Job Salary Prediction with Python

Published by John Maiden (http://blog.nycdatascience.com/author/john/) at January 27, 2015

Video Recording:

John Maiden's Video

Contributed by John Maiden. John took Vivian Zhang's "Data Science with Python: Machine Learning" class from November to December 2014. The post is based on his final project submission. Slides are available here (http://www.slideshare.net/jwmaiden/ds-final-project-jwm).

Browse

- Home (http://blog.nycdatascie
- R Visualization (http://blog.nycdatascience.co works/r-visualization/)
- R Shiny (http://blog.nycdatascience.co works/r-shiny/)
- Web Scraping (http://blog.nycdatascience.co works/web-scraping/)
- Machine Learning (http://blog.nycdatascience.co works/machine-learning/)
- Capstone (http://blog.nycdatascience.co works/capstone/)
- Faculty (http://blog.nycdatascience.co
- Meetup (http://blog.nycdatascience.co

- r-bloggers.com (http://www.r-bloggers.com/)
- Blog Post Guide (http://blog.nycdatascience.co work/blog-post-guide/)
- How to Add a Blog Post With S (http://blog.nycdatascience.co works/how-to-add-blog-with-s

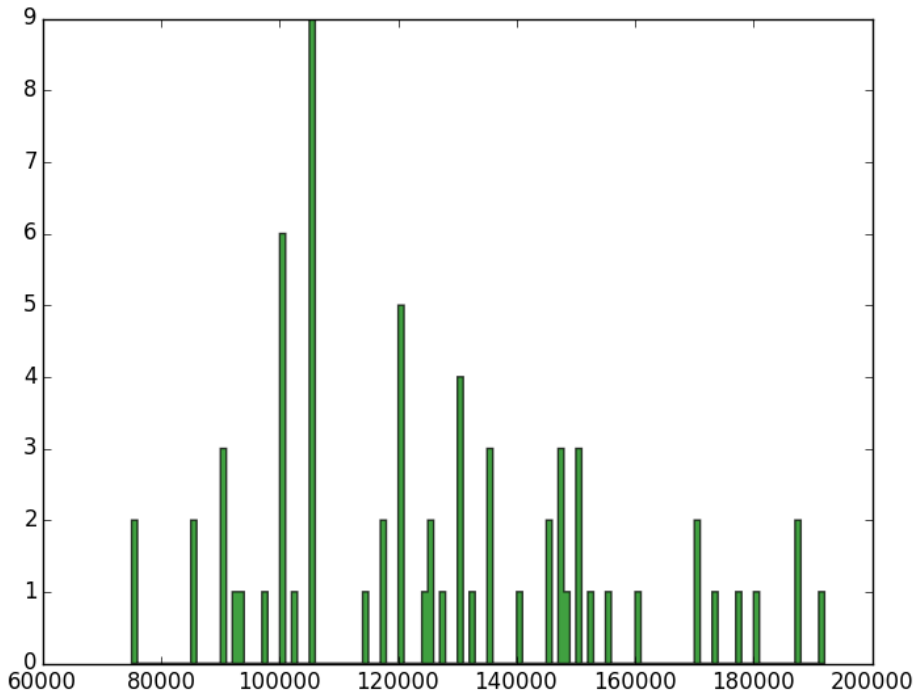
(https://nycdatascience.com/wp-content/uploads/2015/01/Money.png)

Salaries are a complicated topic. In some cultures it's extremely impolite to ask someone how much they earn in a year. In other cultures it's a common topic of discussion you'd ask a friend or acquaintance (or when I lived in China, something a person sitting next to me on the bus would ask). When looking for a job, it's a major deciding factor for many people. Sometimes the actual number is important, other times it's relative ("Am I being offered a salary that is below or above market?"). Most sites that offer salary information cover a limited range of jobs and industries (look for "Java Developer" in "New York, New York" (http://swz.salary.com/SalaryWizard/LayoutScripts/Swzl_SelectJob.aspx?jobkeyword=java%20developer&location=New%20York,%20NY) at salary.com - 26 results, but only half look like actual developer jobs), and those that have data usually do not adjust for factors like location, industry, and company size.

Besides sites like Salary.com, there are companies that use data science to predict salaries for jobs within certain industries or geographic locations (see projectSHERPA (http://projectsherpa.com/) or Adzuna by way of their Kaggle competition (http://www.kaggle.com/c/job-salary-prediction)). This brings transparency to a murky side of the market and fills in the data when data is not available.

Using projectSHERPA's database of jobs, the scope of this project was to use the techniques learned in "Data Science with Python: Machine Learning" to predict base salaries for data science jobs in NYC. Additional company information was scraped from LinkedIn, providing useful data such as company size, industry, and company specialties. The data science jobs were identified through filtering for specific keywords in the job title and job descriptions and filtering out required skills that suggested the job wasn't a technical position (e.g. requiring "Salesforce" experience).

Using the filters I found 585 unique data science jobs in NYC across 262 companies. Unfortunately, only 67 of those had posted salaries, which is a very small sample size. I had a previous salary model that I had developed for projectSHERPA, so I decided to split my testing into two groups - see if I could predict using the 67 jobs with posted salaries, as well as test against the 584 salaries produced by "the other model". Here is the salary distribution of the posted jobs:



(https://nycdatascience.com/wp-content/uploads/2015/01/Pay-Rate-Histogram-v3.png)

And here's the distribution of the 584 salaries as produced by "the other model":

Tags

API (http://blog.nycdatascience.co

Big Data (http://blog.nycdatascience.com/t data/)

D3.js (http://blog.nycdatascience. js/)

data science (http://blog.nycdatascience.com/t science/)

data visualization (http://blog.nycdatascience.com/t visualization-2/)

Demo Day (http://blog.nycdatascience.com/t day/)

gbm (http://blog.nycdatascience.com/t

ggplot2 (http://blog.nycdatascience.com/t

googleVis (http://blog.nycdatascience.com/t

Hadoop (http://blog.nycdatascience.com/t

higgs boson (http://blog.nycdatascience.com/t boson/)

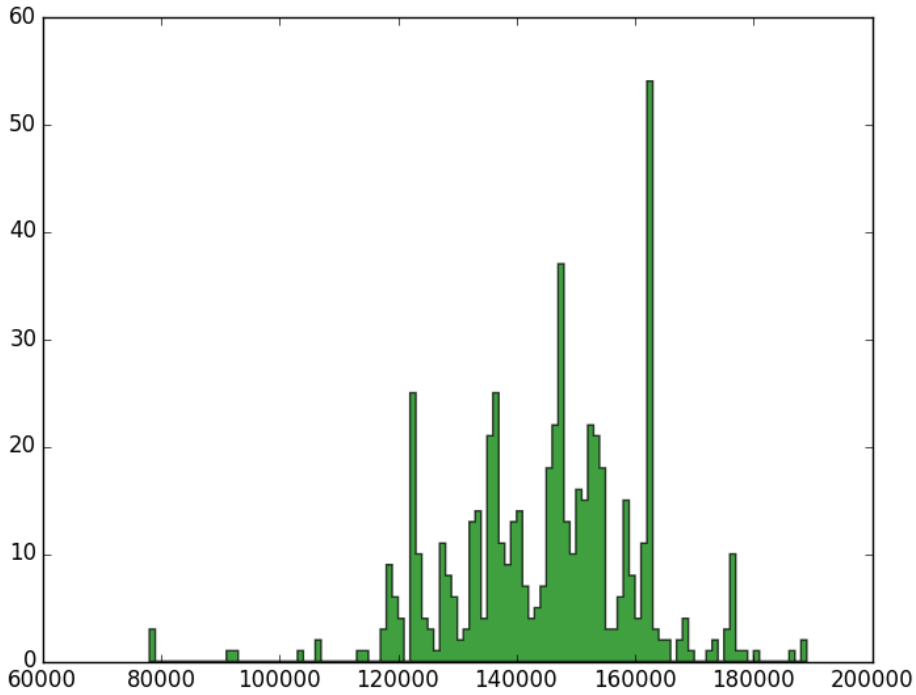
Kaggle (http://blog.nycdatascience.com/t

Logistic Regression (http://blog.nycdatascience.com/t regression/)

machine learning (http://blog.nycdatascience.com/t learning/)

Maps (http://blog.nycdatascience.com/t

neural network (http://blog.nycdatascience.com/t network/)



(https://nycdatascience.com/wp-content/uploads/2015/01/Est-Salary-Histogram-v3.png)

Since most of the competitors in Adzuna's Kaggle competition used a Random Forest model, I decided that would be a good place to start. I used Random Forest with a collection of parameters (via Scikit-Learn's grid search functionality) on the data, with a 50% Training /50% Test split on the smaller data set and 80% Training / 20% Test split on the larger data set. I was also inspired by Zygmunt Zając's post (http://fastml.com/predicting-advertised-salaries/)where he said "More data beats a cleverer algorithm, especially when a cleverer algorithm is unable to handle all of data (on your machine, anyway)". Since I had more data than the original Kaggle competition (by pulling in the company data from LinkedIn), I could also see if linear models (Ordinary Linear Regression, Ridge Regression, and Lasso Regression) could perform as well if not better than a Random Forest model. Results for the models was not optimistic - good scores on the Training data, but poor (if not lousy) on the Test data.

The next step was to look again at the data - I was rounding the salary numbers to the nearest 1K, but that is too granular when it comes to salary decision behavior (would 101K vs. 102K make a huge difference to you?). I rounded to the nearest 10K unit (101K = 10, 128K = 13) and added in new classification models (Linear Discriminant Analysis, K-Nearest Neighbor) to see if the results could improve. Here is the salary distribution for the posted jobs after rounding:

Neural networks (http://blog.nycdatascience.com/networks/)

NYC (http://blog.nycdatascience.com/

NYC Open Data (http://blog.nycdatascience.com/open-data/)

Open Data (http://blog.nycdatascience.com/data/)

pandas (http://blog.nycdatascience.com/

python (http://blog.nycdatascience.com/2/)

python machine learning (http://blog.nycdatascience.com/machine-learning/)

python web scraping (http://blog.nycdatascience.com/web-scraping/)

python webscraping (http://blog.nycdatascience.com/webscraping/)

Python Workshop (http://blog.nycdatascience.com/workshop/)

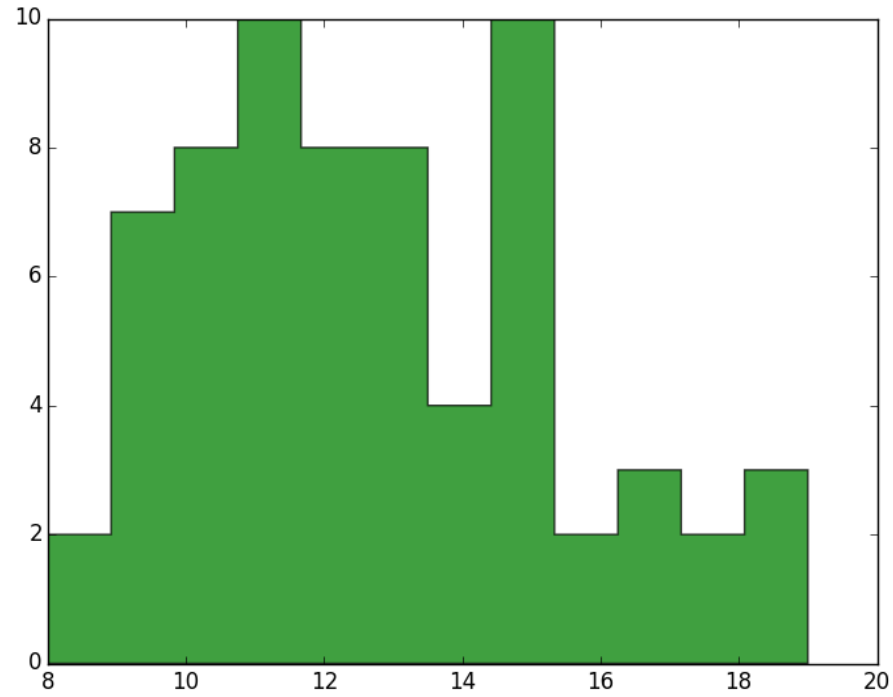
R (http://blog.nycdatascience.com

random forest (http://blog.nycdatascience.com/forest/)

R Programming (http://blog.nycdatascience.com/programming/)

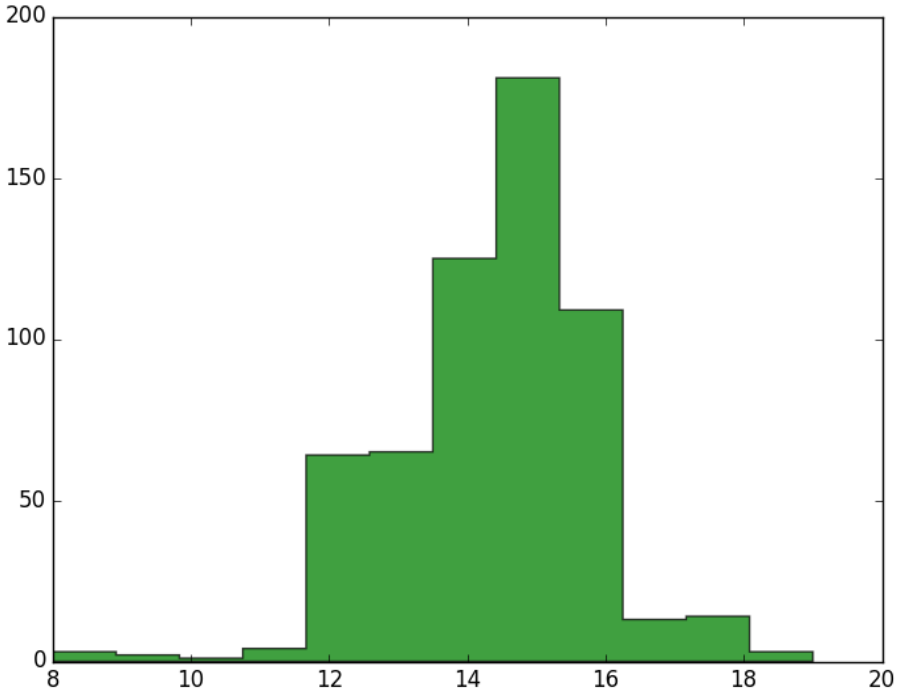
R Shiny (http://blog.nycdatascience.com/

r studio (http://blog.nycdatascience.com/studio/)



(https://nycdatascience.com/wp-content/uploads/2015/01/Pay-Rate-Range-Histogram-v3.png)

And here is the salary distribution after rounding for the numbers produced by "the other model":



(https://nycdatascience.com/wp-content/uploads/2015/01/Est-Salary-Range-Histogram-v3.png)

The final results - smoothing the data did produce better numbers, but the results were still unimpressive. Random Forest, which produced the best numbers, had an accuracy of 58% on the Test set of the smoothed larger data set. Here are the results of the best performing models, where "Pay Rate" = jobs with posted salary (67 records), and "Estimated Salary" = salaries as predicted by "the other model" (584 records).

R Visualization (http://blog.nycdatascience.com/t visualization/)

R Workshop (http://blog.nycdatascience.com/t workshop/)

Scrapy (http://blog.nycdatascience.com/t

scrapy visualization (http://blog.nycdatascience.com/t visualization/)

Selenium (http://blog.nycdatascience.com/t

Shiny (http://blog.nycdatascience.com/t

Sports (http://blog.nycdatascience.com/t

statistics (http://blog.nycdatascience.com/t

SVM (http://blog.nycdatascience.com/t

Tableau (http://blog.nycdatascience.com/t

visualization (http://blog.nycdatascience.com/t

web scraping (http://blog.nycdatascience.com/t scraping/)

webscraping (http://blog.nycdatascience.com/t

XGBoost (http://blog.nycdatascience.com/t

Pay Rate

Model	Training Score	Testing Score
KNN	1.000	0.147

Estimated Salary

Model	Training Score	Testing Score
Decision Tree	1.000	0.342

Smoothed Pay Rate

Model	Training Score	Testing Score
LDA	0.585	0.286

Smoothed Estimated Salary

Model	Training Score	Testing Score
Random Forest	1.000	0.581

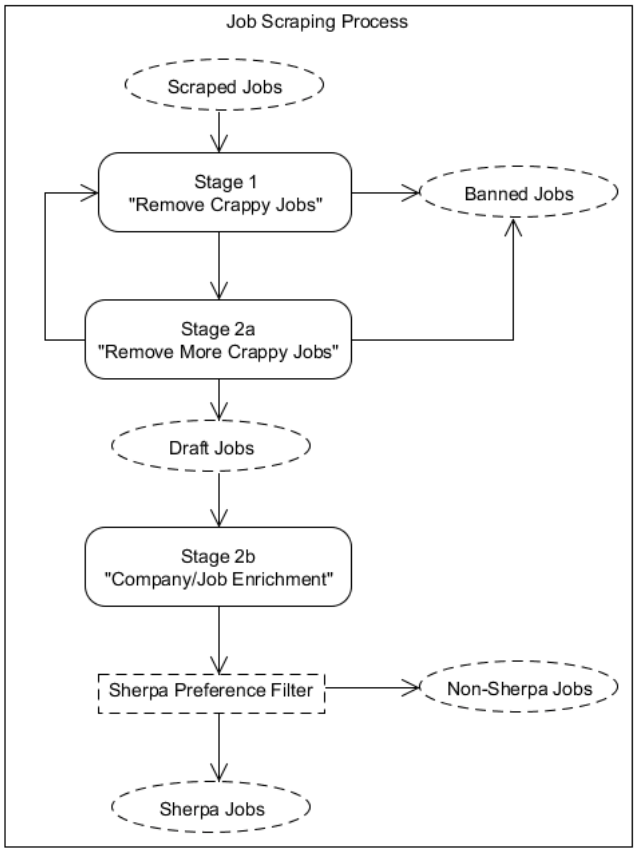
(https://nycdatascience.com/wp-content/uploads/2015/01/Data-Results.png)

How could we improve the results in the future? First of all, I looked at 585 jobs from a collection of 70k+ jobs, so I could definitely go after more jobs instead of just looking at data science jobs in NYC. The shape of the data is also important - I rounded to the nearest 10K to get better numbers, but I could play with the relative size of the ranges to reflect job seeker behavior (to a job seeker, 20K vs. 30K is definitely more important than 120K vs. 130K). Finally, pulling the data from LinkedIn required a fuzzy match between the name of the company in the database and whatever I could get from LinkedIn; looking through the matching results it looks like I got the company right about 90% of the time, so there is room for improvement.

So why do I think a project like this (predicting salaries from job posts) is a marketing gimmick? Most of the jobs that have posted salaries are from agency recruiters, which means that a) the jobs are usually entry level or mid-tier, and b) I don't know anything about the actual company posting the job. Based on my observations of the jobs in the projectSHERPA database, usually only 15% of the jobs posted include salaries, so we're always extrapolating from a small data set. Next, making predictions based on keywords in the job title or job description isn't precise. Job titles vary from company to company, as well as industry to industry ("Vice President" is higher up in the management structure in Manufacturing than in Finance), and I think everyone will agree that job descriptions are mainly boilerplate. Finally, this only works for jobs that don't pay a significant amount of the compensation in bonus or benefits. Companies usually do not list non-base compensation in the job post. Startups are a great example of this - the base salary is usually lower than the market for a similar job at a more established company, but the upside from stock options can be huge.

I'd like to finish by saying that I think that predicting salaries is very useful - modeling salaries using total compensation numbers (base + bonus + benefits) based on data provided directly by companies is very important data, both for job seekers and other market participants. I'd discount using job post data to predict salaries - I think it worked well for the Adzuna competition, where most of the salaries that I saw paid hourly rates or had a compensation structure where base was a significant component of the total. I don't think it produces accurate numbers for jobs that require candidates with high-level technical skills.

The code I used is below - I can't share the code that I used to extract the job posts (😕) but I can show you everything after that. For the record, here's the projectSHERPA job scraping process:



(https://nycdatascience.com/wp-content/uploads/2015/01/Job-Scraping.png)

Let's start with how I came up with the keywords to retrieve the data science jobs. Originally I looked through a spreadsheet filled with the jobs that I had filtered based on a few initial keywords in the job description (e.g. "data science", "machine learning"). I extracted the job title and job description from those posts to look at common words, bigrams, and trigrams; this took a couple of iterations to find the right set of keywords that produced jobs that were "data science-y" enough (i.e. not developer jobs, not sales related jobs, not business analyst jobs). See function test_ds_words() for the text analysis outputs.

```
def get_word_tokenize(text):
    # Tokenize a string of input text

    # Input
    # text: input text

    # Output
    # list of tokenized words

    sentences = [s for s in nltk.sent_tokenize(text)]
    normalized_sentences = [s.lower() for s in sentences]
    return [w.lower() for sentence in normalized_sentences for w in nltk.word_token
ize(sentence)]

def get_top_n_words(words, n, stopwords):
    # Return the top n most frequent words from a tokenized list of words, using th
e input stopwords
```

Once I had a set of jobs that I was satisfied with, I used the Company search API that comes with Python-LinkedIn (https://github.com/ozgur/python-linkedin) to extract additional data about the company that posted the job. I needed a fuzzy string function to match the parsed company name from the job post to the results returned from LinkedIn.

```
stemmer = stem.PorterStemmer()

def normalize(s):
    # Normalizes + tokenizes input text + punctuation
    # from http://streamhacker.com/2011/10/31/fuzzy-string-matching-python/

    # Input
    # s: input text

    # Output
    # string of cleaned text

    words = tokenize.wordpunct_tokenize(s.lower().strip())
    return ' '.join([stemmer.stem(w) for w in words])

def fuzzy_match(s1, s2):
    # Calculates the edit distance between two normalized strings
```

Now we arrive at the LinkedIn data retrieval code - I pulled in basic company data (e.g. company name, homepage, company type, company size), industry, and specialties (a collection of unstructured tags). If you want to test the code, you'll have to go to the LinkedIn Developer site (http://developer.linkedin.com/) to get an API key.

```
from linkedin import linkedin

# Linkedin oauth details:
LINKEDIN_CONSUMER_KEY       = ''
LINKEDIN_CONSUMER_SECRET    = ''

# For LinkedIn API calls:
LINKEDIN_OAUTH_USER_TOKEN = ''
LINKEDIN_OAUTH_USER_SECRET = ''
RETURN_URL = 'http://localhost:8000'

def update_company_data_from_linkedin():

    # Retrieves all of the company names from the job postings,
    # and queries LinkedIn for additional information

    # Define CONSUMER_KEY, CONSUMER_SECRET,
```

The next step was to merge the data I retrieved from LinkedIn with the job data, and convert the LinkedIn codes (https://developer.linkedin.com/documents/company-lookup-api-and-fields) into numeric values. See function prepare_and_merge_data() for the merge step.

```
def estimate_seniority(job_position):
    # Estimates the seniority of a job
    # based on key words in the job position text

    # Input
    # job_position: input text

    # Output
    # seniority: 'junior', 'default', or 'senior'

    seniority = 'default'
    jobtitlewords = job_position.lower().split()

    # ignore internships
    if (('intern' in jobtitlewords)
        or ('internship' in jobtitlewords)):
        return 'INTERN'
```

I wanted to test multiple models on different versions of the data, so I originally wrote a separate function for each model and collection of data. I quickly realized it was not scalable (I kept "fixing" the code everytime I thought of a new way to test the data), so I wrote a collection of generic modeling functions. Here's the function for running a scikit-learn model and extracting the score and mean-squared-error:

```
def get_model_values(model, x_train, y_train, x_test, y_test):
    # Fit a model and return the score and mse

    # Input
    # model: scikit-learn model
    # x_train: independent variables training set
    # y_train: dependent variable training set
    # x_test: independent variables test set
    # y_test: dependent variable test set

    # Output
    # train_score: training score
    # test_score: test score
    # train_mse: training mse
    # test_mse: test mse

    model.fit(x_train, y_train)
```

Here's a function to return score and mean-squared-error for a scikit-learn model using a grid search:

```
def get_grid_search_values(model, grid_params, x_train, y_train, x_test, y_test, sc
oring_criteria = 'mean_squared_error'):
    # Run a grid search on a model, and return the train / test score and MSE on th
e best result

    # Input
    # model: scikit-learn model
    # grid_params: dict of parameter space
    # x_train: independent variables training set
    # y_train: dependent variable training set
    # x_test: independent variables test set
    # y_test: dependent variable test set
    # scoring_criteria: model scoring criteria

    # Output
    # best_model: model that produced the best results
    # para_search.best_params_: best grid parameters
```

I encountered a problem with colinearity when using all of the data features with the LDA model, so I also wrote a function to retrieve "Best-K" for a scikit-learn model, using test score as the evaluation criteria.

```
def get_best_k_model(model, max_k, x, y):
    # Fit a model using a range of best-k values,
    # returning the model that produces the best test score

    # Input
    # model: scikit-learn model
    # max_k: maximum k-value to iterate to (inclusive)
    # x: independent variables
    # y: dependent variable

    # Output
    # best_k: Number of dependent variables using to produce output
    # train_score: training score
    # test_score: test score
    # train_mse: training mse
    # test_mse: test mse
```

Combining all of the above code, here is the entire set of code that I used to predict job salaries. The modeling component is kicked off in function run_model().

```
import pandas as pd
from pandas.io.pickle import read_pickle
import nltk
from nltk import stem, tokenize
from linkedin import linkedin
import numpy as np
import sklearn.cross_validation as cv
from sklearn import linear_model, metrics, ensemble, grid_search, lda, neighbors, t
ree
import math
import matplotlib.pylab as py
from sklearn.decomposition import PCA
import sklearn.feature_selection as fs
import xlwt

# Linkedin oauth details:
LINKEDIN_CONSUMER_KEY       = ''
```

John Maiden (http://blog.nycdatascience.com/author/john/)

Related posts

May 28, 2017

An Interesting Study: Exploring Mental Health Conditions in the Tech Workplace (http://blog.nycdatascience.com/student-works/r-shiny/interesting-study-exploring-mental-health-conditions-tech-workplace/)

Read more (http://blog.nycdatascience.com/student-works/r-shiny/interesting-study-exploring-mental-health-conditions-tech-workplace/)

May 27, 2017

Analyzing the Evolution of Rap Music from 1989 to 2016 (http://blog.nycdatascience.com/student-works/web-scraping/analyzing-evolution-rap-music-1989-2016/)

Read more (http://blog.nycdatascience.com/student-works/web-scraping/analyzing-evolution-rap-music-1989-2016/)

May 15, 2017

Scraping Fiverr To Analyze Freelancing Market Trends (http://blog.nycdatascience.com/student-works/scraping-fiverr-analyze-freelancing-market-trends/)

Read more (http://blog.nycdatascience.com/student-works/scraping-fiverr-analyze-freelancing-market-trends/)

1 Comment

#projectSHERPA (http://projectsherpa.com/) says: (http://blog.nycdatascience.com/student-works/job-salary-prediction-with-python/?replytocom=22#respond)
February 2, 2015 at 3:56 pm (http://blog.nycdatascience.com/student-works/job-salary-prediction-with-python/#comment-22)

Thanks for sharing that insight. Quite interesting John.

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name * **Email *** **Website**