decision tree decides how to split the data.

Definition: measure of impurity in a bunch of examples. What you try to do when building a DT is you're trying to find variables and split points along those variables that's gonna make subsets that are as pure as possible.

Entropy is basically the opposity of purity.

Ex 1: all examples are in the same class (say all are fraud), so entropy is 0 (cause purity is 100%)

Ex 2: all examples are evenly split between all the available classes, in this case entropy is 1.0, the maximum value

Entropy formula is E[i] -Pi log2 (Pi). Pi = i/total, where i = total of samples of i label and total is the sum of all samples of all labels.

Python: to calculate entropy formula

4 samples, 2 of class A and 2 of class B:

import math

-0.5 \* math.log(0.5, 2) -0.5 \* math.log(0.5, 2) = 1.0, maximum impurity, samples are evenly split.

4 samples, all 4 of class A:

import math

-1 \* math.log(1, 2)

Formula of Entropy: SUM of all P(classes), where P(class) = -P(class)/total \* log(P(class))

"""

Calculating Entropy in a dataset of 2 classes

evenly split, which yields entropy = 1, the max impurity situation

"""

import math

#Pslow 2/4 and Pfast 2/4

pslow = 2.0/4

pfast = 2.0/4

entropy = -(pslow) \* math.log(pslow, 2) -(pfast) \* math.log(pfast, 2)

print "entropy: ", entropy

------------------------------------------------------------------------------

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

Decision Tree Algorithm goal is to MAXIMIZE INFORMATION GAIN

Example: 3 features

[grade ] [bumpiness ] [speed limit] speed(label)

steep bumpy yes SLOW

steep smooth yes SLOW

flat bumpy no FAST

steep smooth no FAST

==== entropy of parent = 1.0.

let's decide which variable we'll use to division.

1 - Starting to calculate information gain on grade (inclinação):

SSFF

/ \

steep 3 flat 1

(SSF) (F) <- entropy 0, as all the observations are of the same class.

/\

|

|

entropy is: -(Pslow) \* log(Pslow) - (Pfast) \* log(Pfast) = -(2.0/3) \* math.log(2.0/3,2) -(1.0/3) \* math.log(1.0/3,2) = 0.9184

entropy(children) = (slow)3/4 \* 0.9184 + (fast)1/4 \* 0 = 0.6888

so, information gain based on grade is 1.0 - 0.6888 = 0.3112

--------------------------------

2 - Calculating information gain on bumpiness (irregularidade do terreno):

SSFF

/ \

bumpy 2 smooth 2

(SF) (SF) <- entropy 1, as all the observations are evenly split.

/\

|

|

entropy is 1 as all the observations are evenly split

entropy(children) = (bumpy)2/4 \* 1 + (fast)2/4 \* 1 = 1.0

so, information gain based on grade is 1.0 - 1.0 = 0.0

A split on bumpiness yields two children (bumpy and smooth) with an equal split of slow and fast classes.

There is zero information gain. This is probably not where we wanna split the sample to build the DT.

--------------------------------

3 - Calculating information gain on speed limit:

SSFF

/ \

yes 2 no 2

(SS) (NN) <- entropy 0, as all the observations are of the same class.

/\

|

|

entropy is 0 as all the observations are of the same class

entropy(children) = (yes)2/4 \* 0 + (no)2/4 \* 0 = 0.0

so, information gain based on grade is 1.0 - 0.0 = 1.0,

because it's very pure (all yes speed limit are labeled as SLOW and all no's are FAST)

definitely it's where we want to make a split for the DT.

--------------------------------

Bias-Variance Dilemma

- a high bias machine learning algorithm is one that practically ignores the data.

It has almost no capacity to learn anything, and it is called a bias.

A bias car would be one that I can train, and no matter which way I train it, it doesn't do anything differently.

The other extreme is it can only replicate stuff it has seen before. That's an extremely high variance algorithm.

The problem with that is it will react very poorly in situations it hasn't seen before because it doesn't have the

right bias to generalize to new stuff. In reality, what you want is something in the middle.

You have what's called a bias-variance trade-off.

You want an algorithm that has some authority to generalize, but is still very open to listen to the data.

You can turn a knob and make it more biased or it can be high variance and

the art of tilting that knob is the art of making machine learning amazing.

--------------------------------

Decision Trees strengths and weaknesses:

- They're prone to over fitting, especially if you have data that has lots of features and a complicated DT it can over fit the data

you have to be careful with the parameter tunes that you're picking when you use the DT to prevent this from happening.

- You can build bigger classifiers out of DT in something called ENSEMBLE METHODS.

--------------------------------

Mini project on DT - tackle (abordar) - try to undertand who wrote an email based on the word content of that email using a DT.

(decision\_tree/\*.py)

Would a large number of features give you a more or less complex DT, all other things being equal?

More complex

**LESSON 4: CHOOSE YOUR OWN ALGORITHM**

In this mini project, you will extend your toolbox of algorithms by choosing your own algorithm to classify terrain data, including k-nearest neighbors, AdaBoost and Random Forests.

* K nearest neighbors: simple, straightforward
* Adaboost: very powerful, usually used with DT (also called boosted decision tree), ensemble methods
* Random Forest: usually used with DT, ensemble methods

\* Ensemble methods: meta classifiers built from (usually) decision trees,

many classifiers working together to come up with a single decision. It's a little bit like how we choose our president by voting.

There's a single decision of who's the president going to be, and there are many different people who have different opinions on what

that answer should be. So what you have to do is you ask the question of many different people and all together you come up with a

single answer.

choose\_your\_own/\*.py

[Process]

1 - do some research, get a general understanding

2 - find sklearn documentation

3 - deploy it! (get your hands dirty)

4 - use it to make predictions

5 - evaluate it. what's the accuracy?

**LESSON 6: DATASETS AND QUESTIONS**

Find out about Enron data set used in the next lessons and mini project.

Enron dataset. https://www.technologyreview.com/s/515801/the-immortal-life-of-the-enron-e-mails/

Types of data:

- numerical: numerical values (numbers) e.g. salary

- categorical: limite number of discrete values (category), class label, if is woman of man

- time series: temporal value (date, timestamp)

- text: words

Open the starter code: datasets\_questions/explore\_enron\_data.py

**LESSON 7: REGRESSIONS**

See how we can model continuous data using linear regression.

Regression is [Continuous] supervised learning.

Before we had discrete output (binary, fast or slow etc) but in many learning problems, our output could be continues as well.

For example, input a height of a person and get the weight as the output probably we'll get a function.

Continuous is about the output. Because input was always continuous in the past examples.

Continuous VS Discrete

Age -> Continuous

Weather (sunny, rainy) -> Discrete

Person who wrote e-mail -> Discrete (there's no order)

Phone number given the person -> Discrete (there's no order)

Income -> Continuous (because 10.000 is almost 9.999 and 20 is almost 19)

Continuous implies there's some sort of ordering to it. Linear order.

For instance, calculating net worth given the age.

Age 0 = 0 net worth

Age 80 = 500 net worth

So: net worth = 6.25 \* age + 0

net worth = TARGET VARIABLE, the one we're trying to predict, often called the OUTPUT.

age = INPUT VARIABLE

6.25 = this fact is called the SLOPE (declive)

0 = this number, which happens in this case to be zero, is called the INTERCEPT (interceptação)

Coding with Katie: http://scikit-learn.org/stable/modules/linear\_model.html

Created a data set of net worths and ages and split it into training and testing sets, just like we do in supervised classification.

regression/studentMain.py and regression/studentRegression.py

python studentMain.py

One way to evaluate regression is by evaluating the metric R squared. There's also the sum of the errors.

The higher the R squared is, the better, with a maximum value of 1. If there's an overfitting going on that'll show up in having a lower score when you look at your testing data.

So we can ask how good is our regression R-squared is by:

print "\nr-squared score:", reg.score(ages\_test, net\_worths\_test)

0.812365729231

If we ask it of our training dataset we'll also get some interesting information, but only by using the test dataset can we be sensitive to thinks like over-fitting.

print "\nr-squared score:", reg.score(ages\_train, net\_worths\_train)

0.874588235822

Exercise: regression/regressionQuiz.py e regression/ages\_net\_worths.py

## Let's talk some more about the types of errors that you can have on regressions and how you can quantify them.

error = actual net worth of a particular person - predicted net worth by our regression line

Example: age = 35, so predicted net worth is 6.25 \* 35 = 218.75. Actual net worth is 200. The error for this point is -18.75.

Another way to think about that visually, is it's this distance between the line and the point.

SSE - best regression is the one that minimizes the sum of squared errors.

SUM of all training points(actual - predicted)^2. y = Mx + b, where M is the slope and b is the intercept. There are several algorithms that

help you to find this: 2 most populars are Ordinary Least Squares (OLS) used in sklearn LinearRegression and Gradient Descent.

Problem with SSE is that the more datapoints you have, higher is SSE value, even though there's a good fit.

An evaluation that doesnt't have this shortcoming is called "R SQUARED" and is a very popular evaluation metric for describing the goodness of fit of a linear regression. And what r squared is, is it's a number that effectively answers the question: HOW MUCH OF MY CHANGE IN OUTPUT(Y) IS EXPLAINED BY THE CHANGE IN MY INPUT (X)?

0.0 < R^2 < 1.0

if the number is very small, that generally means that your regression line isn't doing a good job of capturing the trend in the data.

On the other hand if the r squared is large, close to 1, what that means is your regression line is doing a good job of describing the relationship between your input, x variable, and your output, y variable. And remember that this is the whole point of performing a regression, is to come up with the mathematical formula that describes this relationship. So if your r squared is close to 1, it basically means your fit is doing a good job. The good thing about R SQUARED is that it's INDEPENDENT OF THE NUMBER OF TRAINING POINTS. So it will always be between 0 and 1. In general it's not affected by the number of points in your data set, being a little bit more reliable than a SUM OF SQUARED ERRORS, especially if the number of points in the data set could potentially be CHANGING.

R SQUARED SCORE: regression/studentRegression.py, python studentMain

0.857 is a good r squared. It's possible that there's still could be variables out there, for example, features that if we were able to incorporate the information from additional features we would be better able to predict a person's net worth. So in other words, if we were able to use more than one feature, sometimes we can push up this r squared even further. On the other hand, there are sometimes really complicated problems where it's almost impossible to get an r squared that would be anywhere near this high. So sometimes in Political Science for example they're trying to run a regression that will predict whether a country will go to war.

What data makes a good linear regression?

Data that draws a line.

GOOD (y = mx + 8 where m is 0)

|

|

| xxxxxxxxxx

|

|

------------

BAD (tricky, the problem is there's no variation in X axis. Many Y for 1 x.)

| x

| x

| x

| x

|

------------

BAD (data all over the place, not a good candidate)

| x x x

| x x

| x x

| x x

|

------------

BAD (almost good. but need 2 different lines to fit the pattern adequately)

| x

| x x

| x x

| x x

| x

------------

BAD (y = x^2) curve. not a example of linear

| xxxx

| x x

| x x

|x x

|x x

------------

[Comparing Classification and Regression]

[PROPERTY] [SUPERVISED CLASSIFICATION] [REGRESSION]

output type discrete (class labels) continuous (number)

what are you trying to find? decision boundary (assign class label) "best fit line"

evaluation accuracy "sum of squared error or r squared"

You should think of regression as a different type of supervised learning, not as a completely new topic that you now have to learn from scratch.

[MultiVariate Regression]

More than one feature to predict target. Example:

AGE and IQ -> Networth.

AGE = X1, IQ = X2 and NETWORTH = Y

X2 = IQ |

|

120 | 200 300 400

|

100 | 150 250 350

|

80 | 100 200 300

|

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

20 40 60 X1 = AGE

Y = a\*X1 + b\*X2 + c

Y = 5\*X1 + 2.5\*X2 - 200

x1 goes by 20, y goes by 100. So 5 times.

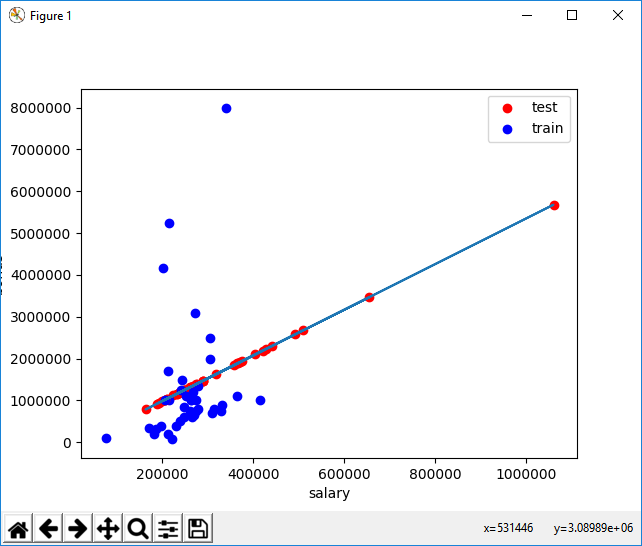
x2 goes by 20, y goes by 50. So 2.5 times

[Regression Mini-Project Video]

Predict bonus from someone's salary and stock options, are you able to predict how much money they're making in their bonus every year. And the answer is, well, maybe, but you have to use a regression to find out. Second part of the project is about outliers. Outliers are data points that fall FAR outside the pattern as a large and we'll get the idea of how those actually affect the result that you get on something like your regression. It can be bigger than you might think.

In this project, you will use regression to predict financial data for Enron employees and associates. Once you know some financial data about an employee, like their salary, what would you predict for the size of their bonus?

Get the data from here "regression/finance\_regression.py"



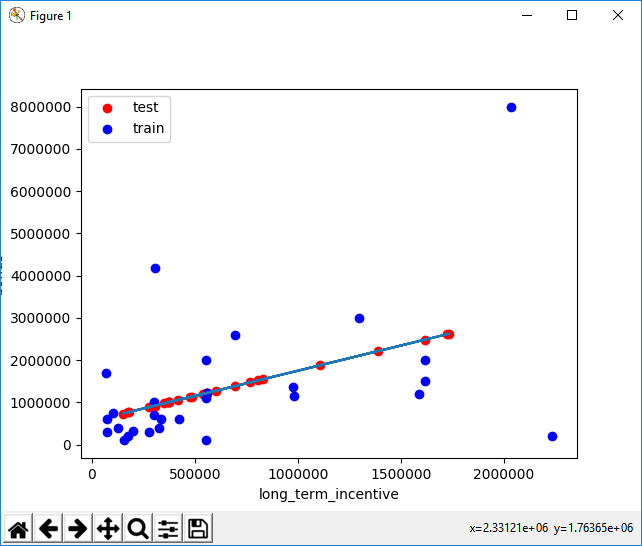
features\_list = ["bonus", "salary"]

SLOPE: 5.44814028881

INTERCEPT: -102360.543294

SCORE TEST: -1.48499241737

SCORE TRAIN: 0.0455091926995



features\_list = ["bonus", "long\_term\_incentive"]

SLOPE: 1.19214698985

INTERCEPT: 554478.756215

SCORE TEST: -0.59271289995

SCORE TRAIN: 0.217085971258

Is salary or long-term incentive a better feature for predicting someone’s bonus?

That's right! We have a better score when using long-term incentive to predict someone's bonus, which translates to a better fit.

**Sneak Peek: Outliers Break Regressions**

This is a sneak peek of the next lesson, on outlier identification and removal. **Go back to a setup where you are using the salary to predict the bonus, and rerun the code to remind yourself of what the data look like.** You might notice a few data points that fall outside the main trend, someone who gets a high salary (over a million dollars!) but a relatively small bonus. This is an example of an outlier, and we’ll spend lots of time on them in the next lesson.

A point like this can have a big effect on a regression: if it falls in the training set, it can have a significant effect on the slope/intercept if it falls in the test set, it can make the score much lower than it would otherwise be As things stand right now, this point falls into the test set (and probably hurting the score on our test data as a result). Let’s add a little hack to see what happens if it falls in the training set instead. Add these two lines near the bottom of *finance\_regression.py*, right before *plt.xlabel(features\_list[1])*:

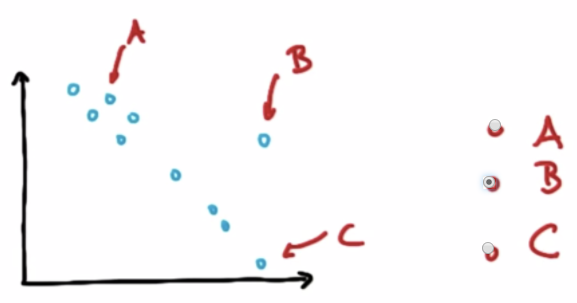
*reg.fit(feature\_test, target\_test)  
plt.plot(feature\_train, reg.predict(feature\_train), color="b")*  
  
Now we’ll be drawing two regression lines, one fit on the test data (with outlier) and one fit on the training data (no outlier). Look at the plot now--big difference, huh? That single outlier is driving most of the difference. **What’s the slope of the new regression line?**

(That’s a big difference, and it’s mostly driven by the outliers. The next lesson will dig into outliers in more detail so you have tools to detect and deal with them.)

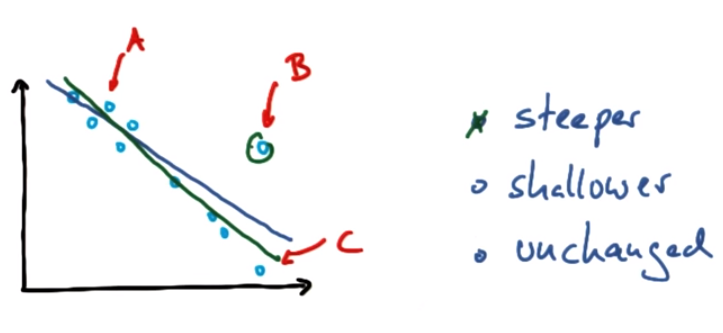
**LESSON 8: OUTLIERS**

Sebastian discusses outlier detection and removal.

Residual error of datapoint (maior distância) do ponto até a linha de regressão.



After removing the outlier point from the data (outlier rejection), the regression line would become steeper (green line).





**Outliers mini-project**

This project has two parts. In the first part, you will run a regression, and identify and remove the 10% of points that have the largest residual errors. Then you’ll remove those outliers from the dataset and refit the regression, just like the strategy that Sebastian suggested in the lesson videos.

In the second part, you will get acquainted with some of the outliers in the Enron finance data, and learn if/how to remove them.

<https://github.com/udacity/ud120-projects/tree/master/outliers>

**Slope of Regression with Outliers**

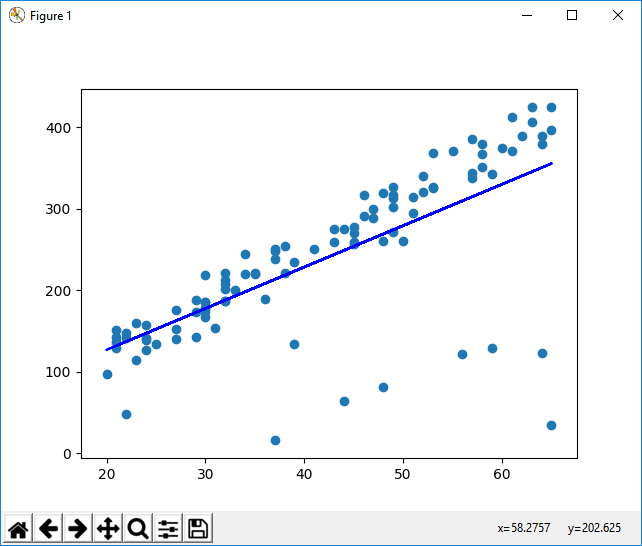
Sebastian described to us an algorithm for improving a regression, which you will implement in this project. You will work through it in the next few quizzes. To summarize, what you'll do is fit the regression on all training points discard the 10% of points that have the largest errors between the actual y values, and the regression-predicted y values refit on the remaining points.

Start by running the starter code (*outliers/outlier\_removal\_regression.py*) and visualizing the points. A few outliers should clearly pop out. Deploy a linear regression, where net worth is the target and the feature being used to predict it is a person’s age (remember to train on the training data!).

The “correct” slope for the main body of data points is 6.25 (we know this because we used this value to generate the data);

**what slope does your regression have?**

**5.07793064**

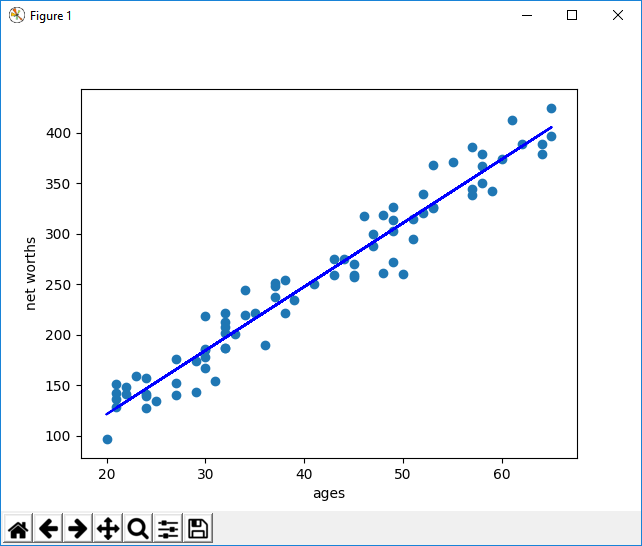


**What is the score you get when using your regression to make predictions with the test data?**

0.878262470366

**Slope After Cleaning**

In ***outliers/outlier\_cleaner.py***, you will find the skeleton for a function called *outlierCleaner()* that you will fill in with a cleaning algorithm. It takes three arguments: *predictions* is a list of predicted targets that come from your regression, *ages* is the list of ages in the training set, and *net\_worths*is the actual value of the net worths in the training set. There should be 90 elements in each of these lists (because the training set has 90 points in it). Your job is to return a list called cleaned\_data that has only 81 elements in it, which are the 81 training points where the predictions and the actual values (net\_worths) have the smallest errors (90 \* 0.9 = 81). The format of cleaned\_data should be a list of tuples, where each tuple has the form (age, net\_worth, error).   
  
Once this cleaning function is working, you should see the regression result changes. **What is the new slope?** Is it closer to the “correct” result of 6.25?



# Quiz: Slope After Cleaning

NOTE: In outliers/outlier\_removal\_regression.py, in the section where outlier cleaning is performed (starts with the comment ### identify and remove the most outlier-y points), make sure that the input argument to reg.predict is ages\_train and not ages so that you are cleaning based on just the training data. The arguments to the cleaner should also be based off of the \*\_train variables.

==== After refitting data ====

Coeff: [[ 6.32006691]]

Intercept: [-5.32173018]

Score Train: 0.413564421515

Score Test: 0.982470051004

**Enron Outliers**

In the mini-project for the regressions lesson, you used a regression to predict the bonuses for Enron employees. As you saw, even a single outlier can make a big difference on the regression result. There was something we didn’t tell you, though, which was that the dataset we had you use in that project had already been cleaned of some significant outliers. Identifying and cleaning away outliers is something you should always think about when looking at a dataset for the first time, and now you’ll get some hands-on experience with the Enron data.

You can find the starter code in ***outliers/enron\_outliers.py***, which reads in the data (in dictionary form) and converts it into a sklearn-ready numpy array. Since there are two features being extracted from the dictionary (“salary” and “bonus”), the resulting numpy array will be of dimension N x 2, where N is the number of data points and 2 is the number of features. This is perfect input for a scatterplot; we’ll use the **matplotlib.pyplot** module to make that plot. (We’ve been using pyplot for all the visualizations in this course.) Add these lines to the bottom of the script to make your scatterplot:

for point in data:

salary = point[0]

bonus = point[1]

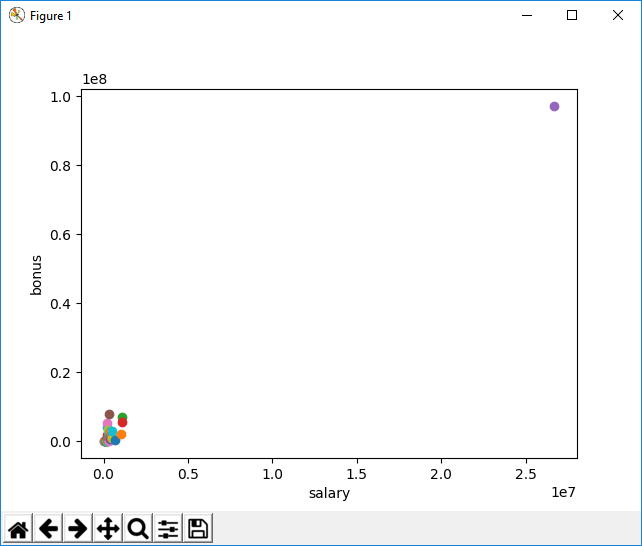
matplotlib.pyplot.scatter( salary, bonus )

matplotlib.pyplot.xlabel("salary")

matplotlib.pyplot.ylabel("bonus")

matplotlib.pyplot.show()

As you can see, visualization is one of the most powerful tools for finding outliers!



**Identify the Biggest Enron Outlier**

There’s one outlier that should pop out to you immediately. Now the question is to identify the source. We found the original data source to be very helpful for this identification; you can find that PDF in ***final\_project/enron61702insiderpay.pdf***   
**What’s the name of the dictionary key of this data point?** (e.g. if this is Ken Lay, the answer would be “LAY KENNETH L”).

TOTAL is the biggest outlier. After removing it:

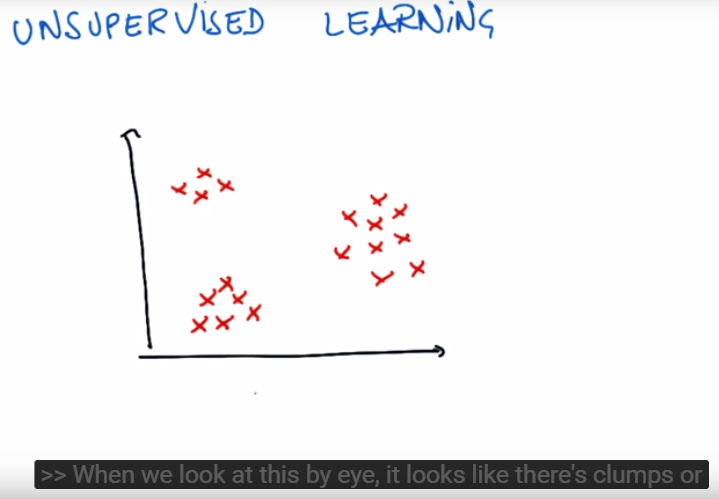


**Four more outliers. Should these be cleaned away, or left in as a data point?**

**Leave them in:** Yes! They're two of Enron's biggest bosses, and definitely people of interest.

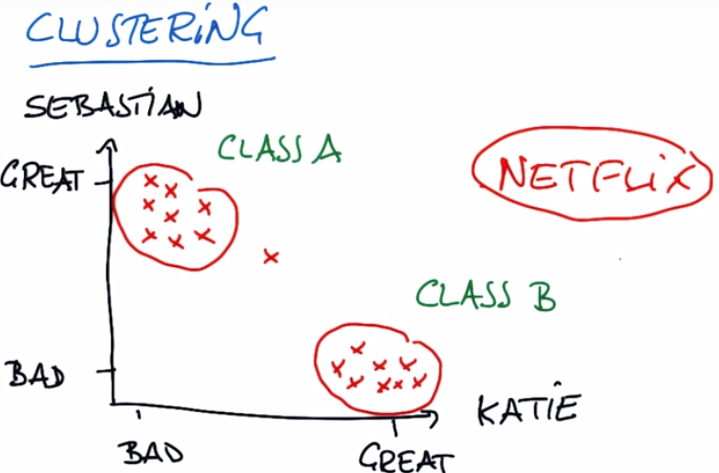
**LESSON 9: CLUSTERING**

Learn about what unsupervised learning is and find out how to use scikit-learn’s k-means algorithm.

****

****

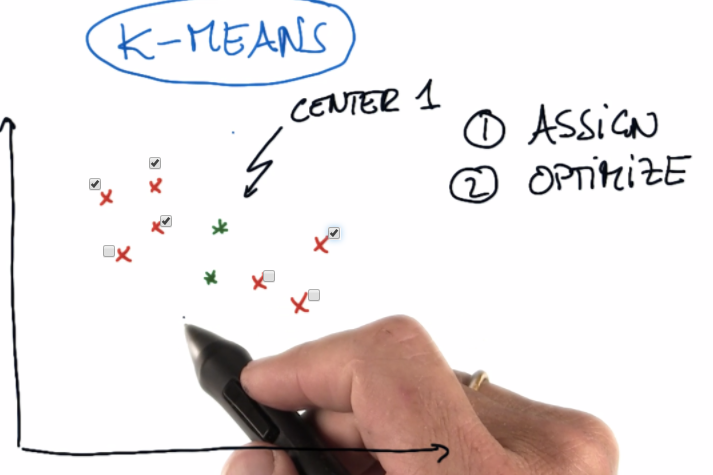
**Clustering Movies**

****

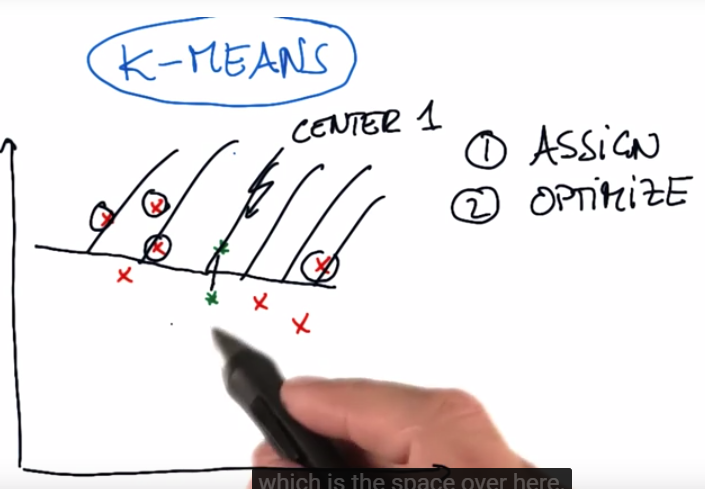
**HOW MANY CLUSTERS?**

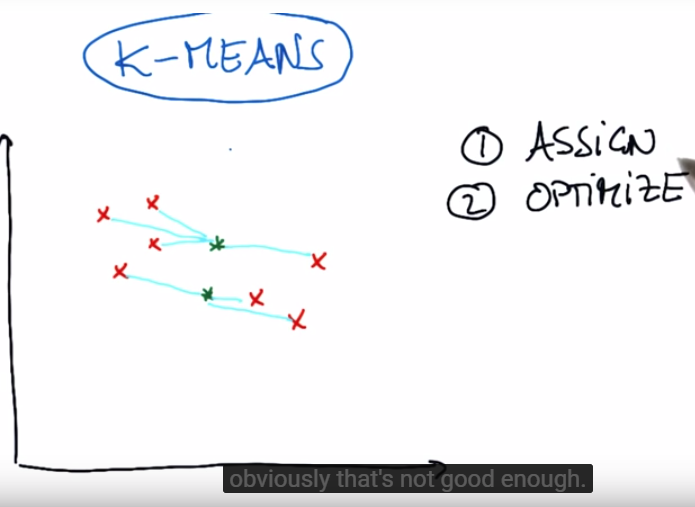
k-means is by far the most basic algorithm used for clustering.

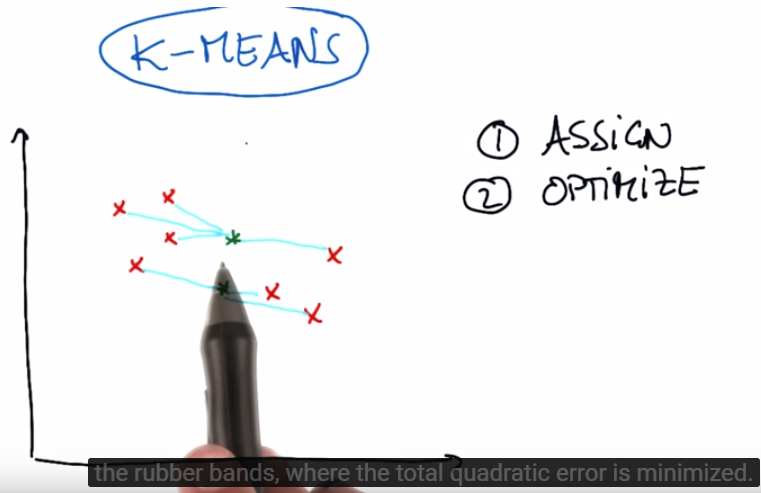


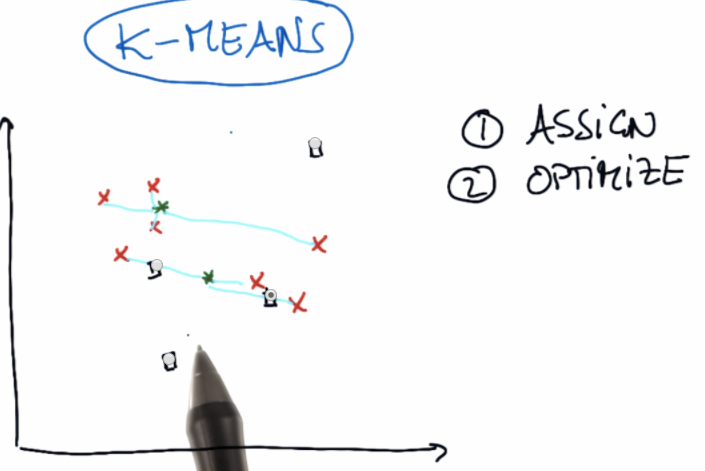


Points closer to center one than center two.

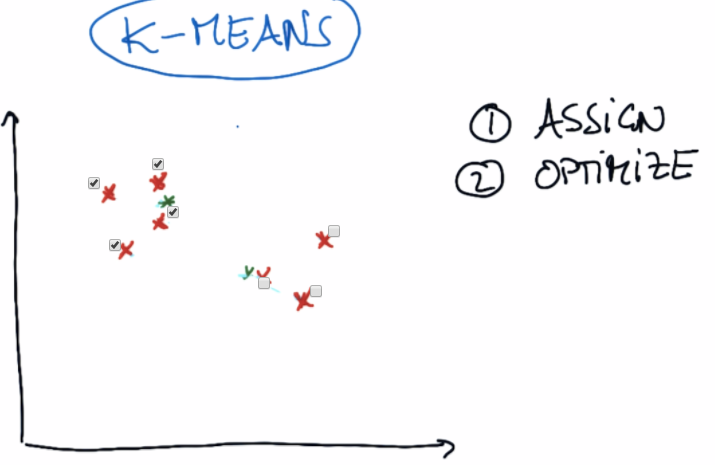


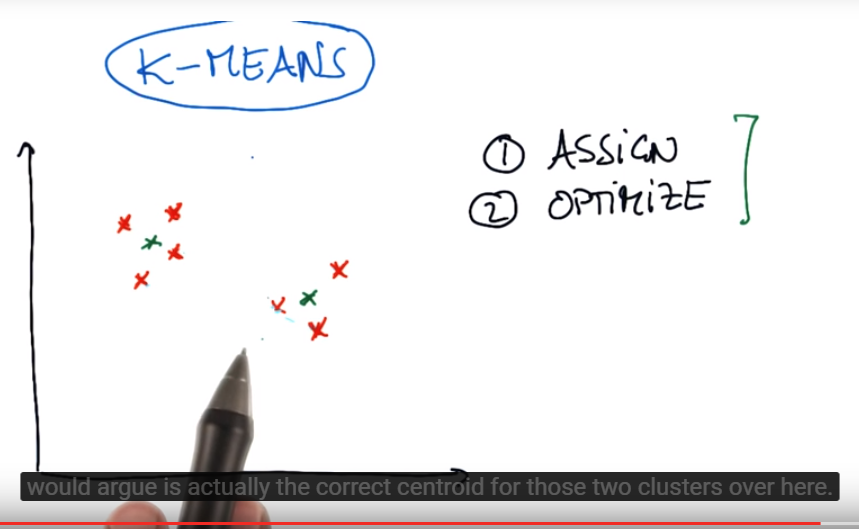
Assignment step, obviously not good enough once the start points were randomly put. To optimize, you’re minimizing the total quadratic square distance of our cluster center to the points.





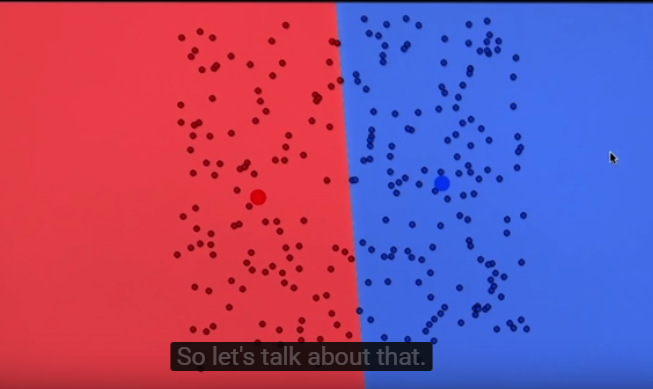
Yep, that center location does minimize the energy of the rubber bands. Now, loop again, assign points to the cluster center.





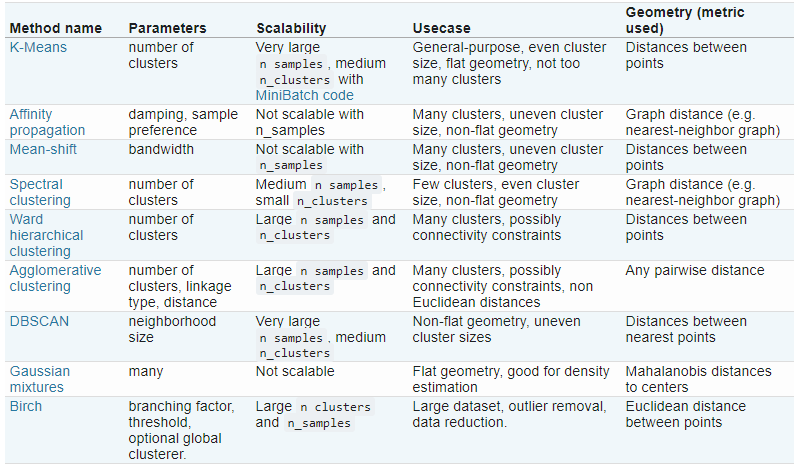
And this is the k-means algorithm.

K-means cluster visualization link: <https://www.naftaliharris.com/blog/visualizing-k-means-clustering/>



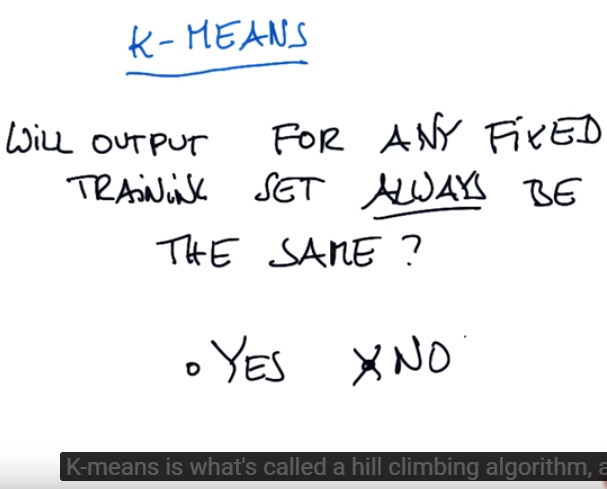
Depending on the initial centroid positions, we could have a vertical line or horizontal splitting the two clusters, you can get clustering in the end that looks totally different. This might seem like a big problem, but there is one pretty powerful way to solve it.

Scikit-learn implementation of clustering: <http://scikit-learn.org/stable/modules/clustering.html>. There are many types of clustering besides just k-means clustering.

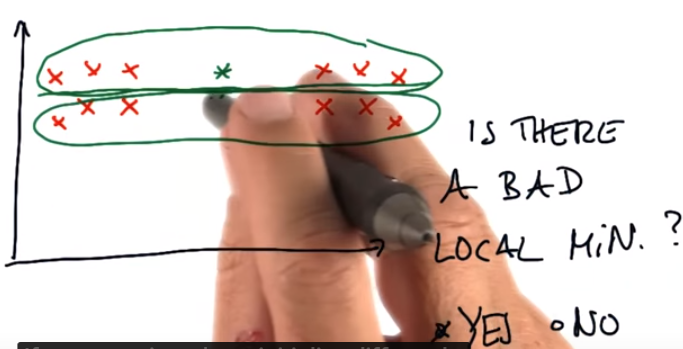


K-means: <http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans> important parameters are:

* **N\_clusters**: default value is eight. But you have to set based on what you think makes sense.
* **Max\_iter**: default value is 300. Iterations that assign points to the centroid and move the centroid.
* **N\_inits**: number of different initializations that you give it. We said that k-means clustering has this challenge, that depending on exactly what the initial conditions are, you can sometimes end up with different clusterings. And so then you want to repeat the algorithm several times so that any of those clusterings might be wrong, but in general the ensemble of all the clusterings will give you something that makes sense, that’s what this parameter controls. It’s basically how many times does it initialize the algorithm, how many times does it come up with clusters. By default it goes through ten times.







**K-Means Clustering Mini-Project**

In this project, we’ll apply k-means clustering to our Enron financial data. Our final goal, of course, is to identify persons of interest; since we have labeled data, this is not a question that particularly calls for an unsupervised approach like k-means clustering.

Nonetheless, you’ll get some hands-on practice with k-means in this project, and play around with feature scaling, which will give you a sneak preview of the next lesson’s material.

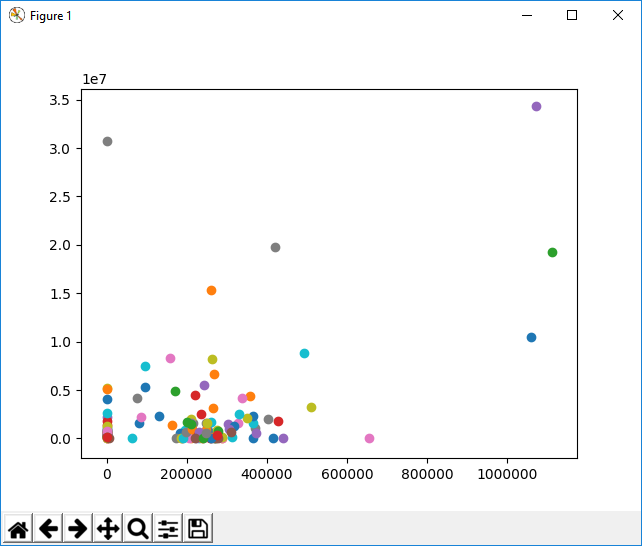
The Enron dataset can be found [**here**](https://github.com/udacity/ud120-projects).

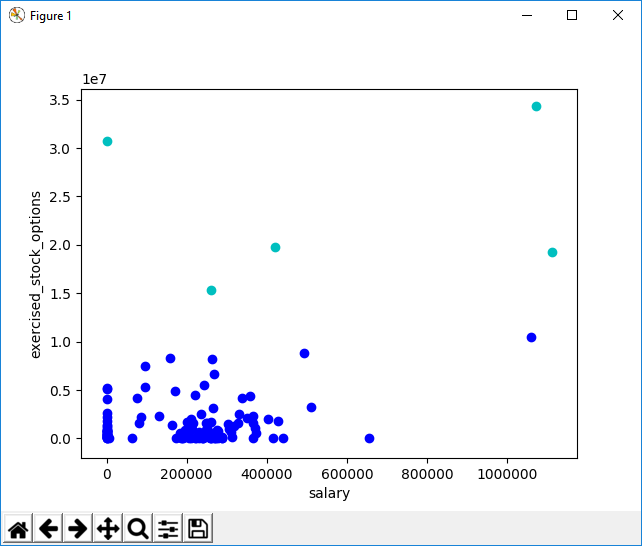
**Clustering Features**

The starter code can be found in **k\_means/k\_means\_cluster.py**, which reads in the email + financial (E+F) dataset and gets us ready for clustering. You’ll start with performing k-means based on just two financial features--take a look at the code, and **determine which features the code uses for clustering**.

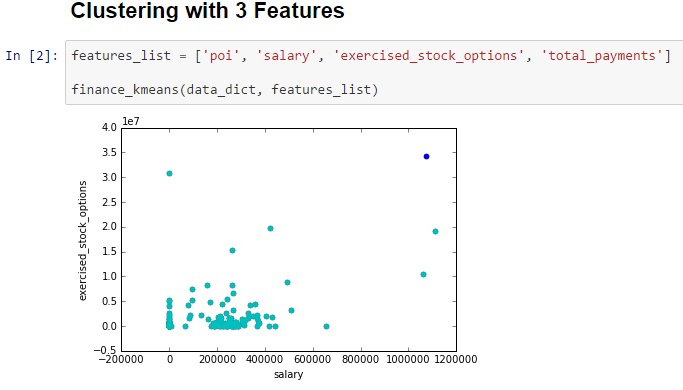
Run the code, which will create a scatterplot of the data. Think a little bit about what clusters you would expect to arise if 2 clusters are created.

Deploy k-means clustering on the financial\_features data, with 2 clusters specified as a parameter. Store your cluster predictions to a list called pred, so that the *Draw()*command at the bottom of the script works properly. **In the scatterplot that pops up, are the clusters what you expected?**





**Clustering with 3 Features**

****

Add a third feature to features\_list, “total\_payments". Now rerun clustering, using 3 input features instead of 2 (obviously we can still only visualize the original 2 dimensions). Compare the plot with the clusterings to the one you obtained with 2 input features. **Do any points switch clusters? How many?**This new clustering, using 3 features, couldn’t have been guessed by eye--it was the k-means algorithm that identified it.

(You'll need to change the code that makes the scatterplot to accommodate 3 features instead of 2, see the comments in the starter code for instructions on how to do this.)

<https://jefflirion.github.io/udacity/Intro_to_Machine_Learning/Lesson8.html>

**Stock Option Range**

In the next lesson, we’ll talk about feature scaling. It’s a type of feature preprocessing that you should perform before some classification and regression tasks. Here’s a sneak preview that should call your attention to the general outline of what feature scaling does.

**What are the maximum and minimum values taken by the “exercised\_stock\_options” feature used in this example?**

(NB: if you look at finance\_features, there are some "NaN" values that have been cleaned away and replaced with zeroes--so while those might look like the minima, it's a bit deceptive because they're more like points for which we don't have information, and just have to put in a number. So for this question, go back to data\_dict and look for the maximum and minimum numbers that show up there, ignoring all the "NaN" entries.)

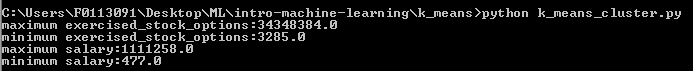
**import** **pandas** **as** **pd**

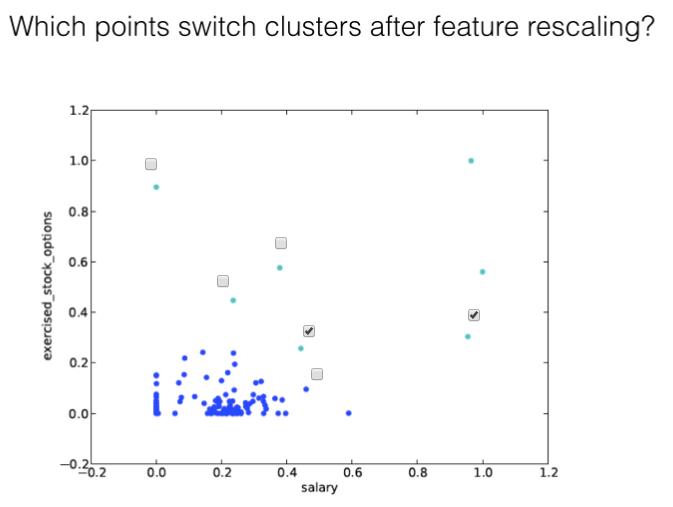
df = pd.DataFrame(data\_dict)

df.loc['exercised\_stock\_options',:] = pd.to\_numeric(df.loc['exercised\_stock\_options',:], errors='coerce')

**print** df.loc['exercised\_stock\_options',:].max(skipna=True)

**print** df.loc['exercised\_stock\_options',:].min(skipna=True)



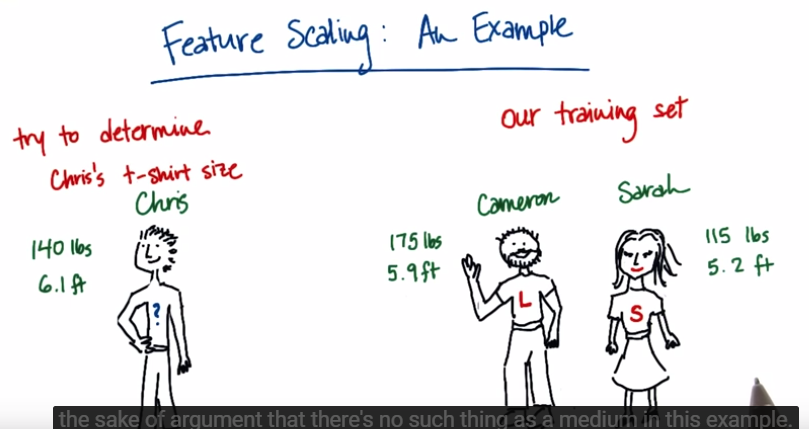
****

That's right, these two points changed clusters when we rescaled the features. We'll talk much more in the next lesson about what just happened, and why.

**LESSON 10: FEATURE SCALING**

Learn about feature rescaling and find out which algorithms require feature rescaling before use.

Feature Scaling is an important step in pre-processing your features for some types of machine learning algorithms.



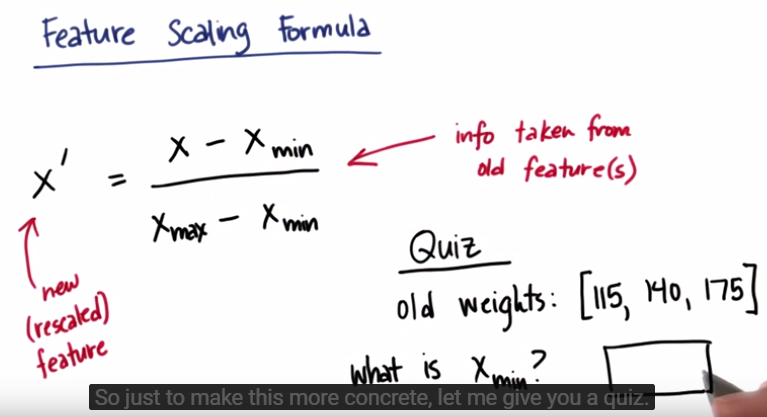
So let’s suppose I have a buddy Chris and Chris wants to know what size T-shirt he should wear. For some reason he doesn’t know his T-shirt size but he does know two good friends Cameron and Sarah and they each know their t-shirt sizes. Chris can then get some data from Cameron and Sarah about say how tall they are and how much they weigh. And based on this information Chris can tell what size t-shirt he should be wearing based on whether he is closer in size to Cameron or to Sarah. So here are some numbers to make this more concrete. Chris is tall and very skinny. He weighs 140 pounds and he’s 6.1 feet tall. Cameron is also fairly tall, but not quite so skinny. He clocks in at a 175 pounds and 5.9 feet. And Sarah’s pretty petite. She’s 115 pounds and 5.2 feet tall. Cameron of course wears a size large T-shirt, Sarah wears a size small, and we’ll assume for the sake of argument that there’s no such thing as a medium in this example. So my first question for you is just your intuition. Should Chris wear a large like Cameron or a small like Sarah, based on his weight and height? **Large.**

Now let’s suppose you’re a computer who’s trying to answer this question. And you don’t have any particular knowledge about what height and weight are. But you can compute numbers pretty easily. And the measure that you’ll use is the height plus weight. Whoever Chris is closer to in the height plus weight metric, that’s the size that he’ll wear. So the first question is, what is the height plus weight for Chris. Height should be measure in feet and weight in pounds. **146.1**. Cameron is **180.9** and Sarah **120.2**. And now here’s the most important question of all. Who is Chris closer to in height plus weight? **Sarah’s.** And the answer, very counterintuitively, is that by this metric of height plus weight, he should be wearing a small like Sarah. What went wrong here?

What went wrong here is that this metric of height plus weight has two very imbalanced features in it, height and weight. So here’s what I mean by that, the height is going to be a number that generally goes between let’s say, the number of five and seven. The weight, on the other hand, takes on much larger values. Between 115 and 175 pounds in this example. So what ends up happening when you compute the sum of the two of them, is that the weight almost always will completely dominate the answer that you get. And height ends up being effectively a rounding error. Whereas what you probably want is something where the two features are equally weighted in, in the sum when you add them together. **And this is what feature scaling does. It’s a method for re-scaling features like these ones, so that they always span comparable ranges, usually between 0 and 1. So then, the numbers you get from height will be between 0 and 1, they’ll still contain the same information. But just expressed in different units. And the weight will also be expressed between 0 and 1. When you add them together, weight won’t completely dominate the equation anymore.**

**Feature Scaling Formula**

There’s a formula and you just have to apply it to your data. X prime is the new feature that you’re making. And you use information about the old features to figure out what value it should take.

****

****

**Min/Max Scaler in sklearn**

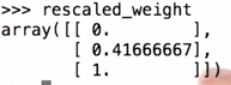
**import** **numpy** **as** **np**

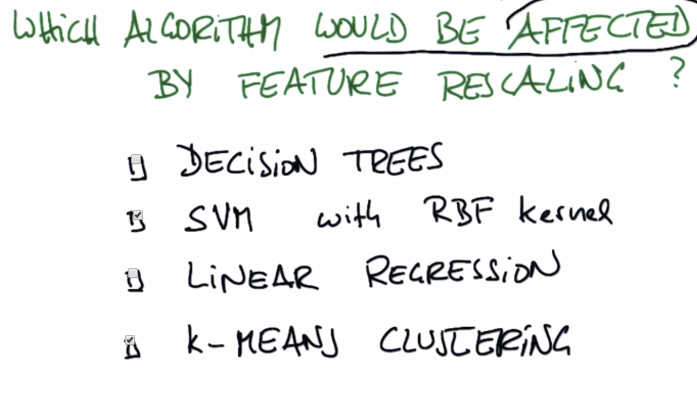
**from** **sklearn.preprocessing** **import** MinMaxScaler

weights = np.array([[115.], [140.], [175.]])

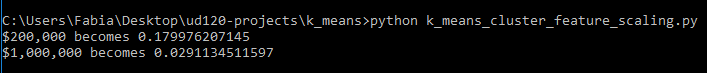
scaler = MinMaxScaler()

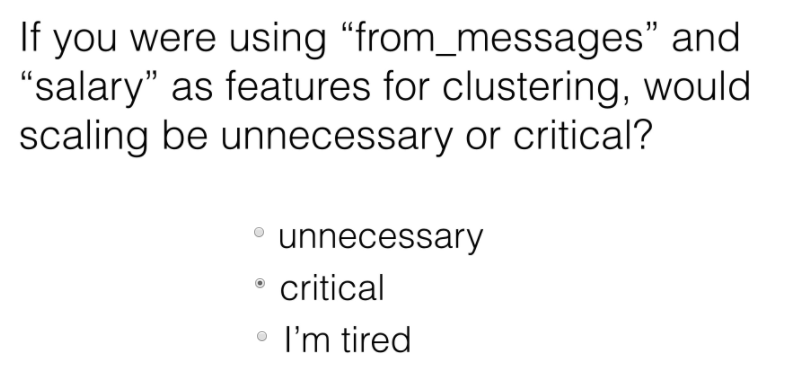
rescaled\_weight = scaler.fit\_transform(weights)

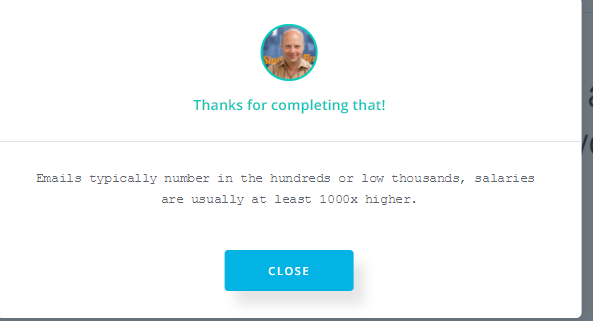
****

****

Apply feature scaling to your k-means clustering code from the last lesson, on the “salary” and “exercised\_stock\_options” features (use only these two features). **What would be the rescaled value of a "salary" feature that had an original value of $200,000, and an "exercised\_stock\_options" feature of $1 million?** (Be sure to represent these numbers as floats, not integers!)

****

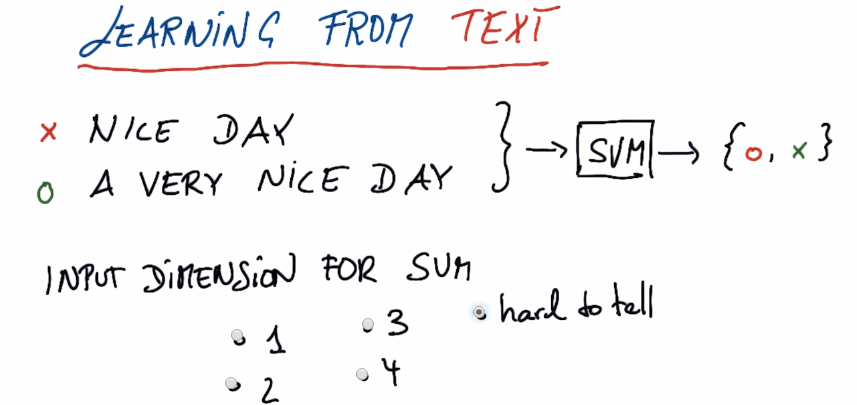
****

****

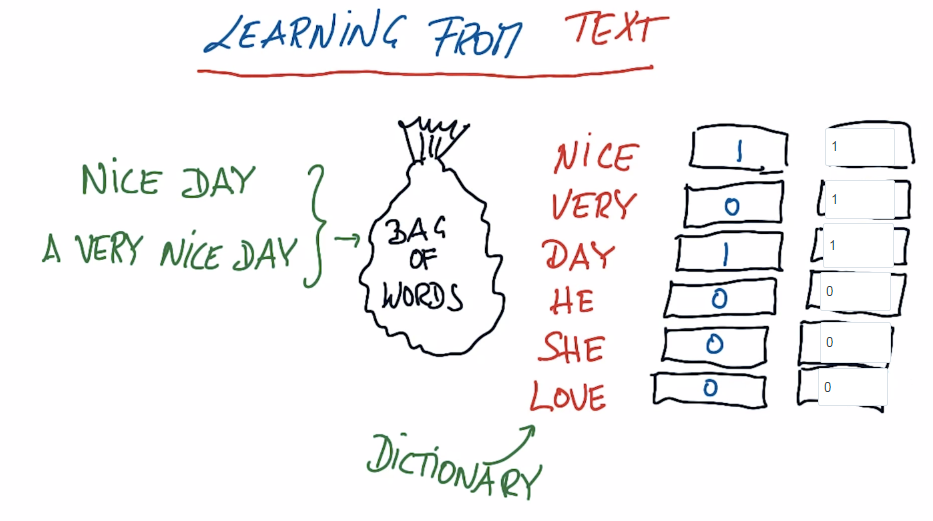
**LESSON 11: TEXT LEARNING**

Find out how to use text data in your machine learning algorithm

The fundamental question in learning from text has to do with what’s the input feature. Suppose you have two strings, two kind of sentences. One is called **nice day** and the other is called **a very nice day**. Suppose for whatever reason, one is a positive example and one of the negative example is indicated by the X over here and this circle. And you want to toss them into your favorite machine learning algorithm like the support vector machines. What do you think is the best input dimension for the SVM?

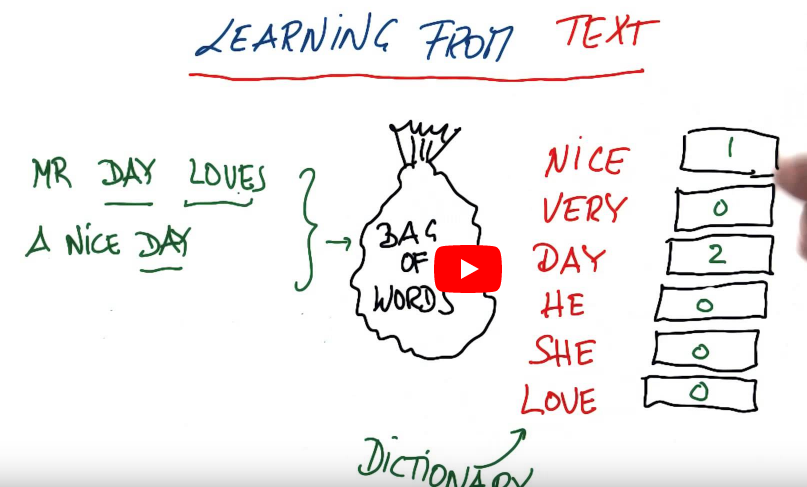
So learning in from text, one of the fundamental problems we have is the length of each document that you learn from, each email or book title, is non-standardized. So you can’t just use each individual word as an input feature, because then long emails would require different input space then short emails. Instead, one of the coolest things in machine learning from text, which analyze all these approaches, is called **bag of words**. And the basic idea is to take any text and count the frequency of words. So the trick is to impose a dictionary. So you have all the words you know. Obviously, they include words like nice, very, day. It might include words that don’t occur on the left side, like he, she, love. So say these are the words you care about. Notice that there’s one on the left side that doesn’t occur on the right side. 

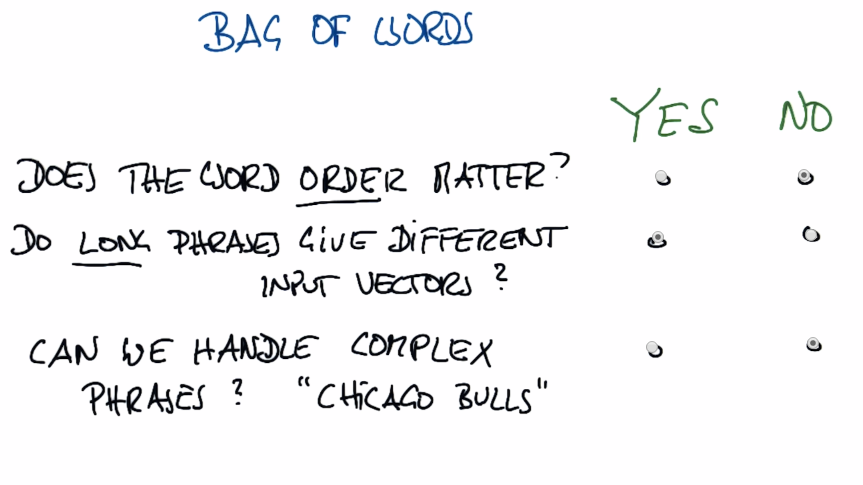
Then you can map each phrase, each email, each title, each text, into a frequency count defined over these six features over here. For nice day phrase:



“A” is often called a stop word, it tends to be removed from text learning, such as the, of, and other stop words. But the key words here are nice, very, day and you can see how these two feature vectors of the same dimension, it estimated six, but they can be applied to different text.

The phrase **Mr Day loves a nice day:**

****

****

The key thing to know about bag of words is that the word order does not matter. It’s clearly a NO, because we are counting words we have no preference to where in the phrase a word occurs, which kind of make it somewhat limited compared to really understanding a text. It’s kind of taking all the words and resorting them in random order. That’s why it’s called bag of words, because you toss them into a bag. Long phrases do give different vectors. So suppose we take the same email and duplicate it. All the counts go up by a factor of two, and it’s going to be a different input vector. So to some extent you’re biasing yourself to also encode the length of your text, not just the frequency of the words, so the answer would be clearly yes. And I would say, can we handle complex phrases? Right now we can’t, unless we make extra buckets for those, which makes it a bit more difficult to preprocess.

Chicago Bulls would count one for Chicago and one for bulls. When Google first came out as a search engine and you typed Chicago Bulls, you saw lots of references to cities and lots of references to animals. And so a bunch of smart Google engineers to change the dictionary to also include combined phrases like Chicago Bulls.

**Bag of words in Sklearn**

In sklearn, Bag of words is called **CountVectorizer**, presumably because it’s counting the number of times various words show up in a corpus. CountVectorizer lives in the feature\_extraction.text part of sklearn.

**from sklearn.feature\_extraction.text import CountVectorizer**

**vectorizer = CountVectorizer()**

**string1 = “hi Katie the self driving car will be late Best Sebastian”**

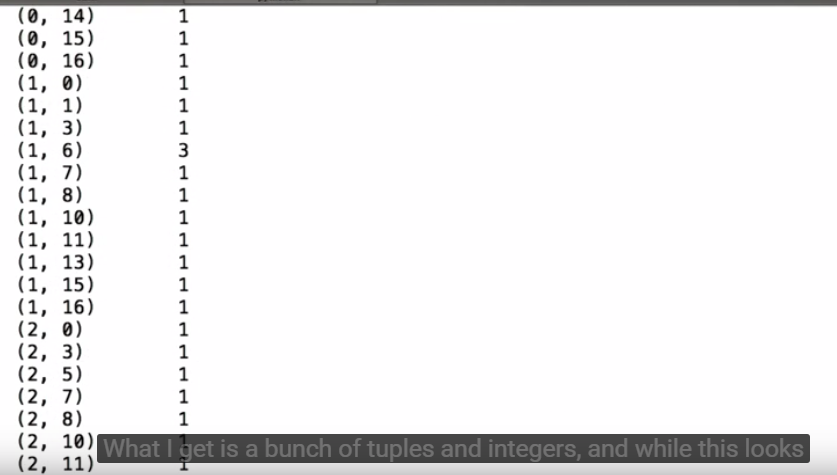
**string2 = “Hi Sebastian the machine learning class will be great great great Best Katie”**

**string3 = “Hi Katie the machine learning class will be most excellent”**

**email\_list = [string1, string2, string3]**

**bag\_of\_words = vectorizer.fit(email\_list)**

**bag\_of\_words = vectorizer.transform(email\_list)**

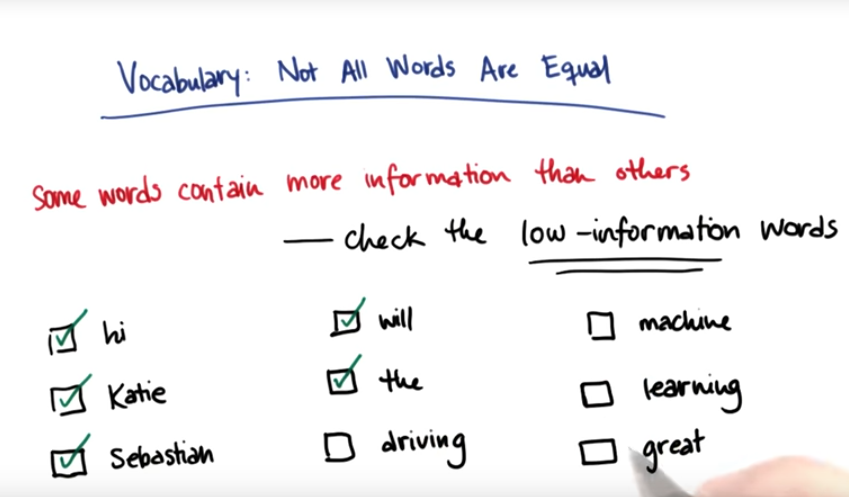


(1, 7) 1

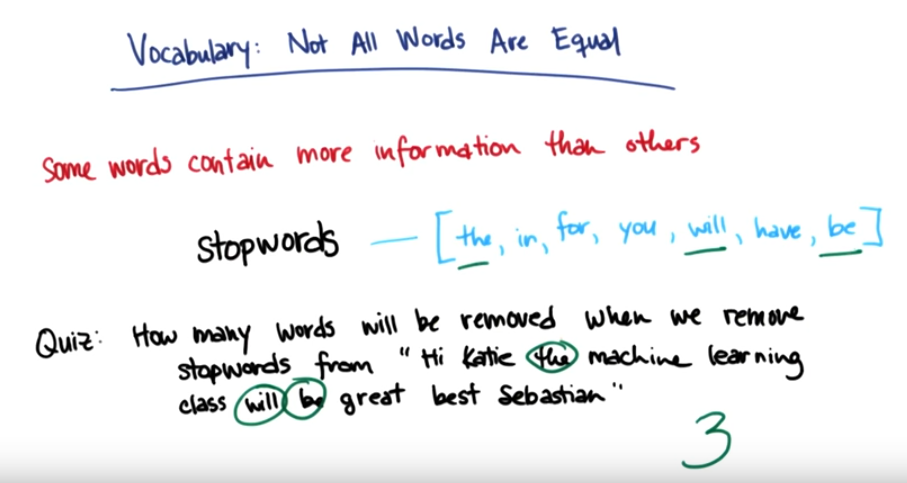
Document number 1, word number 7 occurs 1 time. Document number 1 is string2, because index starts at 0, so:

**print vectorizer.vocabulary\_.get(“great”)** returns 6.

When it comes to vocabulary, not all words are equal. Another way to think about this, is that some words contain more information than others. Check the words that are low information. You don’t technically know what this means yet.



Some words just usually don’t contain a lot of information. And it can be really valuable to have an eye out for these words and to be able to sort of remove them from your corpus, so you don’t have to consider them, you don’t allow them to become noise in your dataset. In general, this list of words is called stopwords. The exact definition of what a stopword is can vary, but in general, it’s a low information word that occurs very frequently. Some examples might include words like and, the, I, you and have. And a very common pre-processing step in text analysis, is to remove the stopwords before you do anything else with the data. Suppose that our body of stopwords is the, in, for, you, will, have and be.



Now that you know about stopwords, I’m going to show how to get a list of stop words out of Python package called NLTK, which stands for Natural Language Tool Kit.

**from nltk.corpus import stopwords**

**sw = stopwords.words(“english”)**

**\*\*\* import nltk**

**\*\*\* nltk.download()**

**sw[0] 🡪 ‘I’**

**sw[10] 🡪 ‘yours’**

**LESSON 12: FEATURE SELECTION**

Katie discusses when and why to use feature selection, and provides some methods for doing this.

**LESSON 13: PCA (Principal Component Analysis)**

Learn about data dimensionality and reducing the number of dimensions with principal component analysis (PCA).

**LESSON 14: Validation**

Learn more about testing, training, cross validation, and parameter grid searches in this lesson.

**LESSON 15: EVALUATION METRICS**

How do we know if our classifier is performing well? Katie discusses different evaluation metrics for classifiers in this lesson.

**LESSON 16: TYING IT ALL TOGETHER**

Spend some time reflecting on the course material with Sebastian and Katie.

**LESSON 17: FINAL PROJECT**