Group M17 - ML 2018

Regression Models Selected

Multivariate Polynomial Linear Regression

One of the models selected for the Kaggle Competition is the **Multivariate Polynomial Linear Regression**. For each row, a polynomial is created using the column values. Hence, a two-dimensional matrix is formed which is then used as an input for Linear Regression.

Experiment Setup

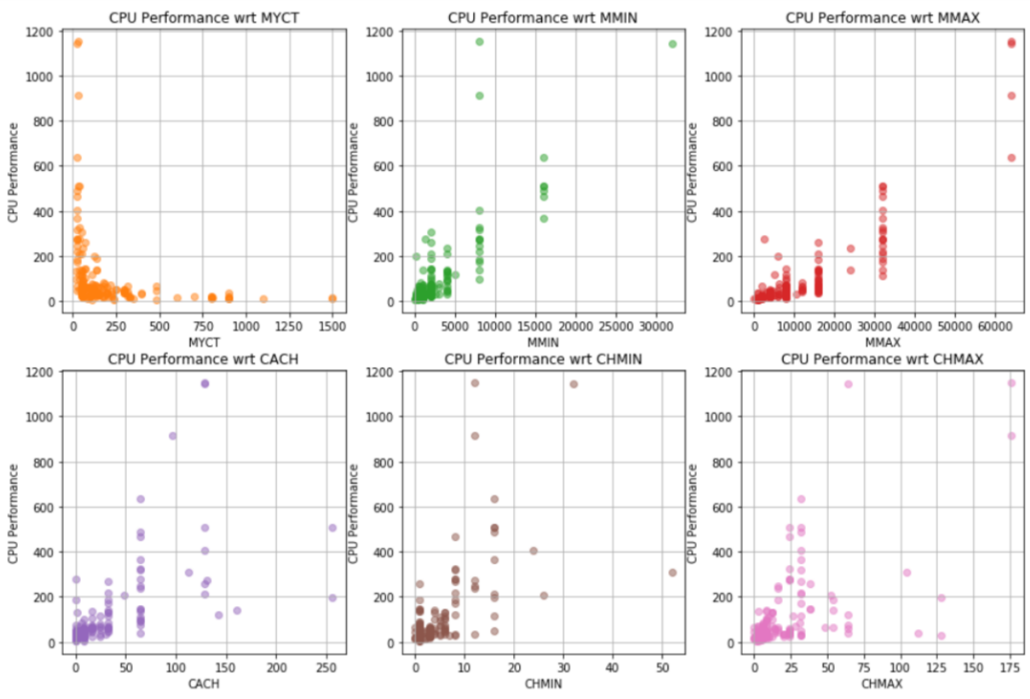
Before selecting the regression model, the following data analysis is done:

- **View data values**: The nature of the data can be seen either using Python or by viewing the CSV file itself.

	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX
0	125	256	6000	256	16	128
1	29	8000	32000	32	8	32
2	29	8000	32000	32	8	32
3	29	8000	32000	32	8	32
4	26	8000	32000	64	8	32

  After importing the data, it can be seen that the dataset is positive integers.

- **Data relationship**: In order to understand the relationship between the computer hardware and its CPU performance, each of the input file columns can be plotted against the output file training data.



As can be seen from the graphs plotted against each computer hardware feature, the relationship is not linear. Hence a higher order polynomial approach is required, so **Multivariate Polynomial Linear Regression** is selected.

To move further with the model experiment, the input data is divided into training and testing data. Out of the total 168 rows given in the training data, 134 are used as training the model and 34 for testing. The python module 'sklearn' provides an inbuilt method, `train_test_split`, to split the data into two subsets. The training subset returned is used to train the model and the test set is used to validate the regression algorithm against and to compare the accuracy of the model.

```
x_train,  x_test,  y_train,  y_test  =  train_test_split(input_x,  input_y,
test_size=0.2, random_state=2018)
```

This training data is then used to create a matrix polynomial of order **2**. The polynomial is created using 'sklearn' pre-processing module called `PolynomialFeatures`. The following code I used to create the polynomial matrix:

```
poly = PolynomialFeatures(degree=2,interaction_only=True)
x_poly = poly.fit_transform(x_train)
```

The fitted matrix now presents with more features than the original is used as an input for the Linear Regression Model. Here `interation_only` is set to **True** to prevent the addition of too many features.

To implement Linear Regression, 'sklearn' provides the `linear_model` module. The `LinearRegression` uses the method of **Least-Squares** to perform regression. The training data is given to the model and the test data is I used to predict the outcome.

```
lr = linear_model.LinearRegression()
lr.fit(x_poly, y_train)
y_pred=lr.predict(x_test_poly)
```

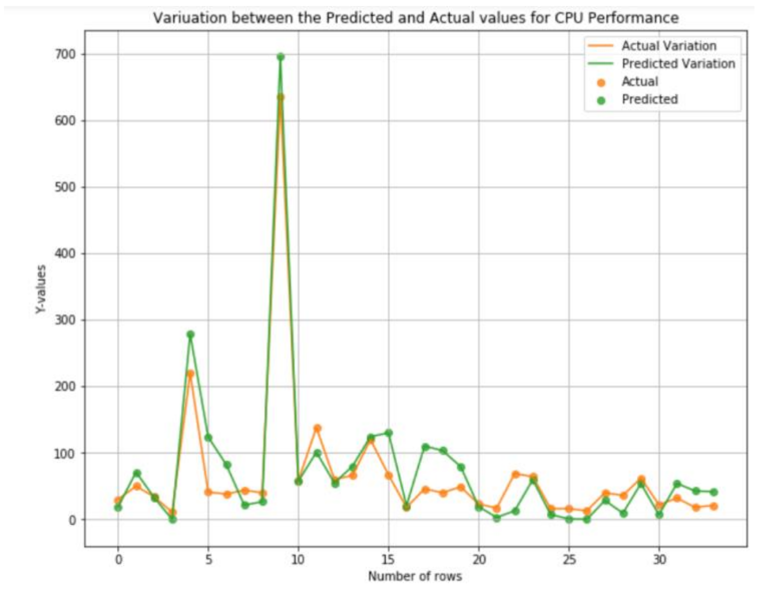
Performance Evaluation

In order to evaluate the performance of the algorithm, two methods were used:

1. **Mean Absolute Error:** The mean absolute error is found for the predicted data generated by the model and is compared with the error generated when getting the mean using the actual data [1]. For this algorithm, using different polynomial degrees and data pre-formatting, the mean is calculated and found to be least for degree two. The average mean getting the best performance is:

   ```
   Model Avg: 25.131501852249592
   Mean  Avg: 79.95063025210084
   ```

   The average error of 25.131 is less than half of the error generated when using the actual data. When the polynomial degree is increased, the Model Avg. value is increased to 27.49861041148047.
2. **Visual Representation:** To check the actual differences between the predicted and the actual values, a visual representation shows the deviations between the points. It shows that the deviation is not that high and the model is a good fit.



Performance Enhancement

To increase the performance of the algorithm, three changes are made to the dataset.

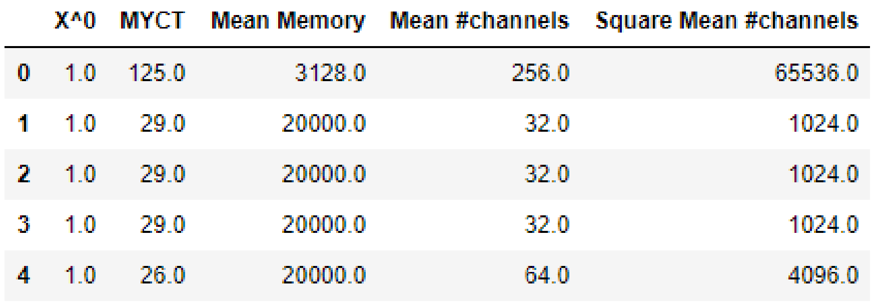
1. **Using Mean:** The columns MMIN, MMAX are combined into a single column which is formed taking the mean of both these columns. A similar approach is taken for the columns CNMIN, CHMAX. This reduces the total number of columns from six to four.
2. **Using column subset:** After reducing some of the columns to their mean, out of the remaining features, only a subset of three are used. This increases the performance significantly.
3. **Removing unnecessary polynomial degrees:** Some columns are removed from the resulting polynomial matrix as they do not contribute towards model fitting. [2] The column having data for 'Ax' is removed and performance is seen to be better.

Bayesian Ridge Regression

A **Linear Bayesian Ridge Regression** model was constructed to fit a prediction model for the performance of 41 CPU's, from a training dataset of 168 CPU performances. The model is based on 6 hardware features of the CPU's, which can reasonably be reduced to 4 by constructing basis functions of the mean Memory size and mean number of I/O channels, from the maximum and minimum of each feature. This reduces the model complexity and chance for overfitting the data. Bayesian regression calculates the posterior distribution proportional to the product of a prior distribution and the likelihood, where the prior is the known distribution form for the parameters in question, and the likelihood is the plausibility of the observation given the model parameters. Therefore, the parameter values are repeatedly updated base on the posterior probability. In Bayesian regression, the likelihood is taken to follow a normal distribution, and hence, though ridge regression, the posterior must also follow a normal distribution. The posterior is the probability of the parameters, given the new data. The prior distribution can be hard to estimate, but Ridge regression, the L2 regularisation technique, estimates a univariate Gaussian distribution for the beta coefficients, helping to prevent overfitting. This is seen to reduce the variance as compared with an Ordinary Least square method. L2 ridge regression penalises the squared loss function by the square of the beta coefficients, whereas L1 regression, known as Lasso regression, estimates a Laplacian prior and penalises the squared loss by the absolute value of the coefficients [3].

Experiment Setup

In this algorithm, the `BayesianRidge` predefined function is called from ‘Scikit’, and a preliminary model is fitted, for a first order linear variables. The variances of each feature in the data are measured and compared to a threshold value of 1e-07. The features with variance under the threshold level are removed, leaving the relevant features as a subset of the original data, thus lowering the probability of overfitting [4]. The ridge regression model is fitted on this subset and used to predict the test dataset. The quality of these results is measured using a comparison of the mean absolute error of the predicted values to the training data, compared with the error from the mean of the training set [1]. This measures the difference between the actual value and the predicted value for each sample, and the lower the mean of all samples, the better the model performance. This was used to infer the threshold variance to be set, and in comparing models of different variable orders. The process was repeated up to the 3rd order of X, where, no higher order features met the threshold variance, and so were irrelevant to predicting the CPU performance.

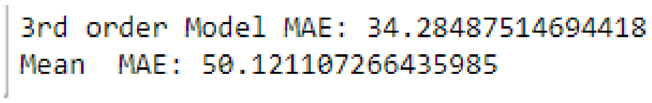


	X^0	MYCT	Mean Memory	Mean #channels	Square Mean #channels
0	1.0	125.0	3128.0	256.0	65536.0
1	1.0	29.0	20000.0	32.0	1024.0
2	1.0	29.0	20000.0	32.0	1024.0
3	1.0	29.0	20000.0	32.0	1024.0
4	1.0	26.0	20000.0	64.0	4096.0

A design matrix was constructed for the nonlinear variable models, by concatenating the relevant features in each order, including the zeroth order, into a joint matrix of all relevant features to be trained. It can be seen that no 3rd order feature met the threshold, and the only feature in 2nd order is the mean number of I/O channels. The values for the predicted performance which were negative were discounted from the data, as it would be illogical to have a negative CPU performance value.

Model Evaluation

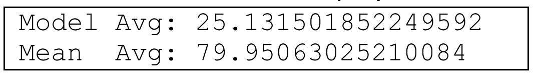
For the final 3rd order Bayesian model, the Mean Absolute Error can be seen to be significantly lower than that of a “poor” model, which in this case was the mean of the data.



```
3rd order Model MAE: 34.28487514694418
Mean  MAE: 50.121107266435985
```

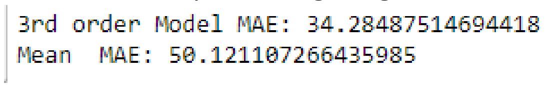
Regression Algorithm Comparison

The performance of these algorithms can be compared using the Mean absolute error returned from each of the algorithm and its comparison with the MAE returned when using the mean of the training output data. The first algorithm which uses multivariate polynomial linear regression returned-



```
Model Avg: 25.131501852249592
Mean  Avg: 79.95063025210084
```

While the second algorithm which uses Bayesian Ridge Regression returned-



```
3rd order Model MAE: 34.28487514694418
Mean  MAE: 50.121107266435985
```

Both these models show a significant reduction in MAE as compared to data formed by taking just the mean of the training data.

Classification Models Selected

Support Vector Machine (SVM)

To identify the different super pixel classes within the cancer classification task, a 'soft-margin' Support Vector Machine is used. This was used as within a high dimensional task (112 dimensions), when the decision boundary is inherently non-linear, classifiers like logistic regression are less accurate.

Experiment Setup

Before implementing a classifier, some initial data analysis was done.

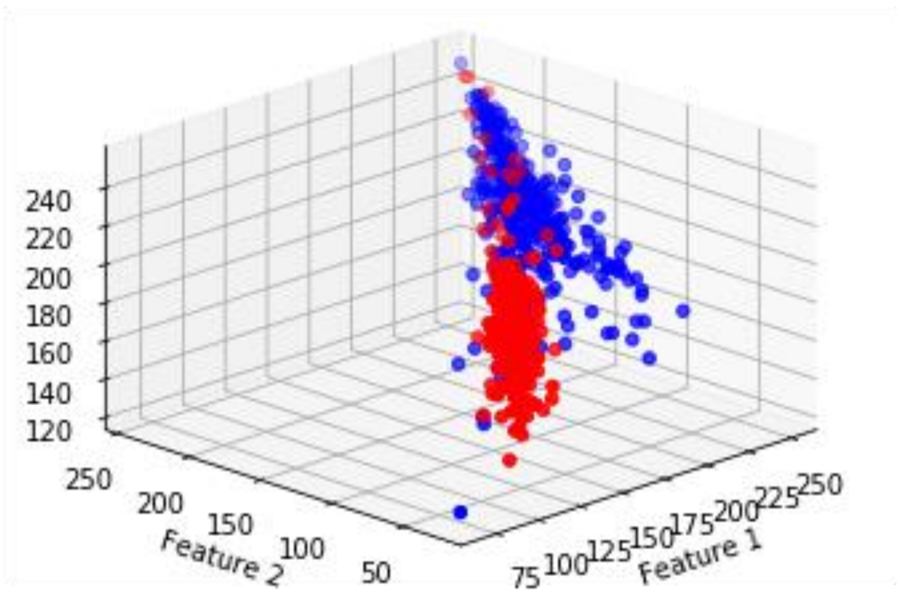


Figure 1: visualization of the first 3 features of the dataset, showing the inherent non-linearity in the class separation

Before using SVM, an attempt was made to use Principle Component Analysis (PCA), an unsupervised dimensionality reduction algorithm, to find a latent lower dimensional space that maximized the covariance between the features, with the aim of increasing the separation between classes. This improved the performance with an 'L1' regularized logistic regression classifier.

$$(eq\ 1)\quad \lambda \sum_{k=0}^{n} |\mathrm{w_i}|$$

'L1' regularization adds a penalty to the loss function, hence it reduces the influence of 'weak' features (i.e features that don't help the loss to minimize). This produces sparse solutions, which is not always ideal, especially if our amount of training examples is relatively low as in this situation. The best accuracy with Logistic Regression on the principle components of the dataset was 0.778 (3 s.f.) on the Kaggle test data, this did not seem like a good score, especially considering that other participants were gaining > 0.8 accuracy.

SVM aims to find a 'hyper-plane' that finds a linear decision boundary within a non-linear task, using what is known as an RBF 'kernel-trick', this algorithm was tried initially because of its capability to find non-linear decision boundaries in a high dimensional space; in a way that could perhaps improve upon the logistic regressions accuracy.

In this implementation, the 'SVC' (C-Support Vector Classification) algorithm was used from the scikit-learn library.

Upon initial data exploration, it is shown that the classes are quite unbalanced:

```
{1.0: 225, 2.0: 375}
```

Thus the parameter class_weight='balanced' was set in the SVC algorithm, which uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data. The tolerance value (stopping criterion) of the optimization parameter (i.e the minimum change in loss function, by which time the optimisation is flagged as converged) was set to '1e-6' instead of the default '1e-3', this allows a slight improvement of accuracy (around +0.04 of the Kaggle score).

The underlying model in the algorithm is the 'soft margin' SVM, with the Gaussian Radial Basis Function (RBF) kernel (eq 3), where we can gain a binary classification from:

$$(eq\ 2)\quad t_{\text{new}} = \text{sign}\left(\sum_{i=0}^{n} \alpha_i\, t_i\, K(\mathbf{x}_i, \mathbf{x}_{\text{new}}) + b \right) \text{ for n data samples}$$

The Gaussian RBF creates a geometry of similarity around the data according to the parameter γ, which allows the hyper-plane to become non-linear in the original space of the data. γ controls the complexity of the decision boundary, a high value indicates a complex decision boundary (tightly fitted with a small variance σ^2), whilst a low value indicates a simpler boundary with a higher variance).

$$(eq\ 3){:}\ K(\mathbf{x}_i, \mathbf{x}_{\text{new}}) = \exp(-\gamma||\mathbf{x} - \mathbf{x}_{\text{new}}||^2),\quad \gamma > 0\text{ , Here } \gamma = 1/2\sigma^2$$

To learn the parameters for the hyper-plane (w and b), we aim to maximize the margin - the distance between chosen points on either side of the hyper-plane, known as the 'support vectors'. Our objective function is (prior to Langrangian transformation):

$$(eq\ 4){:}\ \min\left(w,b \frac{1}{2}\left||w|\right|^2 + C\sum_n \zeta_n\ s.t.\ yn(w^T x_n + b) \geq 1 - \zeta_n \right)$$

C is the scaling parameter of the error term ζ, or the soft margin parameter, which acts like $1/\lambda$ as in our regularization term in *equation 1*. This allows points from either class to cross over the decision boundary, and possibly to become a new support vector. A high value of C in training gives a lower bias, and high variance (thus more prone to overfitting) to our hypothesis function t_{new} , conversely, a low value of C gives a higher bias, and lower variance (thus more prone to underfitting). The role of γ in the RBF is thus linked to C in controlling the bias-variance trade off in the classification.

GridSearchCV within sci-kit learn was used to choose the best parameters C and γ. This tests the parameters in 3 fold cross validation, whereby 2 parts are held for training and 1 part for testing the parameters learned, cycling over all partitions of the data. Here, we have a subset of the mean score (accuracy) and standard deviation over all folds of the cross-validation, for each parameter combination:

```
Grid scores on development set:

0.867 (+/-0.057) for {'C': 0.1, 'gamma': 0.03}
0.875 (+/-0.043) for {'C': 1, 'gamma': 0.0001}
0.898 (+/-0.040) for {'C': 1, 'gamma': 0.001}
0.910 (+/-0.071) for {'C': 1, 'gamma': 0.005}
0.907 (+/-0.052) for {'C': 1, 'gamma': 0.01}
0.905 (+/-0.050) for {'C': 1, 'gamma': 0.02}
0.888 (+/-0.045) for {'C': 1, 'gamma': 0.03}
0.882 (+/-0.038) for {'C': 1.1, 'gamma': 0.0001}


Best combination is {'C': 1, 'gamma': 0.005}
```

With these parameters, an accuracy score of 0.85564 is achieved on the Kaggle test set.

In this particular algorithm, we could perhaps gain a better performance by using the modification proposed by Q L et al. [7]: SVM-RBF-Recursive Feature Elimination (RFE) which expands the nonlinear RBF kernel into its Maclaurin series, and then the weight vector **w** is computed from the series according to the contribution made to the classification hyperplane by each feature.

Gaussian Naïve Bayes

Why Naïve Bayes?

The classification algorithm used here for the classification task of distinguishing between epithelial (where cancer cells live) and stromal (where immune cells and other normal cells live) regions is a method which uses the Bayes' Theorem as the basis, called the Naïve Bayes classifier. This algorithm is very robust which assumes that one feature present in a class is unrelated to the presence of any other feature in the same class [14]. This classifier is easy to use and implement and is very scalable to the large datasets. Since we have multiple features and want to do a multi-class classification, this algorithm is very handy to use it performs well in this scenario. The number of data available with us is comparatively small and if we assume independence of features, then this classifier performs better than other classifiers like logistic and linear classifiers.

There are multiple variations of the Naïve Bayes classifier present, for example, the Gaussian Naïve Bayes, Multinomial Naïve Bayes, Bernoulli Naïve Bayes, etc. The Gaussian Naïve Bayes variation is used for the given classification task as we have used normal distribution and normalized each feature which is done easily by this variation of the classifier. The training time required for the dataset available in this classifier is also less using this classifier which is a known advantage of Naïve Bayes classifier.

The experiment performed

The experiment performed using Naïve Bayes classification [13] is started by taking the input data sets for training and testing. The code uses pipelining [12], normalization [12] and principal component analysis [12], and class balancing [10] where different parameters are tunes for like the percentage of the variance to be maintained for the classification.

The training data is fit into the class balancing and the model is created using the balanced training data and then tested on the test data to come up with one output file containing classifications did which seem fairly accurate according to the Kaggle public scores (private scores pending at the time of writing up the report). Before choosing this classifier, several other classifiers were tried and tested on the training and test data, but their final output in terms of the normalization effect and the accuracy did not match up with the one achieved using Naïve Bayes algorithm. One more difference while experimenting the given data with different classifiers was that the number of mislabeled points in the whole dataset was the least in the chosen classifier as compared to the Multinomial variant, Bernoulli variant, Random Forest classifier, Linear Regression, Logistic Regression, and the decision trees. The parameter tuning during the principal component analysis to increase the performance of Naïve Bayes had a major impact on the final accuracy and the number of mislabeled points. The variance to be maintained is given, based on which the optimum number of components are automatically chosen.

The performance of the Naïve Bayes algorithm is calculated on the total number of mislabeled points out of the total points present in the training set X and then going through the Y_test output to verify if each value in it matches the expected value in the set X or not. Then the total number of such points is calculated and its percentage is given as the accuracy of the algorithm. This same methodology is used as the basis for measuring the performance of all the algorithms evaluated during the evaluation phase, before choosing Naïve Bayes classifier. The training of the classifier is done using the sample files given on kaggle named X and y. The model training was also carried out using the standard way of splitting up the dataset into 30% and 70% data, but later on, commented. One notable observation during the observation was that the performance of the algorithm kept on improving as we decreased the percent of variance to be maintained of the dataset, but only up to the number about 81%, after which it started fluctuating again, before continuously decreasing. A near about similar performance was achieved for when the percentage of variance to be kept was 88, 85, and 79 during the pre-processing. Finally, if during the pre-processing of the dataset the percentage of variance to be maintained was not specified, the algorithm considers all of the features (112) and the performance and the accuracy achieved was not as high as earlier. This led to the conclusion that a better performance is achieved using only a subset of the features provided. The measure of performance throughout was considered as the total number of mislabeled points to the expected values and the score between X_train_res and y_train_res.

Experiment Explanation

The classification algorithm carried out to classify the task uses various modules like numpy, imblearn [11], and sklearn (GaussianNB, StandardScalar, PCA, and make_pipeline) [12]. Data cleaning is applied using normalization of the dataset, where the mean of the distribution is zero and the variance of the distribution is 1.

Following this, the mean is subtracted from each value in the dataset and then divided by the standard deviation of the complete data. The principal component analysis is used to speed up the performance of the classifier. This is also used to specify the percentage of the variance that is to be retained before training the model and classification task.

The other possibility of using PCA is to specify the number of features that are to be chosen. This methodology is not followed using this algorithm and instead, the variance percentage retainment is specified. Based on this choice given, PCA then chooses the optimal number of features for retaining this variance.

The task of normalization, PCA and application of the Gaussian NaïveBayes is calculated on after the other where all three are applied using make_pipeline [12] in which the final estimator's work is done using the Gaussian Naïve Bayes algorithm. The class balancing is done using the library from imblearn [10]. This is necessary so that the number of negatives is somewhat equal to the number of positives. This is done using smote, which creates sample data to train the minor classes which are the positive ones. The model is then trained using the X_train_res and y_train_res training datasets. The prediction is done on the X_test_actual which yields an output of y_pred. This final output is then measured against accuracy by calculating the number of mislabeled points out of the total points (removed from code as only a print statement). The y_pred is a CSV file contains the header line and the points with their classifications. This is submitted to the kaggle competition which gives an accuracy of 0.82845.

Comparison of Gaussian Naïve Bayes and SVC-RBF classifiers

SVC-RBF (SVC) has an improvement of 0.027 over the Gaussian Naïve Bayes (NB) classifier in the Kaggle test set for the classification task.

Taking the same split of the training data using train_test_split, using 70% training and 30% for test, and outputting the classification report from scikit_learn, we obtain the following for each algorithm:

```
       class      precision  recall   f1-score   support

SVC     1.0       0.90       0.94      0.92        65
        2.0       0.96       0.94      0.95       115

avg / total       0.94       0.94      0.94       180

        class      precision  recall    f1-score   support

NB       1.0       0.88       0.88      0.88        65
         2.0       0.93       0.93      0.93       115
```

```
avg / total         0.91         0.91       0.91        180
```

We can see that the average difference in the metrics is around 0.03, reflecting the difference in performance on the Kaggle test set.

It is interesting to note the variance in the precision vs recall values for SVC, and the lack thereof for NB. This could be due to SVC 'allowing' misclassification of a certain class in its training, to avoid drawing decision boundaries that fit to the noise.

Literature Review

Generative Adversarial Nets

In Deep Learning, the models most often used are discriminative models owing to their success in the classification of high-dimensional rich sensory data [5]. In contrast, Generative models are not widely used due to complications that arise in computing probabilistic maximum likelihood estimation. To overcome these issues, Goodfellow et al. demonstrate the use of Generative Adversarial Nets which form a combination of both generative and discriminatory models. It is a framework that pitches the two models against each other through synchronised training. In this framework, a generative model is trained to generate data similar to the training set and the discriminator model that checks if the sample is from the original sample set or created by the generator model. The framework tries to minimise the generative model and maximise the discriminative model. The end result is when the discriminative model is unable to distinguish between the generated distribution and data distribution.

The model is tested on three well know datasets; the MNIST, TFD, CIFAR-10. The experiments showed that even when noise is added to the generator model, the samples generated are of good competitive quality. The model does suffer from various limitations, the biggest being setting the synchronisation between the two models and also the Helvetica scenario in which the generator keeps revolving around copying the same data. While these limitations pose a big problem while designing the model, it also offers many advantages over its competitors like the use of back-propagation to get the gradient and complete independence from Markov chains.

Distributed Representations of Words a Phrases and their Compositionality:

This research compares efficiency and accuracy of variations of a skip-gram neural network model for natural language processing (NLP). This model uses vector representations of words and or phrases in unstructured text, and allows for simple arithmetic linear calculations, as opposed to previous methods which involve complex matrices. An example of this form is given as vec("Madrid") - vec("Spain") + vec("France) giving a result which closely matches vec("Paris").

Skip-gram calculates the average of the log probability of the conditional word, ie, given a chosen word, the surrounding training data is evenly split on either side within the text, and the probability of words in the vocabulary are calculated for each point. The probability of the conditional is known as the Softmax function. The basic Softmax function is computationally costly, being proportional to W, the size of the vocabulary. However, the hierarchical Softmax uses a binary tree representation, with W leaves, and relative probabilities at each node. This results in a proportionality with the length of the path L, which is generally less than logW.

Negative sampling (NS) and Noise Contrastive Estimation (NCE) are alternatives to the Softmax functions, using logical regression to determine the maximum probabilities from the noise in the training data. The noise is a free parameter and was found t be best modelled with a unigram distribution. Sub-sampling discards the most frequent words in the data, based on a chosen threshold value of frequency. The most frequent words, those which appear less than 5 times, were also discarded.

An internal Google dataset was used of 1billion words sourced from news articles. Whilst benefiting from being a large dataset, the sources, all from news articles, may limit generality. A possible extension of this research could be to compare model efficiencies for different source types, such as social media, particularly of character-restricted sites which may further encourage language abstraction and complexity. The models were tested for word combinations and phrase combinations. Sub-sampling was found to increase performance and accuracy in the phrase task. In both tasks, it was found that NS outperformed NCE and Softmax methods. Overall, the Skip-gram method was found to be significantly more efficient than other model architectures.

Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) [8] has been used to solve the problem of topic modelling for text corpora and other collections of discrete data; given a 'context' (e.g a document) can we infer a 'topic', or a mixture of topics in this case, thus preserving the essential statistical relationships between documents in a large collection.

The previous state-of-the-art is Probabilistic Latent Semantic Indexing (pLSI), which is probabilistic alternative of LSI, which uses the variance capturing linear subspace in the space of tf-idf(term frequency- inverse document frequency) to produce a mixture model, where-by the components can be viewed as representations of topics, giving a 'reduced description' associated with the document. LDA takes this a step further by treating the topic mixture weights as a 'k-parameter hidden random variable, rather than a large set of individual parameters which are explicitly linked to the training set', and thus allowing the inference of a topic mixture for a new document. During learning, latent variables Alpha and Beta set priors on the per document topic (onto a Dirichlet distribution), and the per-topic word distributions respectively.

To test the proposed model, the authors compared a number of latent variable models including LDA on two text corpora 'to compare the generalisation performance', using a measure of 'perplexity' for the held-out test set per corpus. They find that as the number of topics in the corpus increases, the perplexity decreases faster with LDA and with a lesser AUC than other models proposed including pSLI. Thus the algorithm succeeds in its goal of generalisation across documents. Second, they test on document binary classification using SVMs on the low-dimensional representations achieved by LDA and word features, in this case LDA performs higher accuracies on most cases apart from the highest proportion of data used for training on the 'Grain vs not grain' data set. Finally they test in a collaborative filtering setting, whereby a user-item matrix is used in place of the document-word representation, they find better predictive perplexity over all (1-50) numbers of topics for LDA vs other models including pLSI.

Auto-Encoding Variational Bayes

This paper proposes a way for training graphical models with unmeasurable posteriors and continuous latent variables. The reparameterization of latent variables is done so that a deterministic mapping can be devised. According to the authors, it enables the lower bound estimators to be different in comparison to the variational parameters. The previous work and existing models have been considered in brief and explanations have been provided. These include the Generative Stochastic Networks, Denoising Auto-Encoder theory and wake-sleep algorithm, which has a scope of improvements [9].

The authors have a proposed an application of auto encoding variational Bayes where the period is a Gaussian cantered on zero and the mean of the distribution is determined by a neural network output. According to the authors, this addresses the parameter estimation problem in a graphical model. The proposed method has a fast training to get the parameters for data generation, an approximation of the posterior for data representation and approximation of the marginal for the evaluation of the model for other activities and models. The authors have discussed various noise distributions, but have not compared them in detail.

The quality of supporting evidence provided by the authors seems fair and extensive. Multiple experiments are run on both Frey-faces dataset and the Most dataset [9]. The authors have compared the new lower bound with the wake-sleep algorithm, the new marginal likelihood with the likelihood of the wake-sleep algorithm. The visualization of 2D manifolds learned with the proposed solution and outlining of the samples got by sampling the dataset from the generative model is also shown by the authors in their study. However, they have not detailed much about the overfitting problem: when does it occur and how is it being handled by the new solution. The result of the experiments also reveals that the proposed model can improve the wake-sleep algorithm.

Bibliography

[1] Jakub Semric, Introduction to machine learning with scikit-learn, 2018, https://www.kaggle.com/nessus/introduction-to-machine-learning-with-scikit-learn

[2] David Hoffman, Multivariate polynomial regression with numpy, 2017, https://stackoverflow.com/questions/10988082/multivariate-polynomial-regression-with-numpy

[3] Anuja Nagpal, L1 and L2 Regularization Methods, 2017, https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

[4]Asaithambi Sudharsan, Why, How and When to apply Feature Selection, 2018, https://towardsdatascience.com/why-how-and-when-to-apply-feature-selection-e9c69adfabf2

[5] Ian J. Goodfellow∗, Jean Pouget-Abadie†, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, 2014, Generative Adversarial Nets.

[6] Mikolov, T. et al, 2013, Distributed Representations of Words and Phrases and their Compositionality. NIPS

[7] Q. Liu, C. Chen, and Y. Zhang, "Feature selection for support vector machines with RBF kernel Feature selection for support vector machines," Artificial Intelligence Review,. August 2011

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," Journal of Machine Learning Research vol. 3, pp. 993–1022, 2003.

[9] Kingma, D. P. W. M., 2013. Auto-Encoding Variational Bayes. arXiv:1312.6114, Issue arXiv:1312.6114.

[10] Brendan Martin, N. K., n.d. LearnDataSci - Predicting Reddit News Sentiment with Naive Bayes and Other Text Classifiers. [Online] Available at: https://www.learndatasci.com/tutorials/predicting-reddit-news-sentiment-naive-bayes-text-classifiers/

[11] G. Lemaitre, F. N. D. O. C. A., n.d. imblearn.over_sampling.SMOTE. [Online] Available at: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html

[12] Pedregosa, F. V. G. G. A. M. V. T. B. G. O. B. M. P. P. W. R. D. V. V. J. P. A. C. D. B. M. P. M. a. D. E., 2011. Scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825-2830.

[13] Ray, S., 2017. 6 Easy Steps to Learn Naive Bayes Algorithm. [Online] Available at: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

[14] Ray, S., 2017. Essentials of Machine Learning Algorithms. [Online] Available at: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/