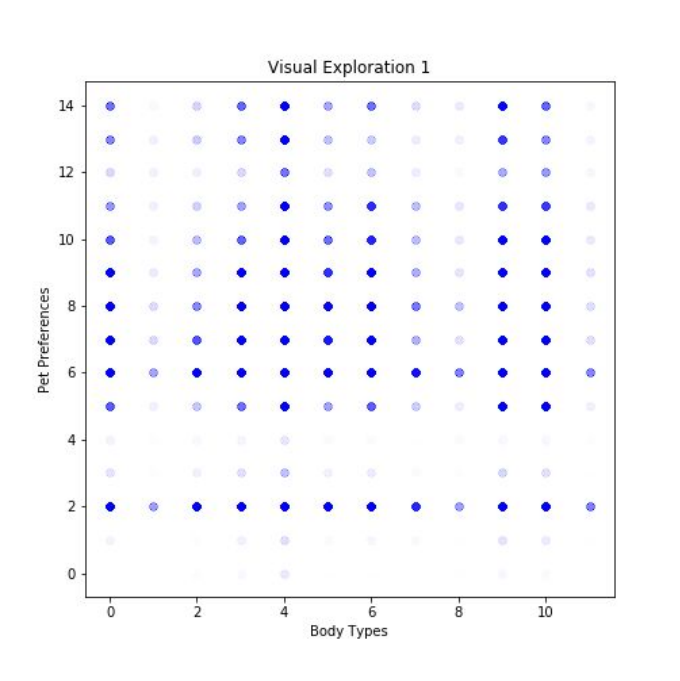
Amazing Title Slide

Machine Learning Fundamentals
Héctor Del Manzano
11 Nov 2018

Proceedings & Table of Contents

Visualize the Data*

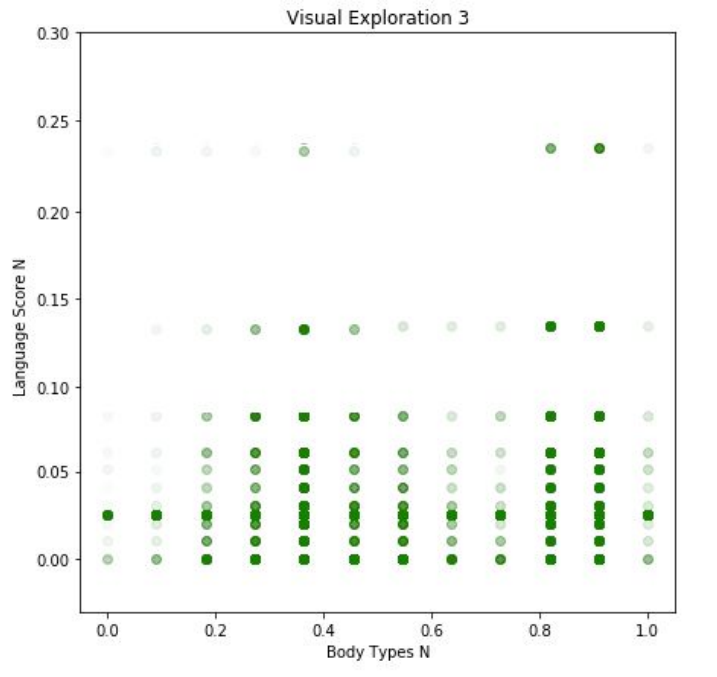


Scatter of one feature against another: On the x-axis we have encoded the body_type answers as integer numbers; low means underweight high means overweight highest means jacked! On the y-axis we encoded the multitude of pet preferences as an integer; low hints preference ("likes") while high hints commitment ("owns, has"). Diving in, we find:

- "Dislikes dogs dislikes cats" people (y=2) exist across all body types.
- "Dislikes dogs and likes cats", "dislikes dogs and has cats", "dislikes dogs", "dislikes cats" answers are almost non existent (y=3,4,1,2). Across any body types.
- People with "used up"(x=1) body type can't handle pets
- "Overweight" (x=8) people don't want to fuss with pets
- "Jacked" (x=11)people are too busy at the gym for pets!

*: For "Explore the Data" see the Jupyter Notebook, just a bunch of .head() and .value_counts()

Visualize the Data 2



Silly me normalized the body_type codes and the language_score (see Augmenting Data slide) and scattered one feature against the other.

Perhaps the other Normalization (not MinMax, as shown) technique would have been more productive, since the ranges are not evenly distributed.

If we can infer something (weakly shown) from this, it may be that there's a hint suggesting that "fit, athletic, jacked" people show higher Language aptitude (upper right). This Hypothesis makes sense from the standpoint of commitment: withholding from pizza and learning multiple languages both take strong commitment.

Question 1: Can we come up with a classifier for body_type from pet and language information?

Original Features Chosen:

- body_type (labels, target)
- pets
- speaks
- essays 0 through 9
- income

These Features Were transformed as explained to the Right.

- body_type_code: 0 is no-answer/NaNs, low is "used up"/"skinny", high is "overweight", top values is "fit, athletic, jacked".
- pets_code: encoded from low commitment to high commitment.
- speaks: Scoring function awarded points for quantity of languages spoken and also considered the usefulness of the language by awarding higher points to the languages with more speakers (See Code Line @@@)
- essays: were parsed for mentions of pets or pet related vocabulary
- income: simple map keeps the original value except when unanswered it is set to the national average of $44,564

Augmenting the Dataset

"speaks" feature is orignally testy:

```
english (fluently), german (okay)                                  160
english (fluently), russian (fluently)                             147
english (fluently), spanish (okay), french (poorly)                143
english (fluently), spanish (fluently), french (poorly)            143
english (fluently), french (okay), spanish (poorly)                133
english (fluently), french (poorly), spanish (poorly)              130
english (fluently), italian (poorly)                               115
                                                                   ...
english, hindi (poorly), sanskrit (okay), spanish (poorly)           1
english (fluently), yiddish (poorly), spanish (okay)                 1
english, spanish (fluently), italian (poorly), french (poorly)       1
english, french (fluently), c++, lisp                                1
english (fluently), japanese (poorly), hindi (fluently), spanish (poorly)   1
english (fluently), japanese (okay), spanish (okay), french (poorly), c++ (okay)   1
```

What a mess!

Assigning a unique integer code to each answer seems like a way to treat similar stuff as different stuff … So, we create the numerate_speaks function (right).

"essay0" through "essay9" parsed for pet synonyms and the count was saved as new feature.

def numerate_speaks(lang_str):

A function that uses the Wikipedia top-20 spoken languages around the world as a map, where the most spoken gets a 20 value and the 20th spoken receives a 1 value.

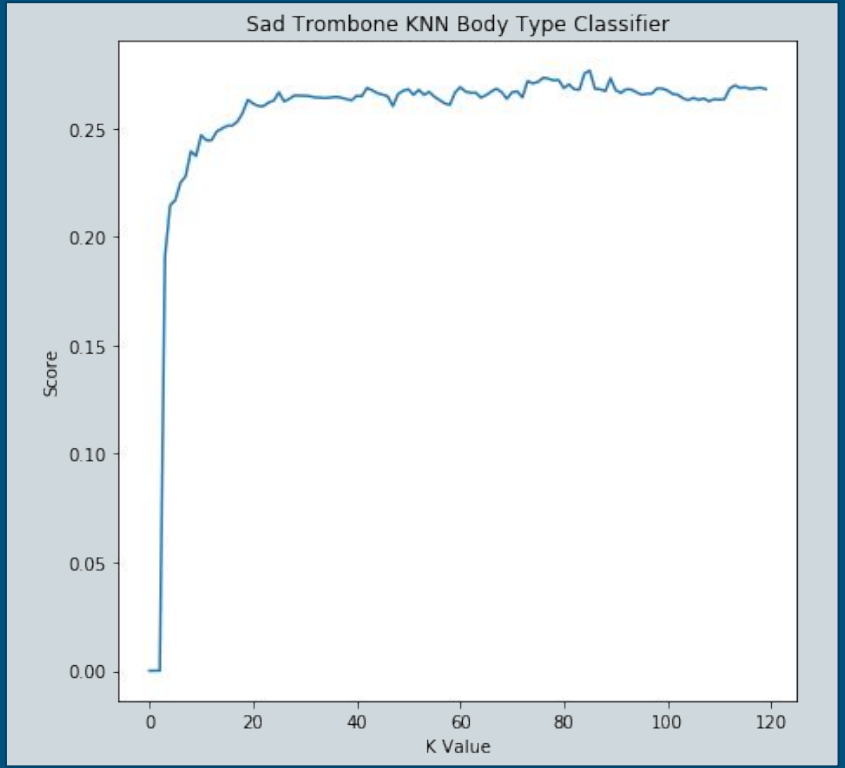
The final score takes this popularity value and adds to it a bonus for the raw number of languages you've listed. So if you've listed 3 languages the language_score will be 3 + 20 + 19 + 18 if they were Mandarin, Spanish and English (the top-3 spoken languages around the world).

If the language is not in the top-20 list ("c++, cool story") or, the entry is a NaN it gets assigned to "other" and that component contributes 0. Fluency ratings (parenthesis) are completely ignored.

Classification Approach: KNearestNeighbor

KNN Classifier Chosen:

- All the features were Normalized with the MinMaxScaler() (preprocessing package)
- train_test_split function used to split the 54650 records on the 4-feature ('income', 'langscore', 'pets', 'petscore') DataFrame into 80% training 20% validation groups.
- The target (labels) classifications were the body_type codes from 1 through 11.
- An Instantiate(k=variable), .fit(), .score loop was developed in order to find optimal K value (see right).
- Optimal K value found to be K=85
- Best Score was 0.2764



Question 2: Can we turn this around and infer income from body_type, language and pets information?

Features used as prepared for the Classifier.

The only difference is that now 'body_type' is an input/feature and "income" is the output/target

The scoring function for the Regressor chosen would not cooperate; was roadblocked, shamed, shunned, trashed by it. Something with the shapes and dimensions… until it worked and the score was awful.

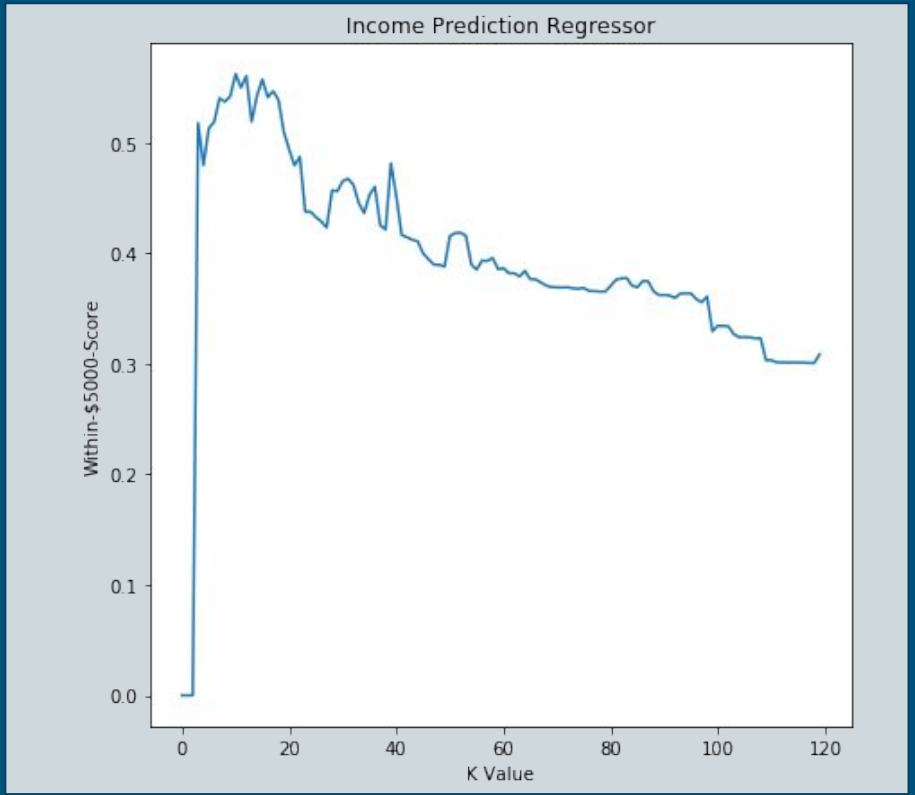
Own Simple scoring function developed.

After National Average substitution, income.value_counts()

44564	48442
20000	2952
100000	1621
80000	1111
30000	1048
40000	1005
50000	975
60000	736
70000	707
150000	631
1000000	521
250000	149
500000	48

Regression Approach: KNearestNeighbor

- All features were Normalized with the MinMaxScaler from preprocessing package
- train_test_split function for an 80/20 split used again.
- Features were: 'body_type', 'pets', 'langscore' and 'petscore'
- Looking to predict the 'income' figure target
- Initially K=85 chosen since this was "optimized" from the Classifier exercise. The .score() function for that yielded -0.10434 which according to documentation *checks notes* just plain sucks.
- An Instantiate(K=variable), .fit(), .score() loop was executed to find optimal K value (right).
- Optimal K value found at K=10



Conclusions and Next Steps

About the body_type Classifier:

- At first I was discouraged by the 0.27 score result. But then I remembered we're trying to classify among 11 different category types which means that the Pure-Chance rate should be 1/11 = 0.09...
- So this means our Classifier is 3 times better than Pure Chance. Please disregard the fact that I will get 3 out of 4 guesses wrong!
- We could explore reducing the target labels to just 3: underweight, in-shape, overweight. This may increase our classifiers' success
- We should also create an SVM Classifier with the same featureset and compare results.

About the Income Regressor

- Group size was significantly smaller than on the Classifier (K=85 vs K=10) this to me suggests that the feature added ("body_type") is information rich, we can say more with less neighbors in the group ... if that makes sense.
- A custom scoring function was found beneficial in the sense that it encapsulates a tolerance: The income prediction is considered "good" if it is off by up to $5k
- Look, the Regressor is garbage. Even with the $5k tolerance, the predictions score at 0.5617. If opened to $10k it is 0.6863.
- My conclusion is the "pet love/obsession", "body type" and "language aptitude" are not good predictors of Income

Final Notes

Answering the Income question in OKCupid is not popular; as seen on the Question-2 Slide, close to 45 out of 60 thousand people just don't answer. Perhaps substituting those no-shows and NaNs with the National Average was not the best strategy. This exercise should be run again but using the remaining ~15k records from people that did answer, and the results compared.

But assuming success, will we have a general predictor for personal Income, or a predictor *for Income for the type of people that like to answer the "how much to you earn" question?*

A second LinearRegressor exercise was performed but results are unworty... let's call it inconclusive. It can be found in the Notebook, towards the end.

Please do review the Jupyter Notebook for any questions you have about the reasoning or process or code. I was verbose with the comments.

I had a blast, and I am grateful for your teaching efforts. Best regards. -h.