Machine Learning Engineer Capstone Project Report

Project Overview

Data Science as a field is relatively young, with new discoveries being made and new techniques being developed at a breakneck pace. With this comes a variety of specialist roles flooding the job market as businesses try to find the very best talent to create cutting edge solutions.

Data Scientist who utilize their skills in both technology and social science to find trends and manage data;
Data Engineers who prepare the "big data" infrastructure to be analyzed by Data Scientists;
Machine Learning Engineers responsible for taking theoretical data science models and helping scale them out to production-level models.

Each of these roles comes with it's own responsibilities and prerequisites, but there is a fair amount of overlap in their descriptions. Machine Learning Engineers and Data Engineers are often described as sitting "between Software Engineering and Data Science". Data Scientists are often expected to carry out the roles of a Data Engineer, especially in start-ups who can't afford separate specialists.

These are simply the three most common categories. It's not unusual to see roles like Software Engineer with ML, Deep Learning Engineer, Insights Analyst and so on when looking at roles in the field. Some even predict these roles will evolve or be eliminated in the near future. How is anyone supposed to keep everything straight in this fast moving field?

Problem Statement

As a jobseeker, it can be confusing to know which roles are most relevant or suitable to ones given skillset, and the process of sifting through job adverts to find the right roles to dedicate time to can be tedious.
There are several issues that contribute to this: many businesses aren't clear on their own requirements and just using buzzwords to try to attract talented individuals with false promises; search engines are far from perfect, and it's not unusual for a search of "Data Science" roles to come up with more general but unrelated institutional research and scientific roles; position titles alone can be misleading, making brute force methods such as string pattern matching less effective.
Wouldn't it be nice if Data Science and Machine Learning, the source of this confusion, could help tackle this problem?

Solution Statement

Using NLP techniques developed earlier in the course, namely the LSTM model, the project aims to determine the relevance of a job advert based on the job description provided. This comes in two parts, a binary classifier to determine if

a role falls under the Data Science umbrella (I choose 5 classes: Data Science/Machine Learning Engineer/Software Engineering/Data Analyst/Data Engineer) or not, and a multiclass predictor to categorise the roles that fall under the umbrella. While you can think of the first leading into the second, each part is treated as independent in training for simplicity in interpreting the results.

As part of my own work, I am developing a webspider to crawl job sites for Data Science positions. However, the spider is out of scope for this project, and a pre-existing dataset from Kaggle will be used instead.

This project is merely a proof of concept to better understand the challenges involved. The expectation is that, in the unlikely event that the NLP model doesn't have perfect 100% accuracy on the first try, the process should reveal weak points for when my own crawler is complete. For example, perhaps there are additional features missing from the Kaggle dataset that could improve the model's performance, and this could be included in my own crawler.

The Dataset

The dataset comes from kaggle user Shanshan Lu who scraped 7000 US job listings from Indeed.com.
The following fields are found in the dataset:

- Position - the position title/high level description
- Company Name
- Description - detailed job description and requirements
- Reviews - the number of reviews of the company, generally an indication of company size.
- Location For the purpose of this project, only Position and Description will be used to build the model. Positions will be used to categorise the descriptions as relevant or not relevant, as well as a breakdown into and prediction of specific roles.

Metrics

There are several options in the choice of performance metrics of such a classification problem. Accuracy is the most obvious, however others such as precision or recall can be useful depending on the nature of the data or the specific constraints on the problem.

In this particular case, there is no asymmetry in the desired outcomes of the project (i.e. false negatives are no worse than false positives) so accuracy seems to be a sensible suggestion.

A problem stems from an imbalance in the dataset; for both the single class and multiclass case we see a class with a clear majority, making accuracy a worse choice of metric. There are two options at this fork in the road - Choose a metric more resistant to an imbalance dataset, such as precision and/or recall, or resample or augment the data to solve the imbalance. Instead of

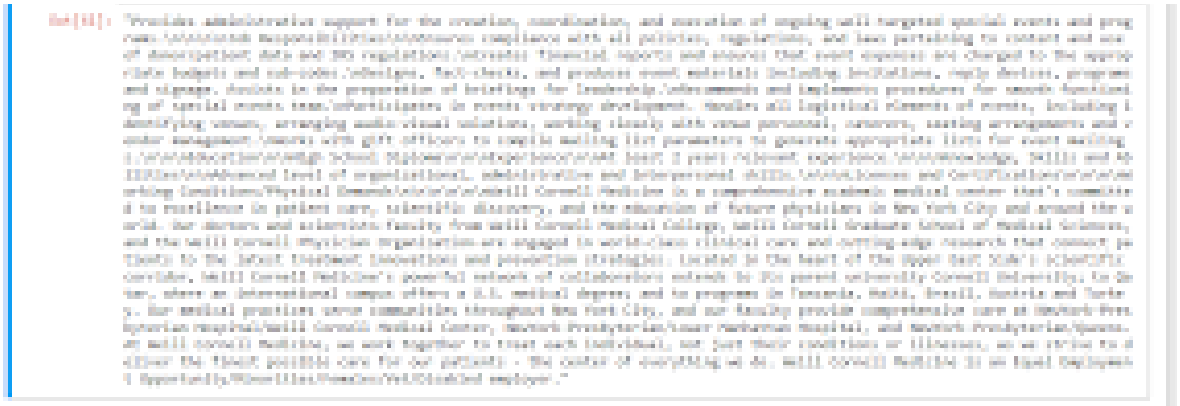
abandoning accuracy, I decide to resample and reshuffle the data to clean up this imbalance, sacrificing some data points to preserve the validity of accuracy.

In order to test the significance of the model relative to the benchmark, I apply McNemar's test (https://machinelearningmastery.com/mcnemars-test-for-machine-learning/) rather than a t-test. This paper by Dietteich (http://ieeexplore.ieee.org/document/6790639/) has a great comparison of the methods, and informed my decision to use McNemar's test.
McNemar's involves building a "Contingency table" that tabulates the outcomes of the two models on description, and calculates the relevant statistics. This is conveniently available through the Statsmodel library. Part of the justification for using this method was that, as we are utilising AWS Sagemaker, I didn't want to incur the extra costs (both time and money) of retraining the model for cross validation.

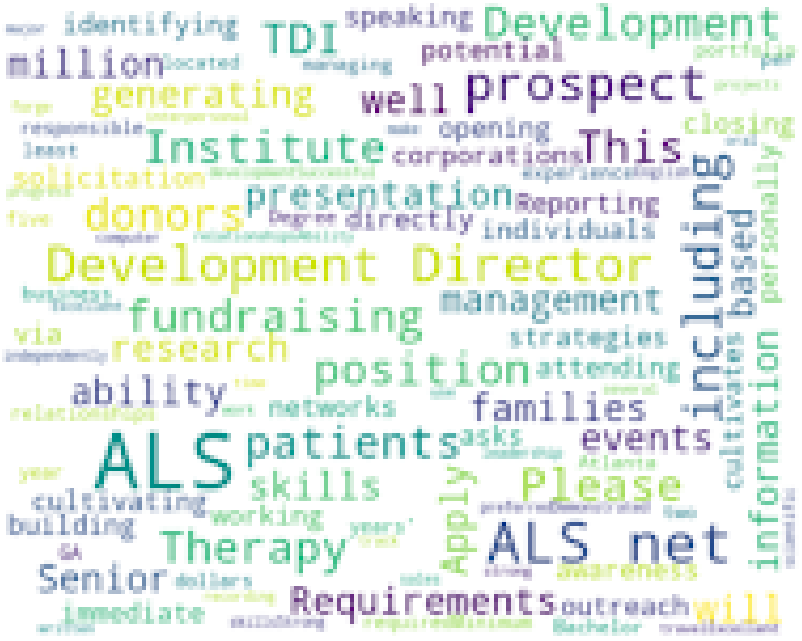
Data Exploration and Visualisation

Data Exploration is only carried out in the binary classifier notebook as it would be redundant to include it elsewhere.

I begin by inspecting the raw descriptions themselves in order to determine any textual artifacts that may need to be handled in the preprocessing steps, as well as to try to build an intuition for the common words or phrases that we hope our algorithm can learn to pick up on. Here we see new line escape characters that need to be handled as well as other punctuation that could confuse our model.

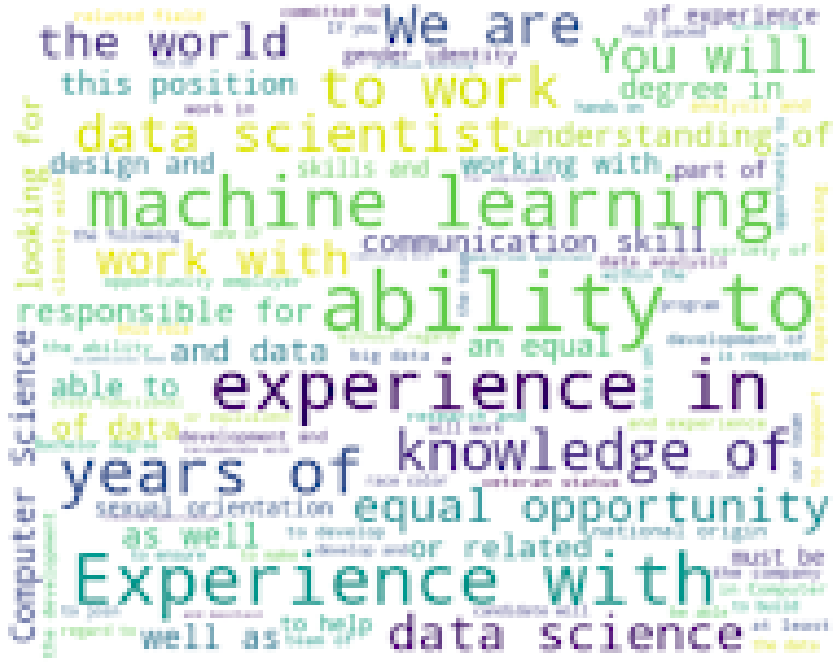


I then generate a word cloud to once again try to spot any patterns in the frequency of key phrases, both for single descriptions and across the whole dataset. First we see the case for a single description. The most common phrases see, to be quite company and role specific. It might be hard to see in this but there are also words that appear to be malformed and concatenated, such as "experienceEnglish" and "requiredMinimum".



When looking across all descriptions, we see more generic phrases. Some are likely useful (we want the model to look for the phrase Data Science or Machine Learning obviously) whereas others are likely found across all roles (experience, understanding) so would be less useful for predictive purposes.

Something I hadn't considered is that many descriptions contain phrases that bear no relevance to the job at hand. For example sexual orientation appears reasonably often as adverts want to emphasise inclusivity (or more likely just want to meet HR guidelines). Future iterations may want to intentionally remove such phrases as they offer no additional information to solve this problem.



Another thing to notice is that formatting slips into the word clouds, so these initial attempts to visualise the data contain malformed words. We see that this is solved after the preprocessing steps to clean the data.

Though we lose some detail in the stemming process, this final visual clearly shows useful patterns in phrase frequency that we expect our model to utilise in it's predictions.

The Algorithm

The algorithm of choice for this project was the LSTM, first introduced in a paper by Hochreiter and Schmidhuber (https://www.bioinf.jku.at/publications/older/2604.pdf).

As a brief history, Natural Language Processing had a revolution in the 1980's when the old 'handwritten' rule based models were succeeded by the statistical approach of machine learning. Recurrent Neural Networks leveraged the power of deep learning whilst maintaining the sequential/temporal nature of language.

The LSTM was introduced in 1997, with it's recurrent unit able to handle long-term dependencies found in speech. They became the cutting edge approach in the 2010s with the rise of voice and text processing (think Siri and Google's Voice Assistant).

The LSTM has since been succeeded by Transformer based models after the Google brain team (Vaswani et al) released their research on the Attention Mechanism (https://arxiv.org/pdf/1706.03762.pdf) in 2017. Transformers learn by attending to a sequence as a whole, rather than each word sequentially as in an LSTM. This means fewer backprop steps and a generally faster learning speed. Transformers can leverage the parallelism inherent to GPU computing, which LSTMs cannot (due to their sequential nature). Finally, Transformers can successfully carry out transfer learning, leading to cutting edge models such as BERT being available to everyone and anyone.

So why not use Transformers for this task? There are two main reasons:

- Transformer models such as BERT are particularly good for sequence-to-sequence tasks. While this project could be framed in such a way, (ALERT: opinion incoming) this word vector bag-of-words classification method is a simpler approach.
- I have more experience with LSTMs, so chose the model I am most comfortable with. I'm certain to try Transformer models for future versions of this project.

Benchmark

For the purposes of benchmarking, a Scikit-learn Dummy Classifier is used as a 'worst case scenario'. The intention is to create a classifier that is only slightly better than random chance (if that) to provide an output for which we can determine the statistical significance of the NLP model. While the classifier gives many options for the dummy classification method, in testing it was found that all provided similar(ly bad) results i.e. close to random as one might expect.
I particularly like this method of benchmarking as it is explainable, AKA a white box model. For example, if using the most frequent class, we can clearly understand why the benchmark accuracy is what it is by looking at the data's distribution.

Though perhaps slightly unnecessary, I also test a completely untrained LSTM as a secondary benchmark. Unsurprisingly, this provides a seemingly random result similar to that of the Dummy classifier. This provides no additional value, but is included for flavour and a hint to the development process.

Methodology

Firstly, I categories the data based off of the "position" feature from the data. For the binary classifier, this is a simple string pattern matching step to look for words such as "data", "engineering" or "learning" in the position. For the multiclass case, I carry out a regex search for the exact roles "Machine Learning", "Data Scientist" e.t.c and enumerate each class.

It's important to note that there is an intrinsic ordering to this process. Searching for "Machine Learning" before "Data Scientist" means a role listed as

"Data Scientist with Machine Learning" would be classed under Machine Learning.

Next I preprocess the relevant descriptions (all for the binary case, only those with relevant roles in the multiclass case). I then perform a train/test/validation split on the data. I've included a validation set here to diagnose issues further down the line.

Each set is converted to a bag-of-words model, which involves: removing non-alphanumeric characters; tokenising the descriptions into words; stemming words, for example communication and communicate would become commun; and removing stopwords which are common words that add no useful information that our model can learn from, such as "and", "the", or "is".

A vocabulary is built from the training bag-of-words by selecting the top 5000 most commonly occuring words, sorted by frequency. This will ultimately be used to create word vectors for each job description, a representation that encodes the words by their index in the vocabulary (or some default for infrequent words). The idea is that descriptions of similar content will have similarly aligned vectors, and our model can learn to identify these alignments to categorise novel descriptions.

We finally create our word vectors and pad with zeros to maintain a fixed length, as our LSTM is not configured for variable input lengths.

With our preprocessing complete, and our word vectors and vocabulary built, we can define our LSTM model.

Some key differences to note:
The LSTM for the multiclass case has 5 outputs, unlike the 1 for the binary classifier.

There is also no activation function used after the linear layer in the multiclass case, whereas a sigmoid activation is used for the binary model.

Both models run the same training loop, however the binary model uses BCELoss whereas the multiclass model uses regular Cross Entropy Loss. This explains why the multiclass needed no activation on the final layer, as Cross Entropy applied a softmax function in its process.

Results

It's no surprise our benchmark results were ~50% for the binary case and ~20% for the 5 class case, as that is roughly random. The same can be said for our untrained LSTM as it is essentially randomly guessing.

The training accuracy for both models were 90+%, but it is too soon to pop the champagne.

In the binary case, we saw unstable improvements over the benchmark. The test on my personal machine yielded an accuracy of 56% on the test set, barely achieving our cutoff for statistical significance. The run on the Sagemaker instance however yielded an accuracy of 70%, learning to a p value of essential 0 (correct to 5 d.p.). In either case, the lower test accuracy coupled with an increasing validation loss over the training steps clearly shows the model is overfitting and could certainly benefit from regularisation in future iterations.

In the multiclass case, we see consistently vast improvements of ~2x over the benchmark, yielding accuracies consistently around 40%. While a major improvement, this means the model is unfortunately wrong more often than not. This is part of the reason why I wanted to focus on accuracy. For my purposes, I'd want to set a hard minimum success rate of 50%, if not significantly higher, if I were to use this to actually run my job hunt process.

To validate the success of the models, I utilise McNemar's test. In both cases, we see statistically significant improvements. In the binary case, the p value can fluctuate between 0.05, the limit of the threshold set, and ~10e-12. In the multiclass case, we consistently see p values ~1-e-6 or better.

Conclusion

As stated at the outset of this project, the aim was never to create the perfect machine learning model. This was merely a proof of concept, and a chance to throw ideas at a wall to see what sticks. It is clear from the results that both models can achieve statistically significant improvements. It's also clear that these improvements, though statistically significant, are too low to be practically useful, in their current forms.

Some improvements have been mentioned but it is worth elaborating the steps I plan to use to take this project forward:

1) Include other surrounding data to be fed into the dense layer of the network, such as salary expectations or company details.
2) Improve the list of stopwords in the context of job descriptions
3) Add regularisation or dropout to fix the overfitting problem seen in the validation.
4) Determine the source of the instability in the binary classifier.
5) Use cross validation to better estimate the model accuracy.

6) Complete the web crawler to be able to gather more data in a cleaner manner.

Though our final accuracies aren't the highest, as far as a single iteration in the data science prototyping process goes I'd class this as a success.