Dimensionality Reduction & Feature Selection

Rasmus Nisted Velling
Machine Learning Engineer Nanodegree
Capstone Project
Udacity

July 8, 2019

Abstract

This is the paper's abstract ...

Contents

1 Definition **2**
1.1 Background 2
1.2 Problem Overview 3
1.3 Problem Statement 4
1.4 Metrics 5

2 Data Exploration **5**
2.1 Datasets 5
2.2 Distribution of target variables 6
2.3 Distribution of input variables 6

3 Methodology and Implementation **7**
3.1 Technical implementation, overview 8
3.2 Preprocessing: Dimensionality Reduction Techniques 9
3.3 Model Frameworks & Algorithms 11
3.4 Prediction 12
3.5 Benchmark 13

4 Results **13**
4.1 Dimensionality reduction 13
4.2 Model Frameworks 15

5 Conclusion **16**

1 Definition

1.1 Background

Dimensionality reduction and feature selection

Dimensionality reduction and feature selection is at the heart of applied statistics and data science. In it's essence, *dimensionality reduction* is the practice of taking p available features (also known as dimensions) and reducing them to $p_{reduced} < p$, so that we end up with fewer features than before, thus, if done right, simplifying the problem at hand.

Dimensionality reduction can be done in many ways [Wikipedia, 2019a], and *feature selection* is, in short, one particular way of doing so, where one 'cherry picks' a subset of features based on some criteria, often a statistical scoring like the Akaike information criterion or likewise [Wikipedia, 2019b].

Among other types of dimensionality reduction methods commonly used are Principal Component Analysis (PCA) and Random Projections, which both are transformations of the underlying data, so that the new set of features are linear combinations of the original inputs. This has both advantages, in that new, more informationally rich features can arise, and that the new dataset, with fewer features is easier to process; and disadvantages, in that the blurring of underlying features can make interpratability difficult and possible loss of information, if not done carefully.

There can be practical as well as scientific motivations for doing dimensionality reduction in general and/or feature selection in particular.

Dimensionality reduction in general, especially when dealing with hundreds, thousands, or more, features is often born out of a shear practical motivation of wanting to speed up computational processing time or making data easier to comprehend and vizualize.

Feature selection in particular, when done right, will also yield the benefits of quicker processing of the final dataset and easier comprehension of the total data, but has the additional benefits of providing very concrete information about what particular inputs are of (statistical) importance to the problem at hand.

Making sense of mess

In commercial as well as academic settings, it is often common to face statistical problems, where an abundance of features are given, without any clear indication of which are important, and which are not. ‘Throwing everything at the model’, will result in something that takes hours, maybe days, to process, and possibly without any usefull output. Thus the first problem at hand becomes that of ‘boiling down’ the data to something that can be processed and will deliver explanatory value [Hofner et al., 2014].

And even if we could process all available inputs with decent speed and quality, a lot of times supplying insights to a business or academic research also necessitates providing interpretability of some sorts and not just a ‘black box’. So eventually we will also want to strive for the ‘most simple explanation’ that still fits data reasonably well [Wikipedia, 2019d].

At the same time in many realworld settings, especially in businesses, a limited number of data scientist and analyst are expected to solve problems across many domains like marketing and sales, operations, logistics, infrastructure, HR and many more, while at the same time being required to deliver results fairly quickly and of a decent standard. In these types of settings flexibility, genericity, and speed, become important model and framework aspects, as the analysis and data science teams will want to deliver scalable and easily maintainable solutions to the problems posed by the surrounding business.

Thus it is of interest to explore models and frameworks which can help us deal with both feature selection and/or dimensionality reduction in semi-automatic ways, as this will yield better models, speed up processing and save us precious time at work.

NIPS 2003 Feature Selection Challenge

The NIPS 2003 Feature Selection Challenge [NIPS, 2003b] mimicked the above-mentioned requirements into one competition, where participating teams were asked to come up with one model, or framework, which was then implemented across five datasets, sourced from different academic domains. The final score and ranking was then calculated as the average of the *Balanced Error Rate* (defined in section **??**) on each dataset, thus, requiring the teams to come up with flexible and generic models, that where somehow able to distinguish useable features from non-useable, or at least able to extract value from an abundance of supplied inputs - many of which were intentionally created noise.

1.2 Problem Overview

Inspired by a specific, high dimensional, sparse data problem at work, I wanted to investigate flexible and broad methods for dealing with high dimensional, sparse classification problems. And while researching on this topic, I stumbled across the

NIPS 2003 Challenge, which I in many ways feel reflects a lot of the problems I, as a Data Analyst, deal with in my day-to-day work.

In this Capstone Project I will explore model methods and frameworks inspired by the NIPS 2003 Feature Selection Challenge applied to the five datasets made available for the same challenge. A key component of the challenge was that the teams were scored on how well a *single model type* would perform across all five datasets, thus requiring model frameworks to be flexible and generic.

Winning entries of the Challenge are described in the book Feature Extraction: Foundations and Applications [Guyon et al., 2006] which came to life as a product of the best entries in the NIPS 2003 Challenge. Many of the winning entries are coded in Matlab or C, so this in this Capstone project I will focus on exploring how close one can come to the winning entries using readily available and popular tools in Python.

1.3 Problem Statement

Problem

- How do different dimensionality reduction and feature selection methods compare in terms of processing time and final prediction performance?
- How well do easy python-implementations, using readily available tools like sklearn and XGBoost, perform on the NIPS 2003 Challenge datasets?

Solution

- I will implement a series of dimensionality reduction methods and classification models in python across the five datasets from the NIPS 2003 Feature Selection Challenge.
- These models will be inspired by the winning entries from the NIPS 2003 Feature Selection Challenge, described in [Guyon et al., 2006], as well as more recent, popular models like XGBOOST [Chen and Guestrin, 2016], which has enjoyed a lot of success in challenges at kaggle.
- I will apply the same models to all five datasets (as done in the original NIPS 2003 Challenge), to test the genericity and flexibility of the proposed solutions.
- The models will be ranked on their *Balanced Error Rate* score on the official validation datasets.

1.4 Metrics

The primary scoring metric will be the *Balanced Error Rate* as used in the original NIPS 2003 Challenge [NIPS, 2003a]. The BER takes the unbalanced nature of one of the datasets into account, and also makes it easier to compare my own results with those of the wining entries.

The BER is defined using the classification *confusion matrix* shown in Table 1, and plugging the relevant numbers, a, b, c and d, into the equation shown in (1).

		Prediction	
		Class -1	Class $+1$
Truth	Class -1	a	b
	Class $+1$	c	d

Table 1: Confusion matrix

$$BER = \frac{1}{2} * \left(\frac{b}{a+b} + \frac{c}{c+d} \right) \tag{1}$$

2 Data Exploration

2.1 Datasets

The NIPS 2003 Feature Selection Challenge had five datasets. Originally the participants where given no background information about the data, and no column labels where given either, as not to bias the selection / reduction process with the participating teams. Additionally, teams were told, that all datasets had ‘false’ features (called ‘probes’ by the organizers) added, and the order of the features were randomized to make it more difficult to extract ‘good’ features.

So the only available info, was the apparent info: all inputs had been scaled to integers between 0 and 999 for four of the five dataset, and inputs where binary for the 5th dataset. And target variables where classified as either -1 or 1, making all the modelling problems binary classification type problems.

The five datasets vary widely in number of features and number of observations. The five datasets are:

ARCENE Non sparse. Features: 10 000. Observations, training data: 100. Observations, validation data: 100. Arcene is based on mass-spectrometry data, and the underlying task is to separate cancer-patients from healthy patients.

DEXTER Sparse Integer. Features: 20 000. Observations, training data: 300. Observations, validation data: 300. Dexter is based on Thorsten Joachims in the “bag-of-words” representation. The underlying task is to separate which Reuters articles are about ‘corporate acquisitions’ and which are not.

DOROTHEA Sparse binary. Features: 100 000. Observations, training data: 800. Observations, validation data: 350. Dorothea is based on a drug-discovery dataset, and the task is to classify molecules as active (binding to thrombin) or inactive.

GISETTE Non sparse. Features: 5000. Observations, training data: 6000. Observations, validation data: 1000. Gisette is based on a handwritten digit dataset. The task is to separate 4 from 9.

MADELON Non sparse. Features: 500. Observations, training data: 2000. Observations, validation data: 600. Madelon is an artificially created dataset. The set contains 32 clusters of points based the vertices of a five dimensional hypercube and randomly labeled 1 or -1.

2.2 Distribution of target variables

Figure 1 shows how the target variable is distributed across the five datasets. We see that Dexter, Gisette and Madelon are perfectly balanced. Arcene is not perfectly balanced, but only moderately unbalanced. The Dorothea dataset is quite unbalanced with only 78 of 800 observations being of class +1.

2.3 Distribution of input variables

Figure 2 shows histograms across columns in input data for the five dataset. We see different kinds of distributions. Madelon, the artificial dataset, looks to be quite normally distributed in terms of input values per column.

The non-artificial datasets all have a lot of zero-values, especially Dexter and Dorothea. Note also that Dorothea has only zeros or ones, thus the ‘histogram’ is illustrated as a series of bar charts (one per column).

We see the the inputs generally fall within the same kind of distiribution within the datasets. This means that artificial feautres, probes, where created to mimick the real inputs, and thus hard to seperate out using simple metrics as variance, etc.

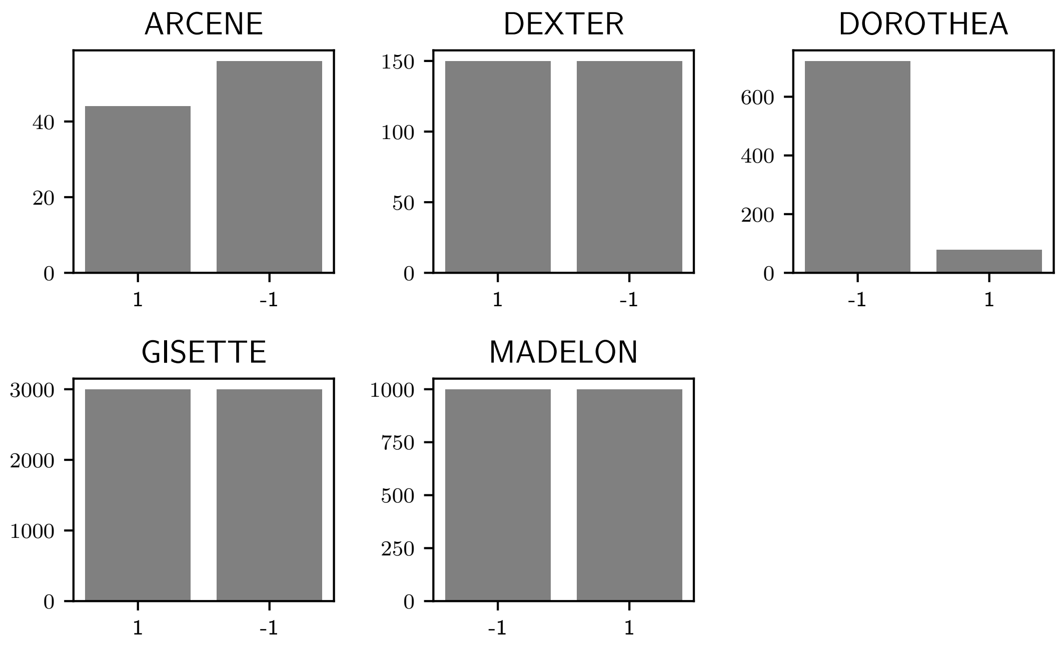

Figure 1: Distribution of target variable in datasets

3 Methodology and Implementation

The overall purpose of this capstone is to explore how different dimensionality reduction techniques and model frameworks perform against one another - and against a set of benchmarks.

To summarize briefly, I will compare four different reduction techniques; PCA, Univariate Feature Selection and two Random Projections techniques; each reducing the input dataset to seven different sizes (5, 10, 15, 25, 35, 50 and 75 dimensions respectively).

On top of this I will apply four different model frameworks; Logistic Regression, Linear Support Vector Classification, LightGBM and XGBoost.

This yields a total of 560 different combinations to process. This was processed in 3 stages:

1. Dimensionality Reduction
2. Model Fitting
3. Prediction and Scoring

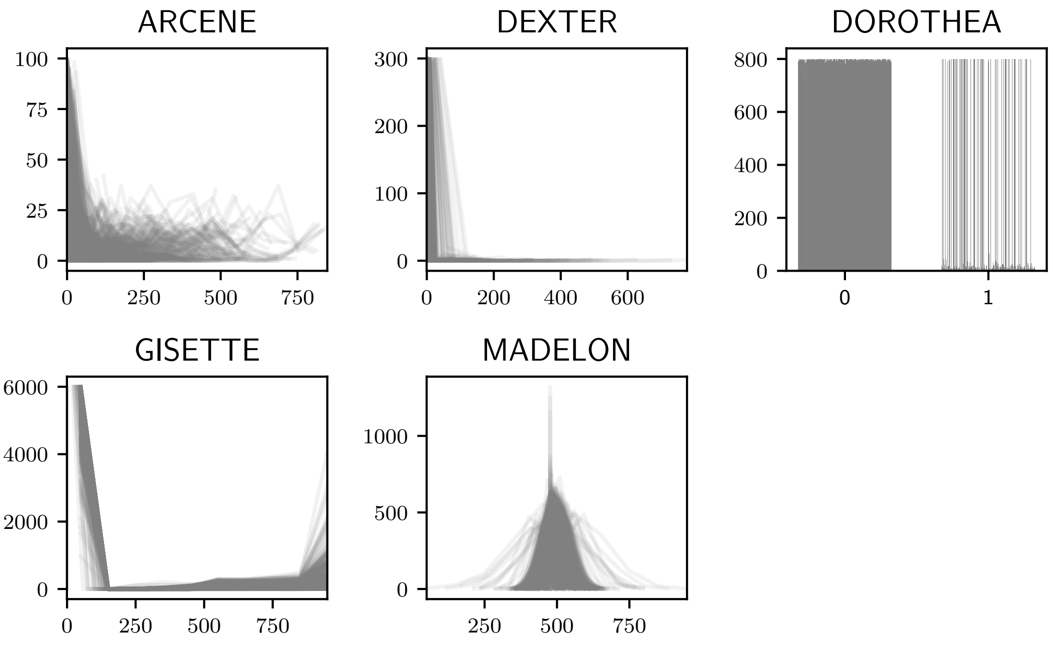

Figure 2: Histograms of input variable in datasets

Each step will be explained in detail after the technical overview in sections 3.2, 3.3 and 3.4 respectively.

3.1 Technical implementation, overview

Step 1 would yield 140 unique (reduced) datasets. I would produce all these and cache them as serialized objects, using the python package called *pickle*, before proceeding to step 2. Producing all 140 dataset took about 8.5 CPU hours. The processing was run in parallel across 4 cores on a modest 2.4 GHz machine, so the end-user waiting time for this step was a bit over 2.5 hours.

Step 2 would take the 140 reduced datasets and fit the four model frameworks in turn. This step was also parallelized across the 4 cores, on the same machine. Although logistic regression, LightGBM and XGBoost would process reasonably fast, with runtimes in magnitude of seconds, the linear support vector classifier would sometimes take extremely long, with runtimes up to the magnitude of hours, though some would fit in a couple of seconds. Support vector classification algorithms where cited quite a few times in [Guyon et al., 2006] as a high performing model frameworks, and thus I initially implemented a polynomial support vector classification and well as one using a radial basis function kernal, but these would also time out or not converge within reasonable time.

Step 2 without the linear SVC would take only a few minutes. I was not patient enough to wait for all SVC implementations to run. 16 hours of waiting time with the machine running on all cores (so that is 64 CPU hours), would yield 40 of the 140 SVC implementations (chosen in random order). I have included results for the SVC's that ran, when relevant, but generally not.

At the end of step 2, each fitted model was cached as a serialized object, also using *pickle*.

In step 3 I would load the fitted models, and predict on the validation set. These predictions were then used to score the *Balanced Error Rate*, the *F1-score* as well as the *area under the ROC-curve*.

3.2 Preprocessing: Dimensionality Reduction Techniques

No data-cleaning, like missing values handling, or similar procedures were needed, as the datasets came pretty well cleaned and structured from the competition website. The data files were delivered as text files, and some minor procedures were needed to load these properly, especially the sparse data sets, as these were delivered in a coordinate-type format.

I implemented normalizing along with the PCA decomposition, in the hope that tis would help the spport vector classification run faster, but nothing noteworthy happened, and thus I did not normalize the input along with the univariate feture selection or the random projections.

Thus I eventually implemented the following four dimensionality reduction methods:

PCA

PCA is one of the most well-known dimensionality reduction techniques. However it was still a vital part of the winning entry to the NIPS 2003 Feature Selection Challenge, where it was used by Radford M. Neal and Jianguo Zhang - along side a pretty complax implemention of Bayesian Neural Networks and Dirichlet Diffusion Trees, [Guyon et al., 2006], Chapter 10.

In this capstone project I have implemented SciKit Learns PCA with a normalization step preceeding. For performance comparison I am subsetting to 5, 10, 15, 25, 35, 50 and 75 dimensions respectively on all dataset - no matter the original amount of features.

The PCA will have the benefit that the components will be linear combinations of multiple inputs, and thus we hope to be able to 'capture' a lot of information in relatively few features. This might be a benefit on the Arcene dataset which we know has about 1500 original inputs (and the about 8500 false probes).

The PCA implementation is shown below in Listing 1.

Listing 1: PCA Implementation

```
def reducer_pca(data, params):
    # if no params set
    if params is None: params = {'n_components': 5}

    # normalization, pca pipeline
    pipeline = Pipeline(
        [('scaling', StandardScaler()),
        ('pca', PCA(n_components=params['n_components']))])
    pipeline.fit(data['X_train'])

    # output
    do = deepcopy(data)
    do['X_train'] = pipeline.transform(data['X_train'])
    do['X_valid'] = pipeline.transform(data['X_valid'])

    return do
```

On a dataset like madelon, which orginially only had 5 genuine inputs, we would perhaps want to opt for another kind of feature selection, where we can 'cherry pick' the best inputs. Univariate feature selection could possibly such a tool.

Univariate Feature Selection

Univariate Feature Selection was also used in the winning entry by Radford M. Neal and Jianguo Zhang, [Guyon et al., 2006], Chapter 10.

In this capstone I am implementing the *SelectKBest* univariate feature selection function found in SciKi tLearn. I am selecting the 5, 10, 15, 25, 35, 50 and 75 best features respectively.

Which features are the best is being decided by the *mutual_info_classif* scorer function also from SciKit Learn's feature selection module, which ranks the features by the mutual information between a given feature and the target variable.

The upside is ofcourse that we are left with very strong candidates for the best explaining features. The downside of this selection method is, that it can take quite a while to do this process over a given dataset. I found that this way of doing univariate feature selection was between $10x$ and $100x$ slower than PCA and Random Projections.

The univariate feature selection implementation follows that of the PCA, but with the middel pipeline-part being altered as shown in Listing 2. The functions

SelectKBest and *mutual_info_classif* being import from *sklearn.feature_selection.*

Listing 2: Univariate Feature Selection Implementation

```
reducer = SelectKBest(
    mutual_info_classif, k=params['n_components'])
reducer.fit(X, y)
```

Random Projections

Random projections area way of reducing dimensionality on high dimensional datasets. The idea is, that if one is given a set of points in a high-dimensional space, it is possible to roughly maintain the distance between the point while reducing the number of dimensions. This is called the *Johnson-Lindenstrauss lemma*, [Wikipedia, 2019c].

I implemented both *SparseRandomProjection* and *GaussianRandomProject* from SciKit Learn's *sklearn.random_projection* module, in a similar fashion as shown for PCA and Univariate Feature Selection. Both random projections will reduce the dataset to 5, 10, 15, 25, 35, 50 and 75 features respectively.

3.3 Model Frameworks & Algorithms

In general all the implementations used in the final version are out-of-the-box implementations.

I initially did experiment with different hyperparameters for the Support Vector Classification, using both different kernel types and different polynomial degrees. These ended up mostly not running, and these experiments where discarded. For LightGBM and XGBoost, there are plenty of unexplored opportunities for hyperparameter settings. I eventually did not go down this road, as I felt the current 560 different outputs was sufficient, and that hyperparameter exploration for LightGBM and XGBoost respectively was beyond the scope of this capstone.

The purpose of this capstone is to see how far fairly easy implementations can take us in terms of predictive performance, and not to beat the NIPS 2003 Challenge winners.

Logistic Regression

Logistic regression is perhaps the most common and most popular classification algorithm, and with good reason: Logistic regression is, in brief, a linear regression wrapped in a non-linear output function. Thus it has the benefit of being easy to interpret, as it will yield parameter values for each input feature. Logistic regression can often perform quite well, and is usually pretty fast to implement.

I used *SciKit Learn*'s implementation of logistic regression via the *LogisticRegression* function from *sklearn.linear_model* as shown in Listing 3.

Listing 3: Logistic Regression Implementation

```
model = LogisticRegression(
    random_state=0, solver='sag', multi_class='ovr',
    max_iter=100000)
```

LightGBM

LightGBM is a gradient boosting method using tree based learners [Ke et al., 2017], which was built with both speed and accuracy in mind. LightGBM has enjoyed quite a lot of succes in competitions on kaggle along side XGBoost.

For this capstone I used a standard *out-of-the-box* implementation of the *LGBMClassifier* function from the *lightgbm* python package.

XGBoost

XGBoost's clame to fame has been it's part in many winning entries to competitions on kaggle. XGBoost was, like LightGBM, built with speed and accuracy in mind [Chen and Guestrin, 2016].

Like with LightGBM I used a standard implementation of XGBoost, using the *XGBClassifier* from the *xgboost* python package.

SVC - Linear

After trying out implementaions of both SciKit Learns's polynomial as well as radial basis function-based support vector classification functions, I eventually end up implementing only a linear type, as previously mentioned. My final implementation ended up being fairly standard, though I did allow the model quite a few more iterations. The implementation is shown in Listing 4

Listing 4: SVC Linear Implementation

```
model = svm.LinearSVC(max_iter=9000000)
```

3.4 Prediction

All model frameworks follow the SciKit Learn structure, and thus have a *.predict*() method. Going through all the fits, and predicting on the validation was therefor fairly straight forward.

3.5 Benchmark

My suggested benchmark model is logistic regression based on the first five components from the PCA decomposition. Logistic regression and PCA decomposition are both very common and popular statistical tools, both easily implemented and usually with fairly good performance. This cocktail is thus in many ways a good place to start when one is faced with a classification problem in a new domain or with a new, unknown dataset.

The benchmark model's performance against the winning model of the NIPS Challenge is shown in table 2. Guessing on either all -1 or all +1, shown in the table as Naive Guess, willl yield a BER of 0.5 on all five datasets, due to the score being *balanced*.

Dataset BER, validation set	ARC	DEX	DOR	GIS	MAD	Total
NIPS Winner	0.072	0.053	0.055	0.016	0.067	0.053
LogisticReg PCA5	0.306	0.390	0.335	0.158	0.398	0.318
Naive Guess	0.500	0.500	0.500	0.500	0.500	0.500

Table 2: Benchmark model performance vs. NIPS Challenge Winner's score

4 Results

4.1 Dimensionality reduction

Processing Times

Reduction method	Mean time, sec.	Median time, sec.
PCA	33.80	9.41
Random Projections, Gaussian	11.38	8.67
Random Projections, Sparse	11.70	9.31
Univariate, k-best	819.19	268.79

Table 3: Mean and median processing times of reduction methods appliead

The combinations of five datasets, four dimensionality reduction methods and seven amounts of desired component, gives a total of 140 unique reduced datasets. The mean and median processing time per method per dataset is shown in table

Method	BER
Univariate k-best	0.233
PCA	0.241
Random Projection Gaussian	0.382
Random Projection Sparse	0.384

Table 4: Average predictive performance per reduction method

3. As expected, univariate selection was by far the slowest method - it will be interesting to see, however, if this is then able to give us better predictions.

PCA would take longer on some of the dataset than random projections, as can be seen on the high *mean* runtime. The *median* runtime however, was in a magnitude similar to the random projections.

Predictive Performance

Table 4 show predictive performance for each dimensionality reduction method measured as average *BER* across all number of components, all datasets and all model frameworks (Linear SVC excluded).

We suspected the *Univariate Feature Selection* should yield the best performance, as this would test each feature against the target value. Univariate feature selection did turn out to have the best performance, however only by a small margin, and perhaps not enough to justify the additional runtime. If however, the task is to deliver the best prediction, without concerns for processing time, Univariate Feature Selection seems to be the best choice.

PCA however turned out very strong. And when one takes into consideration the very reasonable runtimes, it definitely shows why PCA is such a popular tool for dimensionality reduction.

Both Random Projection methods turned out, on average, to perform worse than our benchmark model shown in 2. The usecase for random projections is probably better on even larger datasets, where calculating PCA can not be done.

For all reduction methods it was generally the case that more dimensions lead to better performance, though the marginal gain was decreasing in number of dimensions. The predict performance measured in average BER for each reduction method against number of dimensions is shown in figure 3

4.2 Model Frameworks

Processing Times

Predictive Performance

Table 5 show average BER per model framework. The BER, full show the average BER when run across all unique reduced datasets. The BER, subset shows the average BER across the subset of reduced datasets which were successfully processed by the linear SVC, as to go an indication of the linear SVC's performance.

In general we find XGBoost and LightGBM to outperform Logistic Regression by a percentage point or two. Not much, but considering these are out-of-the-box implementations, reasonable. They also beat the benchmark model which had a BER-score of 0.318. It is reasonable to expect that with finetuning both frameworks would do considerably better.

The NIPS 2003 Feature Selection Challenge Winner had a BER score of 0.053

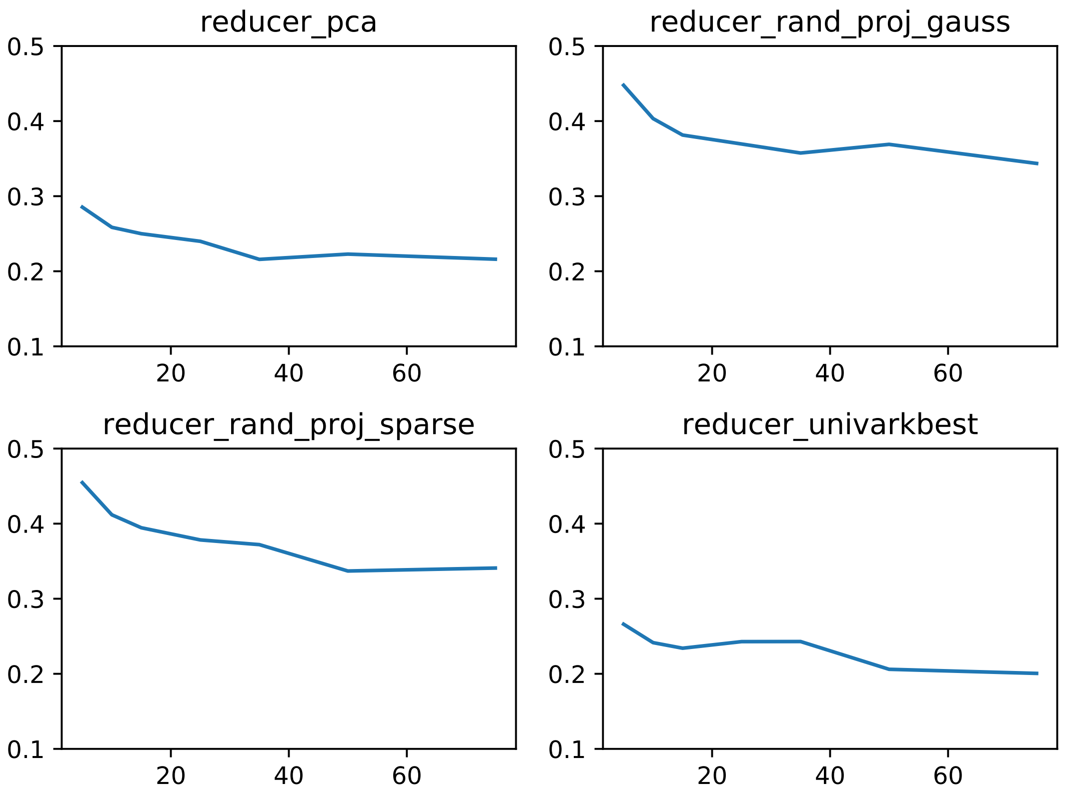

Figure 3: Predictive performance vs n components

model_framework	BER, full	BER, subset
mod_lightgbm	0.301	0.334
mod_xgboost	0.305	0.333
mod_logisticreg	0.324	0.340
mod_svc_lin	N.A.	0.374

Table 5: Average predictive performance per model framework

- so there is still a lot of work to be done if the ambition is to beat this model.

5 Conclusion

Dimensionality reduction

Our first priority was to investigate different methods of dimensionality reduction and features selection.

We saw that PCA is a very strong dimensionality reduction method, that balances both predictive performance as well as computational time. Univariate feature selection yielded the best performance, but at a cost of slow processing times. The two random projection methods were not terrible, but perhaps also better suited for problems where the number of dimensions is even greater than what we've seen in this capstone.

Model frameworks, predictive performance

Our second priority was to investigate how well simple implementations of popular model frameworks would perform compared to our benchmark and to the NIPS 2003 Feature Selection Challenge Winning entry.

With respect to how well simple implementations of popluar models perform, XGBoost and LightGBM did manage to beat our benchmark right out of the box. Tuning hyper parameters would have no doubt brought extra predictive performance to both model frameworks.

We did not see any success worth mentioning with the implementations of SVC. These were ill-suited to deal with the present data.

References

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

[Guyon et al., 2006] Guyon, I., an Masoud Nikravesh, S. G., and Zadeh, L. (2006). Feature selection: Foundations and applications. `http://www.causality.inf.ethz.ch/ciml/FeatureExtractionManuscript.pdf`. [Online; accessed 1-July-2019].

[Hofner et al., 2014] Hofner, B., Mayr, A., Robinzonov, N., and Schmid, M. (2014). Model-based boosting in R: A hands-on tutorial using the R package mboost. `https://cran.r-project.org/web/packages/mboost/vignettes/mboost\_tutorial.pdf`. Online; accessed 1-July-2019.

[Ke et al., 2017] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc.

[NIPS, 2003a] NIPS (2003a). Nips 2003 feature selection challenge: Evaluation. `https://web.archive.org/web/20130503080434/http://www.nipsfsc.ecs.soton.ac.uk/evaluation/`. [Online; accessed 2-July-2019].

[NIPS, 2003b] NIPS (2003b). Nips 2003 feature selection workshop. `http://www.clopinet.com/isabelle/Projects/NIPS2003/`. [Online; accessed 1-July-2019].

[Wikipedia, 2019a] Wikipedia (2019a). Dimensionality reduction. `https://en.wikipedia.org/wiki/Dimensionality_reduction`. [Online; accessed 2-July-2019].

[Wikipedia, 2019b] Wikipedia (2019b). Feature selections. `https://en.wikipedia.org/wiki/Feature_selection`. [Online; accessed 2-July-2019].

[Wikipedia, 2019c] Wikipedia (2019c). Johnson–lindenstrauss lemma. `https://en.wikipedia.org/wiki/Johnson%E2%80%93Lindenstrauss_lemma`. [Online; accessed 2-July-2019].

[Wikipedia, 2019d] Wikipedia (2019d). Occam's razor. `https://en.wikipedia.org/wiki/Occam\%27s_razor#Probability_theory_and_statistics`. [Online; accessed 1-July-2019].