**Business Trip Report**

**Scipy 2019 austin, texas  
(research on Data science tools and Theory)**

**July 2019**

**EUNJAE JANG**

**Quant TEAM**

****

**Subjects in Scipy 2019**

**Background**

PyCon is the largest annual gathering for the community using and developing the open-source Python programming language. Python is widely used for many different applications, and over time, the huge community around this open source language has created various tools to efficiently work with Python.

A number of these tools have been built specifically for data science, and as a result, it is currently most widely used and disseminated language for various research domains of machine learning and artificial intelligence.

**Conference Overview**

Attendees of the conference were engineers, web-developers, data-scientists and beginners to the Python language. Speakers ranged from field practitioners to core developers of Python, who gave tutorials and talks on Python’s platform and environment, practices to streamline workflows, data-science and coding topics, and lastly new developments that will reinforce the Python platform.

**Thoughts after Conference**

Efficient programming is a key-ingredient skillset for data analysts, to say the least for financial quant specialists. Often times, buy-side analysts tend to be bombarded by myriads of sell-side strategies or academic papers that claim to guarantee superior performance in investments. And most likely, these claims are untested or hastily replicated to emulate their strategies which most likely deliver less than expected results.

In many respects, KIC’s Quant team has the advantage of well-supported resource as well as team members coming from various backgrounds. However, given the larger task at hand with ever-changing financial landscape and growing sources of new alternative data (ie. Big Data), we are faced to collaborate more efficiently and deploy new strategies in a seamless mechanism now more than ever.

PyCon conference has touched upon many essential topics which are easily overlooked but important for building a stronger data science team. Collaborative team-effort is only possible when ideas are communicated transparently and tested freely by others. Tools such as Docker, Python virtual environments, and unit-tests are imperative goals in order to share reproducible data science projects. With these pillars in place, ever-increasing tools and new data analytical tools can be tested more robustly and shared across team members that would pave way to a stronger and more efficient team in the long-run.

**Key Take-Aways during Conference**

Although PyCon conference touched upon loads of topics (36 tutorials and 114 talks), regarding the ones that were found to be most useful in Data-Science, these were the key take-away topics:

1. Python Virtual Environment
2. Numerical Computation Optimization
3. Tools to Improve Workflow
4. Agent-based Modeling & Analysis
5. Singular Value Decomposition Modeling
6. New Developments in Python
7. Programming Best Practice

**Take-Away Summary**

**Bayesian Statistics and Python Usage**

* + **Conjoint Probability**

Writing means, the probability of occurrences A and B being both true.

In case of two independent occurrences, such as tossing a coin, the probability of both 2 occurrences to have heads is:

when two occurances are independent

However, if two occurrences are dependent, such as A being raining today and B being raining tomorrow, it is intuitive to infer that .

Thus, the probability of a conjunction is:

* + **Bayes Theorem**

Bayes theorem can be derived by picking up from above concept.

so, rewriting this is:

which leads to Bayes Theorem:

A simple example which utilizes this equation is, the cookie problem:

Suppose there are two bowls of cookies. Bowl 1 contains 30 vanilla cookies and 10 chocolate cookies. Bowl 2 contains 20 of each.

Now suppose you choose one of the bowls at random and, without looking, select a cookie at random. The cookie is vanilla. What is the probability that it came from Bowl 1?

Denoting B1 for the hypothesis that the cookie came from Bowl 1 and V for the vanilla cookie, we could write the problem as solving following equation:

Then assuming selecting either Bowl 1 or 2 is random, .

Selecting vanilla cookie from Bowl 1 is and selecting vanilla cookie out of total cookies, .

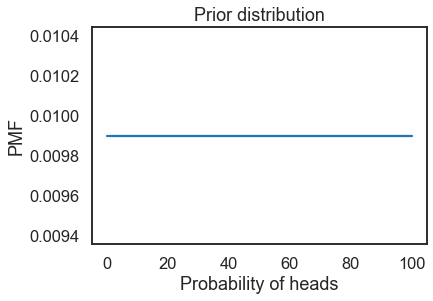
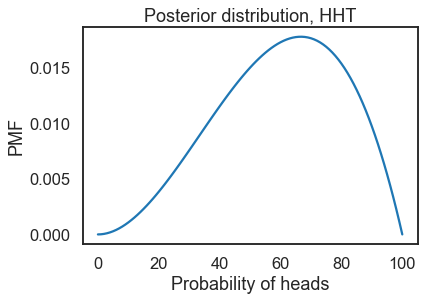
Thus,

* + **Bayesian Theorem with Diachronic Interpretation**

A more general way of interpreting Bayesian Theorem is to utilize it for updating the probability of a hypothesis, *H*, in light of some body of data, *D*.

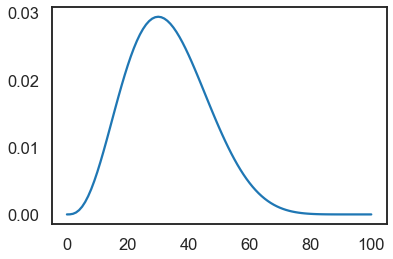
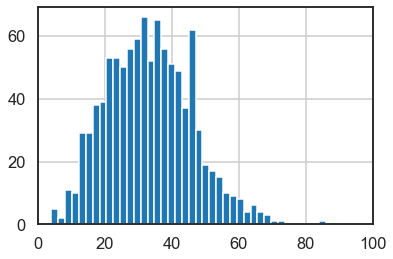
This way of thinking about Bayes’s theorem is called the diachronic interpretation. “Diachronic” means that something is happening over time; in this case the probability of the hypotheses changes, over time, as we see new data.

In this interpretation, each term has a name:

1. is the probability of the hypothesis before we see the data, called the prior probability, or just **prior**.
2. is what we want to compute, the probability of the hypothesis after we see the data, called the **posterior**.
3. is the probability of the data under the hypothesis, called the **likelihood**.
4. is the probability of the data under any hypothesis, called the **normalizing constant**
   * Examples of Bayesian Statistics using Python
5. Euro Problem  
     
     
   This problem requires proceeding in two steps. The first is to estimate the probability that the coin lands face up. The second is to evaluate whether the data support the hypothesis that the coin is biased.  
     
   Any given coin has some probability, *x*, of landing heads up when spun on edge. If a coin is perfectly balanced, we expect x to be close to 50%, but for a lopsided coin, x might be substantially different.  
     
   The key point in using Bayesian Statistics for solving this problem is not about the search for the exact value of *x*, per se, however the distribution of *x*.  
     
   Thus, *x* should be on a range between 0 to 100, where is the hypothesis that probability of heads is . Within python setup, one could starting the prior with a uniform distribution where the probability of is the same for all *x*, as such:  
     
      
     
   Then with incoming data, new information can be updated into the likelihood probability of coin toss resulting in heads with a function like:  
   If is true, the probability of heads is and the probability of tails is .  
     
   **def** Likelihood(self, data, hypo):  
    x = hypo  
    **if** data == ‘H’**:**  
    return x / 100.0  
    **else:**  
    return 1 – x / 100.0  
     
   If so, having updated with 3 sequential occurrences data, ie. **Head –** **Head -** **Tail**, the updated posterior distribution would look like:  
     
      
     
   In such matter, one could argue with 3 input data that, the maximum probability for having a head with a coin toss is 67% with a general average probability of 60% likelihood.  
     
   Again, the key take-away from this exercise is that, while one wishes to find the probability for the result of a coin toss, the output result from a Bayesian statistical analysis is not a single value but rather a probability distribution which holds the information for the general likelihood.

"When spun on edge 250 times, a Belgian one-euro coin came up heads 140 times and tails 110. 'It looks very suspicious to me,' said Barry Blight, a statistics lecturer at the London School of Economics. 'If the coin were unbiased, the chance of getting a result as extreme as that would be less than 7%.' "

From “The Guardian” quoted by MacKay, Information Theory, Inference, and Learning Algorithms.

1. Random Selection with a Custom Probability Distribution  
   Numpy library in python, gives some powerful tools for generating random selection data.  
     
   Let’s say we would like to select a number between 0~100, under a custom probability distribution, such that the distribution could look like below:  
     
      
     
   Then, numpy has following function to make a random selection within above distribution:  
     
     
   Using this, one can create a 1000 sample of selections as following:  
     
      
   As can be observed, the sample roughly follows the distribution as designed to be.

np.random.choice(a, size=None, replace=True, p=None)

key-Parameters

-----------

a : 1-D array-like or int

If an ndarray, a random sample is generated from its elements.

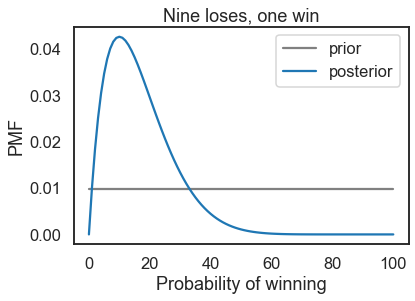
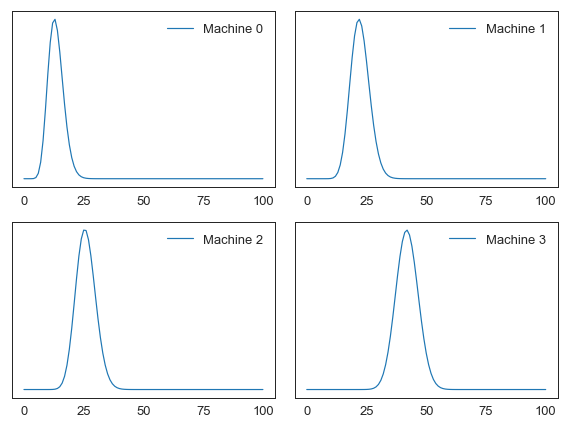
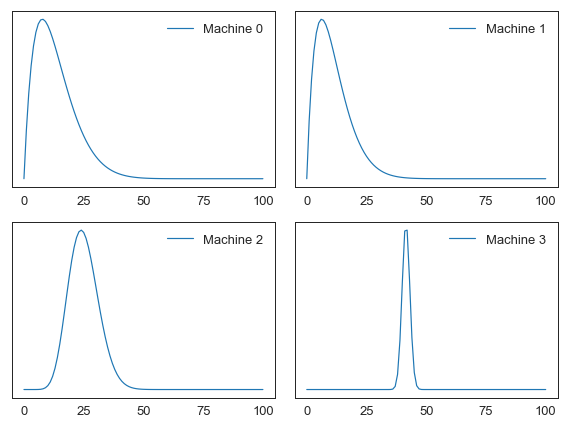
If an int, the random sample is generated as if a were np.arange(a)

p : 1-D array-like, optional

The probabilities associated with each entry in a.

If not given the sample assumes a uniform distribution over all

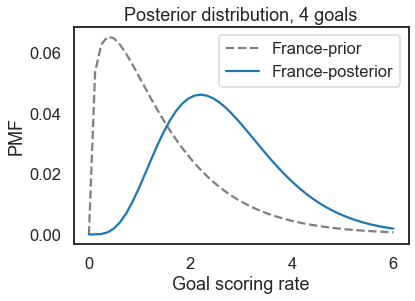
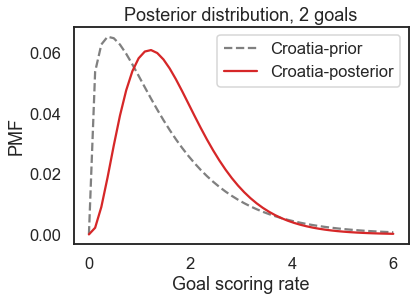
entries in a.

1. Bandit Problem  
     
     
   Considering that one could argue that an investment strategy is analogical to creating a pick-and-choose slot-machine strategy (ie. choosing the right investment strategy), this example is crucial for applying Bayesian Statistics for investment strategy.  
     
   All being equal in process with that of the “euro problem,” we’ll use the same selecting a number between 0~100 as our prior for eventually creating a success probability distribution for each slot machine, and the same likelihood update function. Therefore, if one has a slot-machine which ran a sequence of 1 win and 9 loses, hereafter denoted as ‘WLLLLLLLLL’, the posterior distribution would look like:  
     
      
   \*where grey line is the initial prior distribution and blue is the updated posterior after updating the occurrence data.  
     
   A distribution as such, gives meaningful information such as that the overall weighted-average probability of winning is 16.7% whereas the occurrence with the highest percentage for wining is 10%.  
     
   Now, say we have 4 slot machines with 10, 20, 30, 40% of winning probabilities and create random sequences of win or lose occurrence for each slot machines. For 100 sequence of occurrences for each slot machine, the winning distributions for each slot-machine would look like this:  
     
      
   Resulting average winning probabilities are, 13.39%, 22.32%, 25.89%, 41.96%.  
     
   Since, these winning probability distributions are updated after every round of plays by 4 slot-machine, one can use this data to choose which slot-machine to bet on with every round of play with the updated distribution.  
     
   By setting up the right kind of generating functions, following 1000 iterations of slot-machine plays can set-off following result:  
     
   num\_plays = 1000  
   count\_win = 0  
   for \_ in range(num\_plays):  
    count\_win = choose\_play\_update(  
    beliefs, record=True, count=count\_win)  
     
      
     
   After the run, when counting how many times each slot-machine was selected, the result is:  
     
   Slot-Machine 0 : 13 times  
   Slot-Machine 1 : 16 times  
   Slot-Machine 2 : 46 times  
   Slot-Machine 3 : 925 times  
     
   And the total number of wins was 397 times which is approximately 40% win out of 1000 runs.  
     
   However, on the other hand, if one does a random pick and choose slot-machine strategy, the result would have come up as:  
     
   Slot-Machine 0 : 252 times  
   Slot-Machine 1 : 248 times  
   Slot-Machine 2 : 257 times  
   Slot-Machine 3 : 243 times  
     
   And the total number of wins are 235 times which is approximately 23.5% win out of 1000 runs.  
     
   Thus, updating the model and choosing slot-machine through Bayesian approach not only **improved the outcome** but also **successfully allocated one’s bet to the most winning-probable slot-machine** through empirical trials.

Suppose you have several "one-armed bandit" slot machines, and reason to think that they have different probabilities of paying off.

Each time you play a machine, you either win or lose, and **you can use the outcome to update your belief about the probability of winning.**

Then, to decide which machine to play next, you can use the "Bayesian bandit" strategy.

1. Synthetic Outcome Generation with Bayesian Statistics  
   When predicting complex probability, Bayesian statistics can become handy for generating likely occurrences and deriving the outcome probability.  
     
   Also, it is important to know the different probability distributions available and choose the appropriate distribution to approximate the phenomena.  
     
   One type of problem might be solving the World Cup problem:  
     
     
   In order to setup a probability distribution for the number of goals a team is likely to score, one could use a **Poisson distribution** which is non-negative and discrete (ideal for this case of soccer goals).  
     
   First one could set a prior distribution which would set the max-probable average number of goals for any team to be 1.4 goals (based on data from previous World Cups).  
   Then after, one could update the prior with 4 goals for France and 2 goals for Croatia, which would have an updated PMF as below:  
     
       
     
   Now that distributions for each team has been updated, this can become the basis for creating random game scores and predict the likelihood for the match-up outcome.

In the 2018 FIFA World Cup final, France defeated Croatia 4 goals to 2. Based on this outcome, we can answer the following questions:

1. How confident should we be that France is the better team?
2. If the same teams played again, what is the chance Croatia would win?

**Bayesian Statistics Modelling with PyMC3**

PyMC3 library is a powerful tool for testing Bayesian statistical hypothesis in a concise manner whilst harnessing multi-threading and GPU computation. In real-world problems, there are problems where more-than one parameter are unknown and may be varying. The beauty in PyMC3 is the ability to add more than one priors for problems with multiple parameters, and yet compute the outcome probability with added computational improvement through library utilities.

The basics of Bayesian model building follows the steps of:

1. To completely specify the model in terms of probability distributions. This includes specifying
   * what the form of the sampling distribution of the data is and
   * what form describes our uncertainty in the unknown parameters (This formulation is adapted from [Fonnesbeck's workshop](https://github.com/fonnesbeck/intro_stat_modeling_2017/blob/master/notebooks/2.%20Basic%20Bayesian%20Inference.ipynb" \t "_blank) as Chris said it so well there).
2. Calculate the posterior distribution.

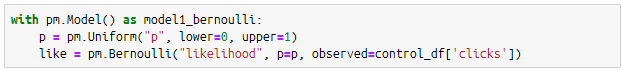
Under the hood, PyMC3 will compute the posterior using a sampling based approach called Markov Chain Monte Carlo (MCMC) or Variational Inference.

1. Example 1 : Verify from Bayesian statistics whether higher contrast button in a website will induce more clicks from users.

Say that we don't think enough people are clicking a button on my website, but we hypothesize that it's because the button is a similar color to the background of the page, meaning they're a difficult time finding the button to click.

In a **‘control’** group we show the original web-page, and count the number of clicks by user. In the **‘test’** group we will show the web-page with a higher contrast button and count the clicks by user.

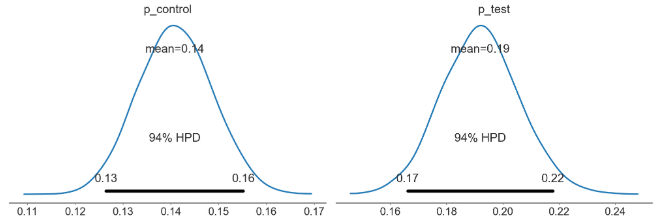
PyMC3 allows creating a Bayesian model wrapped in a python function as following:

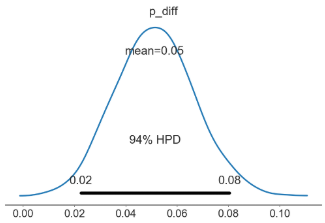


In above, we’ve set up a uniform distribution for the probability to click (0~100%) as prior and used a likelihood update function using the Bernoulli distribution.

By setting up two of same models, one for the control group and one for the test group, we will have two posterior distributions which we can compare against each other to verify the hypothesis.

The outcome is as following:





This shows that there is 5% difference in click rate when using higher contrast button.

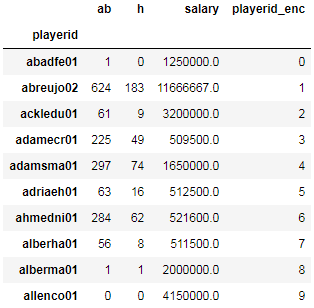
1. Example 2 : Baseball Players

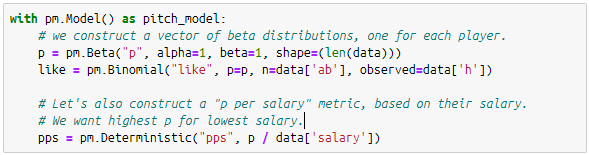
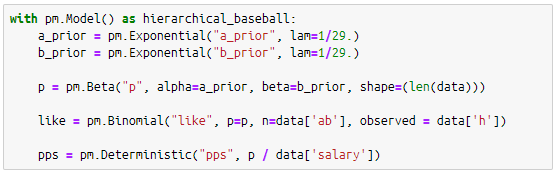
Further usage case for Bayesian statistics can be tested in PyMC3 with following example:

Goal of this analysis is to identify which player we want to make an offer to join our baseball team based on the batting stats.

We are going to make a decision on the basis of just batting average and their salaries.

‘ab’ column stands for at-bat, and ‘h’ stands for hit counts. One would assume that batting average has a direct impact on salary.

… (to 805 players)

* + - * ***Defining Model:***Since we have a number of trials (ab), with a number of successes (h), from which we want to estimate a player-specific property: p (probability of hitting a pitch), we can use Bernoulli distribution as our prior.  
          
        Code can be written as following:  
          
        The core strength behind this modeling in PyMC is that, be assigning a ‘shape’ to the function, we are setting up 805 distributions for each player, instead of just one for the whole.  
        The data richness that comes out of this is substantial compared with a simple 1-dimensional average hit per player.
      * ***Enhancing a Model by Defining Prior Parameters***Now, when a model is not well-rounded enough to take consideration for players with little to no data points, we might get hitting percentile outcome as following.  
        >>  
          
        This is however due to the fact that the model does not have a default distribution for players without enough datasets. We could solve this through defining the parameters for the prior to adjust for these cases. This type model is called hierarchical model.  
          
        An example for setting up this model is as following:  
          
          
        The change from this can be seen below with player, alberha01:

**NLP and Progress Until Today**

There has been so much growth in the field of NLP (Natural Language Processing), to the point that computers can now finish a sentence written by humans. While powerful and accurate at the same time, it is becoming more and more important to understand NLP technology in order to fully harness the power and utilize its strength in assisting investment decisions.

In a general list, following sums up the different studies in NLP which which have made progress:

* Automatic speech recognition
* CCG
* Common sense
* Constituency parsing
* Coreference resolution
* Dependency parsing
* Dialogue
* Domain adaptation
* Entity linking
* Grammatical error correction
* Information extraction
* Language modeling
* Lexical normalization
* Machine translation
* Missing elements
* Multi-task learning
* Multi-modal
* Named entity recognition
* Natural language inference
* Part-of-speech tagging
* Question answering
* Relation prediction
* Relationship extraction
* Semantic textual similarity
* Semantic parsing
* Semantic role labeling
* Sentiment analysis
* Shallow syntax
* Simplification
* Stance detection
* Summarization
* Taxonomy learning
* Temporal processing
* Text classification
* Word sense disambiguation

Although all studies are important in their own respect, these are some details for topics which are likely to be more important in their use-cases for NLP in finance:

1. **Information Extraction**Open Information Extraction approaches leads to creation of large Knowledge bases (KB) from the web. The problem with such methods is that their entities and relations are not canonicalized, which leads to storage of redundant and ambiguous facts. For example, an Open KB storing <Barack Obama, was born in, Honolulu> and <Obama, took birth in, Honolulu> doesn't know that Barack Obama and Obama mean the same entity. Similarly, took birth in and was born in also refer to the same relation. Problem of Open KB canonicalization involves identifying groups of equivalent entities and relations in the KB.  
     
   There are two apparent models for this: CESI (Vashishth et al., 2018) and Galarraga et al., 2014 (IDF).
2. **Lexical Normalization**  
   Lexical normalization is the task of translating/transforming a non-standard text to a standard register.  
   (example: new pix coming in tomoroe / new pictures coming tomorrow)  
     
   Datasets usually consists of tweets, since these naturally contain a fair amount of these phenomena. For lexical normalization, only replacements on the word-level are annotated. Some corpora include annotation for 1-N and N-1 replacements. However, word insertion/deletion and reordering is not part of the task.  
     
   Known models are: MoNoise (van der Goot & van Noord, 2017), Joint POS + Norm in a Viterbi decoding (Li & Liu, 2015), Syllable based (Xu et al., 2015), unLOL (Yang & Eisenstein, 2013)
3. **Named Entity Recognition**Named entity recognition (NER) is the task of tagging entities in text with their corresponding type. Approaches typically use BIO notation, which differentiates the beginning (B) and the inside (I) of entities. O is used for non-entity tokens.
4. **Part-of-Speech Tagging**Part-of-speech tagging (POS tagging) is the task of tagging a word in a text with its part of speech. A part of speech is a category of words with similar grammatical properties. Common English parts of speech are noun, verb, adjective, adverb, pronoun, preposition, conjunction, etc.
5. **Relationship Extraction**Relationship extraction is the task of extracting semantic relationships from a text. Extracted relationships usually occur between two or more entities of a certain type (e.g. Person, Organisation, Location) and fall into a number of semantic categories (e.g. married to, employed by, lives in).  
     
   Example: Elevation Partners, the $1.9 billion private equity group that was founded by Roger McNamee 🡪 (founded\_by, Elevation Partners, Roger\_McNamee).
6. **Sentiment Analysis**Sentiment analysis is the task of classifying the polarity of a given text.  
   One of corpuses which are used for testing this is IMBd dataset which is a binary sentiment analysis dataset consisting of 50,000 reviews from the Internet Movie Database (IMDb) labeled as positive or negative.  
     
   Some of the best models utilize pre-training models, which is also a crucial concept for building highly advanced NLP algorithms.
7. **Text Classification**Text classification is the task of assigning a sentence or document an appropriate category. The categories depend on the chosen dataset and can range from topics.  
     
   It could be said that sentiment analysis is a part of this text classification study.
8. **Word Sense Disambiguation**The task of Word Sense Disambiguation (WSD) consists of associating words in context with their most suitable entry in a pre-defined sense inventory. The de-facto sense inventory for English in WSD is WordNet. For example, given the word “mouse” and the following sentence:  
     
   “A mouse consists of an object held in one's hand, with one or more buttons.”  
   We would assign “mouse” with its electronic device sense (the 4th sense in the WordNet sense inventory).

As shown above, the field of NLP is not a one-size fits all sort of blackbox artificial intelligence, but rather a group of studies which encompasses the various tasks of language processing. It is usually a group of these tasks which are combined to create a model which would interpret text-documents that relate to investments and finance, and filter them into a type of indication or sentiment that could assist investment decision making.  
  
For instance, “Lexical Normalization”, “Part-of-Speech Tagging” are studies which are used to refine the text-information to enhance further meaning extraction. “Information Extraction”, “Named Entity Recognition”, “Relationship Extraction”, and “Word Sense Disambiguation” are fields that can improve accuracy to find the correct subject relation within the text and the proper meanings behind expressions used. Lastly, “Text Classification” and “Sentiment Analysis” tasks will translate text into targeted information that holds informational value about the subject written within the context. This informational value may hold crucial information about the future investment return.

**Transfer Learning in NLP**

1. Background

In the last couple of years we’ve started to see deep learning making significant inroads into areas where computers have previously seen limited success. Rather than requiring a set of fixed rules that are defined by the programmer, deep learning uses neural networks that learn rich non-linear relationships directly from data. Most notable is the success of deep learning in computer vision, as seen for example in the rapid progress in image classification in the Imagenet competition.



Deep learning has also seen some success in NLP, for example in automatic translation. A common feature of successful NLP tasks is that large amounts of labeled data are available for training a model. However, until now such applications were limited to those institutions that were able to collect and label huge datasets and had the computational resources to process them on a cluster of computers for a long time.

One particulaar area that is still challenging with deep learning for NLP, curiously enough, is the exact area where it’s been most successful in computer vision: classification. This refers to any problem where your goal is to categorize things (such as images, or documents) into groups (such as images of cats vs dogs, or reviews that are positive vs negative, and so forth). In NLP, current approaches are good at identifying, for instance, when a movie review is positive or negative, a problem known as sentiment analysis. Models struggle, however, as soon as things get more ambiguous, as often there is not enough labeled data to learn from.

1. How does Transfer Learning Improve NLP

Transfer learning facilitates those that don’t have masses of data and computational resources to deal with some specific NLP problems.

**Transfer learning** refers to the use of a model that has been trained to solve one problem (such as classifying images from Imagenet) as the basis to solve some other somewhat similar problem.

Because the fine-tuned model doesn’t have to learn from scratch, it can generally reach higher accuracy with much less data and computation time than models that don’t use transfer learning.

Simple use-case of transfer learning:

* + - * Using a single layer of weights, in order words, word2vec embeddings from Google to conduct an NLP study.

However, full neural networks in practice contain many layers, so only using transfer learning for a single layer was clearly just scratching the surface of what’s possible.

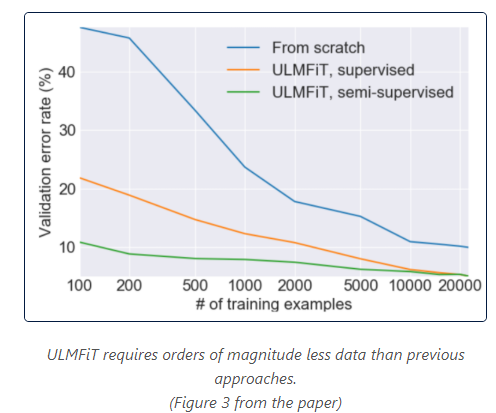
1. Transfer Learning Application by Fast.ai

A project from fast.ai researchers has released a paper on Universal Language Model Fine-tuning for Text Classification (ULMFiT), pre-trained models, and full source code in the Python programming language. It’s paper has been peer-reviewed and accepted for presentation at the Annual Meeting of the Association for Computational Linguistics (ACL 2018).

It is a transfer learning method that dramatically improves over previous approaches to text classification, and the code and pre-trained models allow anyone to leverage this new approach to better solve problems such as:

* + - * Finding documents relevant to a legal case;
      * Identifying spam, bots, and offensive comments;
      * Classifying positive and negative reviews of a product;
      * Grouping articles by political orientation;

The result has shown remarkable improvements without the masses of data and computational resources that might be needed otherwise.



Paper is found in <https://arxiv.org/abs/1801.06146>

As can be said from studies above, with a special condition as in KIC, conducting NLP strategies is a challenging task when considering the small size and limited data and resources. Without massive datasets, KIC is likely unable to train an NLP model from scratch by lack of testing data and access limitation to cloud computation.

For this reason, although applying deep learning for investment strategy might still be too advanced at this stage (for creating a non-biased, non-overfit model), when we do so, transfer learning seems to be a good starting point.

**Transfer Learning in NLP**

* Python as a choice of language for data science has its benefit for being dynamic, interpreted, and flexible. These traits coincidently allow for faster development of ideas and research, which is why it is currently the most utilized language for data science.  
  However, Python is also slow compared to static languages in its native form, especially for intensive numerical operations.  
  These are some optimizing strategies to implement for speeding up numerical operations.