\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Welcome to 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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Naive Bayes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- 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.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Segundo Algoritmo (SVM Supported Vector Machines)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Terceiro e último algoritmo de supervised Learning (Decision Trees)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Fourth algorithm (by my own choice)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

K nearest neighbors: simple, straightforward

Ataboost: 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?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Datasets and Questions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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.

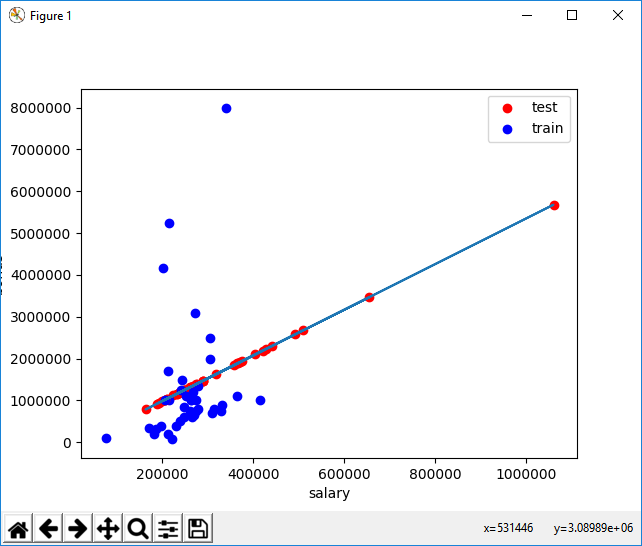
x2 goes by 20, y goes by 50. So 2.5 times

[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"



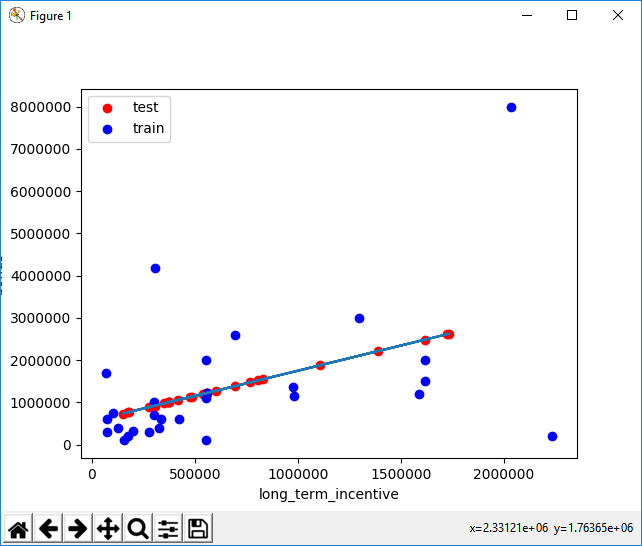
features\_list = ["bonus", "salary"]

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.

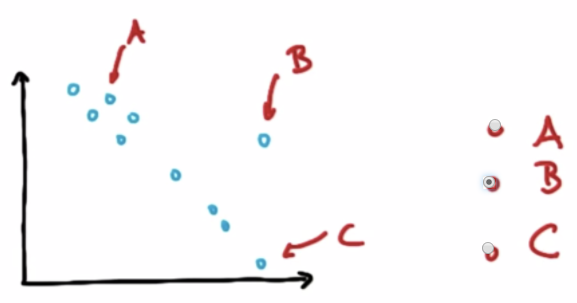
A point like this can have a big effect on a regression: if it falls in the training set, it can have a significant effect on the slope/intercept if it falls in the test set, it can make the score much lower than it would otherwise be As things stand right now, this point falls into the test set (and 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?**

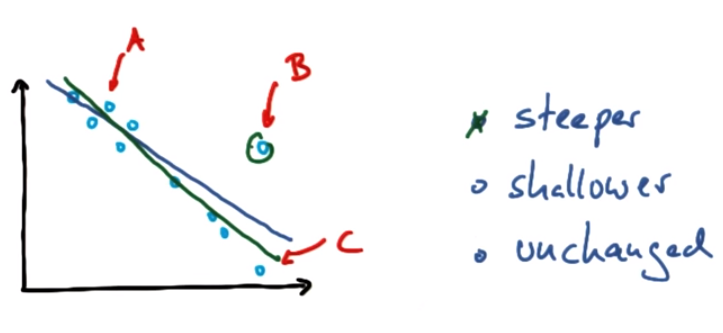
(That’s a big difference, and it’s mostly driven by the outliers. The next lesson will dig into outliers in more detail so you have tools to detect and deal with them.)

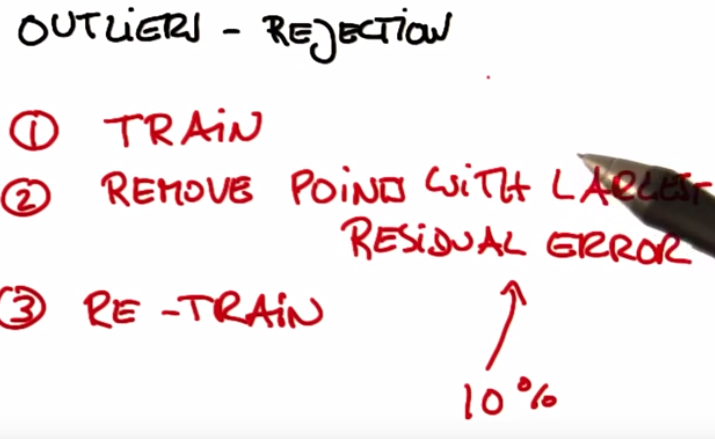
**Capítulo 8 – Outliers**

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?**