Machine Learning Coursework

MSc Data Science 2018-19

Submitted by:

Regan Erin Dvoskin 2381442D
Jesus Vera Diaz 2321773V
Mahima Anurag Sood 2418290S

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8RZ

November 26, 2018

Contents

Introduction

Classification and regression are two integral constructs in Machine Learning, with widespread uses in the field. For the purpose of this coursework, we were asked to implement two algorithms each for classification and regression, and test our models against the datasets provided on Kaggle.

The tasks specified are as follow:

Regression: We were asked to evaluate CPU performances from the given training dataset, and then predict performance scores for 41 unseen CPUs. We had to construct two regressions models for the CPUs and then test out models against the dataset on Kaggle. After testing multiple algorithms and evaluating their performances, we chose random forest regression and polynomial regression. A total of 30 submissions were made for regression.

Classification: This task pertained to Tissue Microarray (TMA), where a classifier identified epithelial and stroma regions for images in large patient cohorts. The training file had 600 data points and 112 features, with each datapoint constructed from a small region of coherent appearance, or a superpixel. We were asked to build two classifiers from epithelial vs stroma superpixel tasks, and test them against the dataset on Kaggle. Logistic regression and k-nearest neighbours. A total of 34 submissions were made for classification.

This report contains a description of the models behind the algorithms used, their implementation, experimental setup, and a measurement of performances. The final section contains literature reviews of three papers, as specified in the coursework document.

Regression

The regression data consisted of information about 41 unseen CPUs, with performance measurements. The metrics included benchmark speeds, user experience, and expert reviews. Based on these, a score was calculated to indicate to the user how well a CPU performed. For this task, we tried various algorithms, which included random forest regression, polynomial regression, multiple linear regression and support vector regression. Based on the score we obtained from each solution, after duly testing it against the data provided on Kaggle, we found polynomial regression and random forest regression were the most optimal choices. The table below summarises the scores corresponding to each algorithm used, with the lowest being the best:

Algorithm	Solution	Score
Multiple Linear	Simple Multiple Linear Regression (MLR)	33.40658
	MLR with memory min and max averages	33.00252
Polynomial	Polynomial Regression (PR) with degree = 2	27.52877
	PR with degree = 2, and removing CHMIN, CHMAX and memory min and max averages	26.18
Random Forest	Random Forest Regression (RFR) with 290 trees	23.43963
	RFR with 289 trees and improving hyperparams	22.68259
Support Vector	Support Vector Regression with kernel RBF	26.74104

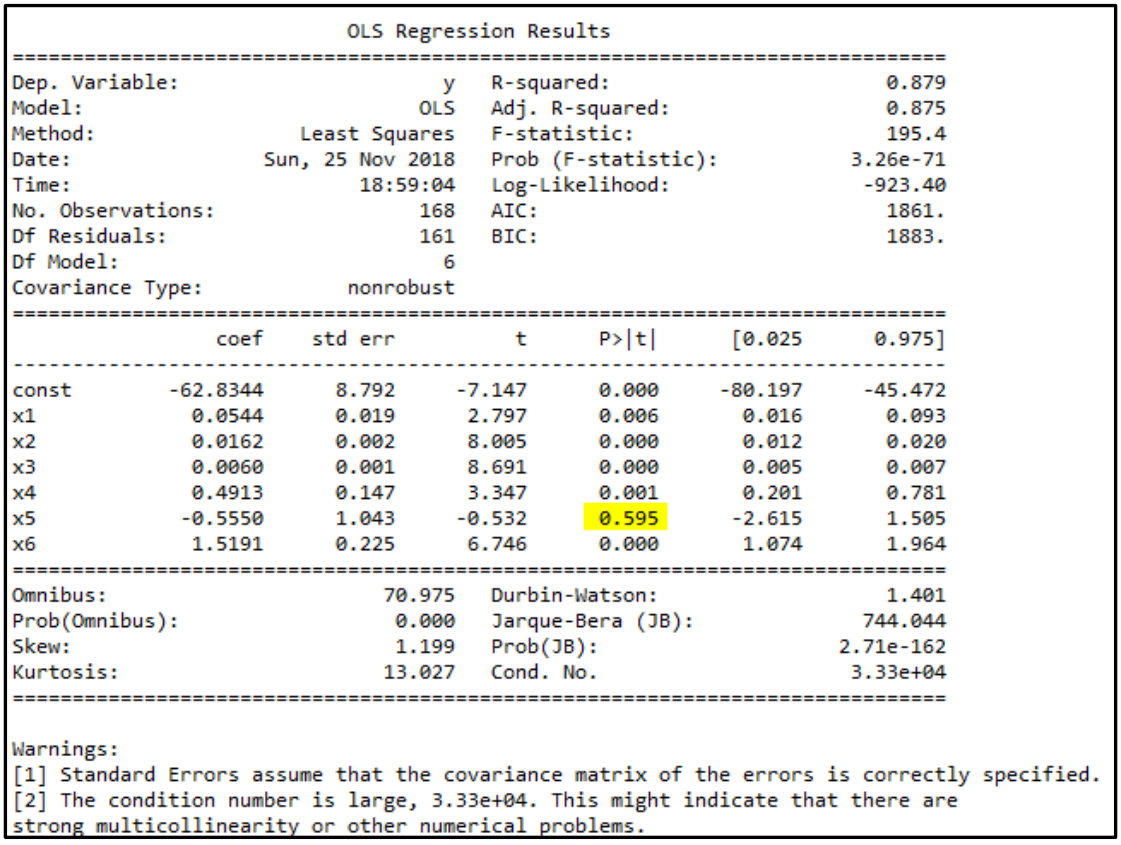
Table 1: Different regression algorithms used, and their respective scores

As evident from the table above, the best scores were obtained via random forest and polynomial regression.

Polynomial Regression

One of the algorithms used for regression was polynomial regression. Polynomial regression is a statistical model where the relationship between an independent variable *x* and a dependent variable *y* can be inferred as nth degree polynomial in *x*. As a non-linear model, it can be used to model non-linear relationships between input features and an output. [4].

For our implementation, we again used scikit-learn. The following modules were imported from scikit-learn: LinearRegression, PolynomialFeatures, and make_pipeline, along with importing stats models for Ordinary Least Squares Regressors [5, 10-12]. Then, the training and test datasets were imported, followed by observing the Ordinary Least Squares Regression for coefficients and results.



OLS Regression Results

Dep. Variable:	y	R-squared:	0.879
Model:	OLS	Adj. R-squared:	0.875
Method:	Least Squares	F-statistic:	195.4
Date:	Sun, 25 Nov 2018	Prob (F-statistic):	3.26e-71
Time:	18:59:04	Log-Likelihood:	-923.40
No. Observations:	168	AIC:	1861.
Df Residuals:	161	BIC:	1883.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-62.8344	8.792	-7.147	0.000	-80.197	-45.472
x1	0.0544	0.019	2.797	0.006	0.016	0.093
x2	0.0162	0.002	8.005	0.000	0.012	0.020
x3	0.0060	0.001	8.691	0.000	0.005	0.007
x4	0.4913	0.147	3.347	0.001	0.201	0.781
x5	-0.5550	1.043	-0.532	0.595	-2.615	1.505
x6	1.5191	0.225	6.746	0.000	1.074	1.964

Omnibus:	70.975	Durbin-Watson:	1.401
Prob(Omnibus):	0.000	Jarque-Bera (JB):	744.044
Skew:	1.199	Prob(JB):	2.71e-162
Kurtosis:	13.027	Cond. No.	3.33e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.33e+04. This might indicate that there are strong multicollinearity or other numerical problems.

Image 1: OLS Regression Results, from polynomial implementation

This result indicated strong multicollinearity, hence the Variance Inflation Factor for multicollinearity was observed. Based on that, two columns were removed:

Column	VIF Factor	Comments
Constant	3.6	
MYCT	1.2	
MMIN	3.0	Merged with MMAX as average
MMAX	3.3	Merged with MMIN as average
CACH	1.8	
CHMIN	2.0	Removed because P value is above 0.05, which shows to be above our significance level of 5%
CHMAX	1.8	Removed due to multicollinearity

Table 2: VIF Factor of each column, and the rationale behind removing a column

This is essentially a trial and error method to improve the predictions. After observing the predictions, columns corresponding to MMIN, MMAX, CHMIN and CHMAX

were dropped. For the same purpose of improving the predictions, memory averages were obtained from the mmax and mmin columns from train and test sets and added to the same sets. Then, linear regression was fitted with polynomial features to obtain a polynomial regression. After obtaining the predictions, they were tested against the dataset on Kaggle and a score of **26.18** was obtained.

Random Forest Regression

The second algorithm used for regression was random forest regression. It is an ensemble algorithm that creates a number of decision trees in training, and outputs the mean prediction of the trees. This prediction translates into the regression value [1].

This prediction can also be seen as a series of models, where each model f_i is a decision tree and the final prediction model g is the sum of all of those. Hence, by creating a decision tree with every model, we are creating a forest.

$$g(x)=f_0(x)+f_1(x)+f_2(x)+...$$

Formula 1: Formula for Random Forest, from Turi Machine Learning Platform User Guide [17].

By adjusting various hyperparameters, we can increase the predictive power and speed of the models. To model this algorithm, decision trees are created from the given dataset. Each node or leaf of a tree represents individual components. After creating the tree, bootstrap aggregating (bagging) is applied, where a random sample is selected with replacement and the tree fitted to it [16].

For our implementation, we used scikit-learn, a machine-learning library in Python [2]. After importing RandomForestRegressor, RandomizedSearchCV, GridSearchCV, and train_test_split from sklearn.model_selection, randomised search on hyperparameters was performed. Random grid combinations were obtained for testing, followed by an evaluation of the model against predictions. GridSearchCV was used for an exhaustive search over a specified parameter value for an estimator, and then fitted to the data [3]. After obtaining the best random hyperparameters from randomised search, a parameter grid was created.

Hyperparam	Selected Value	Comments
bootstrap	False	Method of sampling data points, with or without replacement.
max_depth	26	Max number of levels in each decision tree. Also, prevents overfitting.
max_features	sqrt	Features considered for splitting a node.
min_samples_leaf	1	Min number of data points allowed in a leaf.
min_samples_split	3	Min number of data points allowed in the node before it's splitted.
n_estimators	289	Number of trees in the forest.

Table 3: Parameters grid with best hyperparameters

Then, a model is created using the best hyperparameters and fitted with the training dataset to predict the results of the testing dataset. After testing the predictions against the dataset on Kaggle, a score of **22.68259** was obtained.

Based on the two best scores, random forest regression was chosen as the superior model.

Classification

The classification data consisted of data corresponding to samples of tissue microarrays. There were two classes of tissue samples: epithelial and stromal. Aside from the class of the sample, each sample had 112 features describing it. Of the samples, 600 were given with classes as a training dataset, and 1596 were given as a test dataset. The task was to use some set of the 112 features to create a model that predicts the class of a tissue sample. Prediction labels were created for each sample, which were then submitted to the Kaggle submission page for the task, up to 8 times per day. Kaggle then output an accuracy score for the labels.

For this task, two models were used: one probabilistic and one non-probabilistic. The probabilistic model used was logistic regression, while k-nearest neighbours was the non-probabilistic model.

In both models, the number of features in the final model were limited to 50. Originally, all 112 features were used, but this created highly inaccurate models. This is because not every feature actually has a significant influence on the class of the sample. To choose the top features for a model, scikit-learn provides the class SelectKBest [6]. This uses the ANOVA F-value of each feature to determine the top k features [6]. F-value is the result obtained after running an ANOVA test, to evaluate if the means between two datasets have a statistically significant difference between them. Simply put, it is a ratio of two variances, used to see if the feature affects the predicted value [7]. For our algorithm, several different values of k were used, including 10, 20, 30, 40, 50 and 60. Of the tested values, 50 features provided the highest accuracy models.

For each model, the same base evaluation method was used prior to submitting results to Kaggle. Evaluation prior to submitting to Kaggle was important, as only 8 submissions were allowed per day. As opposed to splitting the training data further into train and test datasets, k-fold cross validation was used to evaluate the models. The goal of this was to avoid overfitting, especially with the training set being relatively small [13]. Models were evaluated using the cross_val_score function from scikit-learn [8]. Five folds were used for the cross validation. The accuracy scores for each fold were printed, and then averaged to create a mean accuracy score. This was examined before submitting the predictions to Kaggle, to determine if the model was promising.

To format the predictions, the output formatting from the example script on Kaggle was used.

Logistic Regression

The first model created for the classification task was a logistic regression model, created using scikit-learn's LogisticRegression class [9]. Logistic regression is a predictive tool used to describe data, that uses a logistic function to model the

relationship between one dependent binary categorical variable and one or more other, independent, variables.

Two models were tested for logistic regression: one using balanced class weights, and one using no class weights. For the model with the balanced class weights, the weight of each class (epithelial and stromal) is based on how often that class appears in the training data [9].

	Mean CV Accuracy	**Kaggle Score**
Balanced class weights	0.90333	0.85983
No class weights	0.90833	0.83472

Table 4: Cross validation and Kaggle scores for balanced weights vs. no class weights, rounded to 5 numbers after the decimal point.

While the model with no class weights had a slightly higher accuracy in cross validation, the balanced class weights scored better in the actual submission to Kaggle. So, the model with balanced class weights was used as the final model, with a score of **0.85983**.

K-Nearest Neighbours

The second model created for the classification task was a k-nearest neighbours (KNN) model. To create the model, KNeighboursClassifier from scikit-learn was used [15]. K-nearest neighbours choses a class for a data point by comparing its distance in vector space to those in the training set. The class is chosen based on the classes of the k points closest to it; the class appearing most often is chosen [14]. This method can also be modified so that the class "vote" of neighbours closest to the data point are weighted higher than those farther away [15].

GridSearchCV from scikit-learn was used to choose the ideal parameters for the model. GridSearchCV works by testing each possible combination of possible parameters given. It records the accuracy from k-fold cross validation for each combination, and returns the parameters which give the highest accuracy [3]. For the cross validation, a k of 5 was used.

The parameters adjusted using GridSearchCV were number of neighbours and weight of neighbours.

Number of Neighbours	**Weights**	**Mean CV Accuracy**	**Kaggle Score**
1	**uniform**	0.815000	Not submitted
1	**distance**	0.815000	Not submitted
5	**uniform**	0.861667	0.82426
5	**distance**	0.860000	0.82217

10	**uniform**	0.883333	0.83472
10	**distance**	0.865000	0.82845
20	**uniform**	0.875000	0.84309
20	**distance**	0.875000	0.83682
30	**uniform**	0.870000	0.83682
30	**distance**	0.870000	0.83054
40	**uniform**	0.866667	0.82845
40	**distance**	0.866667	Not submitted

Table 5: Cross validation and Kaggle scores for different parameters of k-nearest-neighbours. Note some values do not have Kaggle scores because they were not submitted to Kaggle.

The results of the cross validation were used to select which predictions to submit to Kaggle. This allowed for using less of the limited Kaggle submissions. While 10 neighbours with uniform weights had the highest accuracy in the cross validation, the model with 20 neighbours and uniform weights had the highest accuracy given by Kaggle. As such, the model with 20 neighbours and uniform weights was selected as the final model.

Of the two models selected for the final submission, logistic regression was superior, with a Kaggle score of **0.85983**, as opposed to the k-nearest neighbour's score of **0.84309**.

Literature Review

Paper 1: "Hierarchical Dirichlet Processes" by Yee Whye Teh, Michael I. Jordan, Matthew J. Beal and David M. Blei (American Statistical Association, 2012)

This paper introduces Bayesian formalism and hierarchical modelling, citing two examples from varying fields. The authors mention K-binary markers in genetics ad information retrieval, to cite how hierarchical Dirichlet processes can be used to model them. They follow a non-parametric bayesian approach which eposes on Dirichlet processes, which is essentially a measure of measures. A general framework for designing procedures for posterior inference is then mentioned. They also present Markov chain Monte Carlo algorithms for posterior inference under hierarchical Dirichelet measures. By illustrating a stick-breaking process and the Chinese restaurant problem, they successfully explain how to model hierarchical Dirichelet processes in the fields of text modelling and information retrieval.

The problem addressed by the authors is how to segregate information into groups, while knowing that multiple groups might need the same informations, and that the same cluster of information might be used to create new groups, given a prior dataset. This is an important problem because information retrieval and text modelling are two areas whose applications and methodologies can be extrapolated to a plethora of fields. Hence, a unified approach to this problem can significantly optimise the experiments.

A hierarchical Dirichlet process is a measure of a set of random probabilities, over individual groups and globally. The hyperparameters of the hierarchical Dirichlet process contains a baseline probability measure, and the concentration parameters. Using these hyperparameters, we can extend hierarchical Dirichlet processes to two levels, i.e. a tree in which a Dirichlet process is associated to each node, where children of a conditional node are independent, albeit conditionally, given their parents, and one where a particular draw from Dirichlet process at a specific node serves as the base measure for the children. The methodology followed succinctly describes the application of this process to stick-breaking and Chinese restaurant problems. The supporting evidence provided by the authors to back the experiments include conditional distributions using mathematical inference, that helps the reader derive an explicit relationship between individual nodes, and how to cluster them.

Finally, three sampling schemes of Markov chain Monte Carlo for hierarchical Dirichlet process mixture model are explained. These include a Gibbs sampler based on the Chinese restaurant, an augmented representation based on both Chinese restaurant and stick breaking, and a variation on the prior with streamlined bookkeeping. Thus, by assuming that the base distribution is conjugate to the distribution of data, we can focus on issues which are specific to hierarchical Dirichlet processes.

Paper 2: Latent Dirichlet Allocation by David M.Blei, Andrew Y. Ng and Michael I. Jordan (from Journal of Machine Learning Research 3, 2003)

This paper describes Latent Dirichlet allocation, which is a generative probabilistic model used to index and classify collections of discrete data. It is a hierarchical Bayesian model with three levels, where each item of a collection is modelled as a finite mixture over an underlying set of topics. Also, every individual topic is modelled as an infinite mixture over an underlying set of topic probabilities, thus providing a clear representation of the text corpora. The authors then provide various inference techniques based on variational methods, followed by an algorithm to estimate the empirical Bayes parameter. Conclusions of these experiments draw similarities in application in the fields of modelling, text classification, and collaborative filtering. Finally, the results obtained are compared to a mixture of unigrams model and the probabilistic Latent Semantic Indexing model.

The problem addressed by the authors is how to model text corpora and other collections of discrete data. To resolve this problem, they use short descriptions of the members of text corpora, thus enabling an efficient processing of large collections. This does not impact the statistical relationships that are useful for tasks including classification, summarisation, and relevance judgments. The popular "tf-idf scheme" is known for identifying sets of words that are discriminative for sets of words in documents. However, there is also a small reduction in description length of the text, and we do not know much about the inter- or intra- document statistical structure. Latent Semantic Indexing provides an effective solution to this problem, but causes overfitting at times to resolve these issues who existing methodologies, the authors use Latent Dirichlet Allocation (LDA) model to capture exchangeability of documents and words via mixture models.

LDA is a generative probabilistic model of a text collection. Documents within the model are represented as random mixtures over latent topics. Further, each topic in the document is characterised by a distribution of words. Empirical Bayes approach is used to estimate various parameters, leading to LDA distribution on documents in the corpora by marginalising the topic variables and endowing a random parameter of multinomials over topics with a Dirichlet distribution. The approach adopted by the authors works because we obtain a continuous mixture distribution of the specified document from a marginal distribution.

To evaluate the results, the authors then compare LDA to some basic latent variable models, such as the unigram model, a mixture of unigrams, and the probabilistic LSI model. This suitably highlights the conceptual advantages LDA has over other latent topic models. Then, inferences for posterior and variational distribution of the hidden variables given a document are calculated, followed by an illustrative example of a real-world model. This is a subset of the TREC AP corpus, containing over 16000 documents, where experiment's goal is to evaluate the density estimation. The authors then estimate the LDA model's parameters, without referencing actual class labels. They, then use Support Vector Machine to compare results from regular occurrences of words with rare occurrences.

Thus, this paper aptly sums up LDA, while establishing its superiority over other indexing and classification techniques.

Paper 3: "Distributed Representations of Words and Phrases and their Compositionality" by Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean (from Neural Information Processing Systems, 2013)

This paper is about extensions that improve the decor quality and training speed in distributed vector representations. These representations are of high quality and represent exact syntactic and semantic relationships between words, after capturing them. The authors subsample the frequent words, which yields a significant speedup. They also describe negative sampling, which is essentially an alternative to the hierarchical softmax.

The paper addresses the problems word representations have to their order with indifference, and their inability to represent idiomatic phrases. For example, the meanings of "Hand" and "Maiden" cannot be easily combined to obtain "Hand-maiden". Hence, the authors introduce a method for finding particular phrases in text, and demonstrate that learning appropriate vector representations for them is feasible. The authors present many variations of the original Skip-gram model, and a simplified variation of Noise Contrastive Estimation (NCE) for training the Skip-gram model. They conclude that this results in faster training and better vector representations for frequent words, as compared to more complex hierarchical softmax that has been used priorly.

The authors then deduce that the goal of Skip-gram model is to find word representations that can help to predict the neighbouring words in a sentence or a document. More formally, given a particular sequence of training words, this model aims to maximise the average log probability. NCE separated data from noise using logistical regression. The authors present these two models with relevant deceptions and examples, while highlighting how sampling can be optimised. The mythology used appropriately illustrates the same, while demonstrating how to sample large collections.

The results obtained by the authors show that the vectors obtained by the standard sigmoidal recurrent neural networks can have better results with linear word representations, even though they are non-linear. This happens because the vectors perform better as the amount of the training data increases. Proceeding to Additive Compositionality, the additive property of words represented as vectors can be explained by inspecting the training objective. These vectors are trained to predict the surrounding words in the sentence and represent the distribution of the context in which a specific word appears. These values are logarithmically related to the probabilities computed by the output layer, so the sum of two-word vectors is related to the product of the two context distributions. The authors successfully highlight how this can be viewed as a logical AND operation, yielding a result closer to individual meanings of words.

The authors conclude by comparing their results to published word representations. They demonstrate how to use the Skip-gram model to train distributed representations of words and phrases. Then, they conclude that these representations exhibit linear structure that makes precise analogical reasoning possible. Thus, this is an extremely relevant piece of research with manifold applications in the area of syntactic text representations.

References

[1] https://web.archive.org/web/20160417030218/http://ect.bell-labs.com/who/tkh/publications/papers/odt.pdf

[2] https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees

[3]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[4] http://users.stat.umn.edu/~helwig/notes/polyint-Notes.pdf

[5]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

[6] https://scikit-learn.org/stable/modules/feature_selection.html

[7] https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/f-statistic-value-test/

[8]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

[9]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[10] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[11]https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.make_pipeline.html

[12]https://www.statsmodels.org/stable/generated/statsmodels.regression.linear_model.OLS.html?highlight=ols#statsmodels.regression.linear_model.OLS

[13]https://scikit-learn.org/stable/modules/cross_validation.html

[14]https://moodle.gla.ac.uk/pluginfile.php/375201/mod_page/content/36/lecture8.pdf?time=1540945468446

[15] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[16] https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees

[17]https://turi.com/learn/userguide/supervised-learning/random_forest_regression.html