**Project Report**

**Abstract**

The problem tackled involves the ‘valuation’ of mobile devices according to the of their technical (internal) components. The approach taken was to source data directly from Kaggle & databases with raw information and clean it to an acceptable form. A range of machine learning algorithms would be experimented with, and each one’s practical performance against the dataset would be evaluated against direct accuracy & error measures. Overall, the idea is whether the valuation of mobile devices can be directly modelled, in a practical manner.

Baseline algorithms for feature selection & engineering revealed the hidden complexities of the relationship to be modelled. Pre-existing libraries from Sci-Kit Learn & TensorFlow were utilized as a baseline gauge to match and surpass, though high accuracy could not be achieved due to seeming flaws with the data itself.

**Introduction**

The relationship between the ‘technical specifications’ of a modern mobile device and its inherent value is of interest to both the consumer and producer. Only niche research exists on the topic, and investigative journalism and analysis are deficient. Our own investigation seeks to manipulate pre-existing datasets and perhaps reveal any hidden intricacies within a more nuanced and unknown function. As an initial hypothesis, we believe that it is possible to model a mobile device’s value either linearly or through non-linear (e.g. sigmoid kernel) methods.

With the increasing number of smart devices and new, complex innovations that power them, the question of whether such product’s value can be ‘predicted’, may be of value to consumers and companies wanting to buy a new device or undergo innovative research.

**Related Work**

There have been various implementations of machine learning algorithms for the prediction of mobile phone price range. Many of these use the ‘Mobile Price Classification’ dataset from Kaggle. Approaches include artificial neural networks (Nasser & Al-Shawwa, 2019), decision trees and naïve bayes classifiers (Saiteja, 2019). Asim & Khan (2018) also use the same two algorithms on data collected from GSMArena.com.4

The work we aim to do in this report largely simulates the aims described but with data collected ourselves. This allows for more freedom and choice in our features and permits the option of validation with results from related work. We also aim to test additional algorithms such as logistical regression.

|  |  |  |
| --- | --- | --- |
| **Member** | **Work Assigned** | **Work Completed** |
| Jiasheng Qin  z5258237 | Source datasets, clean data & get into right form. LR, SVM learning. Analysis & evaluation; write up & edit report scaffold. | Sourcing datasets (90%),  Scripts for /auxiliary/, data cleaning (80%) & interpolation.  Feature Selection & cluster analysis.  LR.py, SVM.py.  Wrote up report (50%). |
| Edwin Sun  z5112651 | Decision Tree learning & ensembling. Assist in data cleaning & imputing. | Some data cleaning (20%).  Wrote report (20%).  DT.py, LogReg.py, data\_viz.py |
| Genyuan Liang  z5235682 | k-Nearest Neighbors & Naïve Bayes learning. Assist in data cleaning & imputing. | Bayes.py, Knn.py  Wrote report (15%) |
| Chengze Li  z5287104 | Deep Learning, Logistic Reg learning. Assist in data cleaning & imputing. | Sourced 1 dataset.  Wrote report (15%).  LogisticRegression.py, NeuralLearning.py |

**Journal of work completed**

The following table documents the amount and type of work assigned & completed.

\*In the interest of time and the fact that some members had other priorities, e.g. other assignments, it was the most practical to split work in the above manner, and only some custom algorithms could be implemented.

**Phase I: Sourcing Data**

As a first step, all ‘mobile tech specs’ datasets of at least 10,000 in instances in size, were sourced from the web. This included pre-existing datasets on Kaggle1, and datasets sourced via open git repositories2. Despite some interest in ‘web’ scraping for data directly, on sites such as GSMArena.com & etc., potential problems with IP bans meant such methods were not worth the risk. Eventually, it was decided that dataset (gsm2020) would be the main dataset for practical investigation as it was the most recent and contained enough ‘pricing’ information to train supervised learners.

Unfortunately, the data was quite disorganized with columns containing multiple features and non-uniform patterns of feature values, e.g. conversion to USD. Hence, the next step involved quite a bit of data cleaning and interpolation. Another unfortunate fact was that some group members had other priorities or could not contribute diligently. Hence this step took longer than usual.

**Phase II: Data analysis, cleaning & interpolation.**

The large variety and number of features in a mobile phone makes the prediction/classification of prices an ideal problem for machine learning. However, a drawback of this is that not all mobile phones share features found in other phones, e.g. only 47% of the phones have selfie cameras in our dataset.

Another related issue is the numerous presences of null values throughout the data. This may be because of the issue described previously – a certain phone may have these features, or it has been mislabeled. This problem is shown in the figure below, where the white pixels represent null values. It is clear that some pre-processing is required.

A picture containing building

Description automatically generated

*Figure 1: Matrix showing null values (generated using seaborn)*

Python & NumPy would prove useful in the data cleaning step as we were able to create our custom feature-extraction functions and lambdas to apply to entire columns. Eventually, the cleaned dataset without interpolation resulted in 700-800 usable examples (non-null). This result did not seem optimal as it was less than 10% of the original size of the dataset. Hence, the idea of interpolation and imputing missing data was an option to be considered.

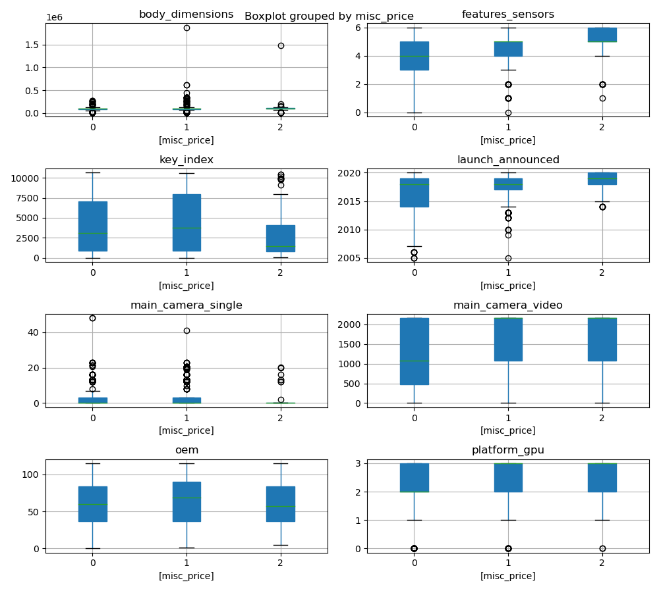
We also plot the correlation heatmap to measure nullity correlation – that is, how strongly the presence or absence of one variable affects the presence of another. Interestingly we see that if an instance is non-null for ‘selfie\_camera\_single’, then there is a higher chance that it is also null for ‘main\_camera\_single’.

A screenshot of a cell phone

Description automatically generated

Figure 2: Correlation Heatmap

After cleaning the data, we plot several boxplots of all the features grouped by their price range. 0 represents phones in the budget range less than 300$, 1 represents mid-range phones between 300$ and 700$ and 3 represents the high-end phones. Note that most features with zero values are imputed to represent a separate class.



Interpolation and imputing features were available in multiple python modules, numpy, sklearn, etc. Rows with fewer missing features (< 3) were targeted for linear and cubic interpolation. The final dataframe was also imputed and would be compared with the original dataframe in terms of ML performance. Since the dataset contained both numeric and categorical features, problems arising from applying regressive procedures to categorical features could be an issue, though linear interpolation seems to work well (See data\_interpolate.py:97)

**Phase III: Baseline Performance, feature selection & dimensionality reduction**

Random forests, MLP, kNN, NB were utilized as performance measures. The highest accuracy attained was 87%7, on linearly interpolated data (data\_interpolate.py: Option A). Whilst non-interpolated/imputed data performed much worse at 50%.8 This raised the question of whether it was possible to improve this accuracy with our own streamlined models and parameters to tweak. The baseline measures mostly involved classification-based predictions by first encoding the labelled ‘misc\_price’ into categorical classes.

The idea of selecting relevant features came to the forefront as through the utilization algorithms from sklearn as baseline performance measures. In feature\_selection.py, random forests allowed us to gauge the expressiveness of each feature. This is due to the ‘randomness’ of features being expressed in individual trees such that more ‘important’ features would therefore show up more times. Through this, we decided to remove features such as “rom”, certain camera-related features, “cpu\_count”. These features were originally thought to be vital to a mobile device’s value, so obtaining these results were quite surprising indeed. It also raised the question of whether the feature-values were recorded in line with proper statistical procedures & standards. This problem would be explored in more detail later.

KMeans clustering was also employed in ‘cluster\_analysis.py’ to get an idea of the intrinsic relationships between the features. The idea of ‘K-Means’ clustering is simply to learn an unsupervised ‘model’ of the data according to their distances in hyperspace. It works like ‘K-Nearest Neighbors’ in that points close to each other are considered to be part of a certain ‘class’. Unfortunately, the results were not ideal as the data was relatively high-dimensional with Euclidean distance measures used by the sklearn clustering algorithms.7 This resulted in high inertias, i.e. squared distances from centre of each cluster, and unintuitive points representing cluster centers on hyperspace. Thus, further dimensionality reduction would be required.

Principal component analysis (PCA) was also utilized in ‘cluster\_analysis.py’. The idea was to ‘compact’ the numerical features as much as possible since some features may simply “explain” another feature. Numeric features, “body\_dimensions", "screen\_size", "scn\_bdy\_ratio", "clock\_speed", "battery", were analyzed, and their variance ratio evaluated. The results were highly significant. ‘body\_dimensions’ seemed to explain over 99.9% of the variance in the other numeric features!7 This was a result that allowed us to dramatically reduce the numeric features required by a factor of five.

The next step was to analyze the performance of K-Dimensional Trees on the data without numeric features at all. The KD-Tree works like a binary search tree in k-dimensional hyperspace and is relatively fast in finding the nearest point . However, this computational complexity increases drastically in higher dimensional space, so the data with removed features (totaling less than 20 in feature-space) was expected to be suitable for KD-Tree predictions. The results were again, not as intuitive due to higher dimensional space. Though the relatively high magnitude in the order of , of the sum of gaussian kernels at each point meant that the points in general were quite close to each other. This was not a great result as lower kernel densities for each point would be ideal for higher information gain. Though the slimmer distances between each neighbor meant that specific clustering & k-NN techniques employed later may be of higher usefulness.

**Phase IV: Experimentation:** **KNN & Naïve Bayes, Baseline**

The K-Nearest Neighbor classification algorithm calculates the distance of the most adjacent data points, then averaging the output.

Unlike most machine learning methods, the KNN algorithm does not have a clear ‘fitting’ step to the training data. We only need to achieve predictions through this method. For distance based KNN; its sensitivity to the scale of the features is a problem as higher dimensions will lead to the observations in each partition rapidly approaching zero. Therefore, in this model, we need to select the features which are more highly correlated with the target to reduce the dimensions.

According to the correlation heatmap, the correlation between price and other attributes is quiet low. The top 5 features of correlation are "ram", "clock\_speed", "features\_sensors", "scn\_bdy\_ratio" and "oem".

First, we look at all the features. In this model we use GridSearchCV from scikit-learn to exhaustive search over specified parameter values for KNeighborsClassifier class. When we set the parameters to "auto", "n\_neighbors" to 9, and "weight" to "distance", the best accuracy 0.66 is obtained. Hence inverse weighting & k=9 must be the middle ground which obtains the right balance between bias & variance.

Perhaps it may improve accuracy by using only the top 5 features. Using the same approach above with features: {'ram', 'clock\_speed', 'features\_sensors', 'scn\_bdy\_ratio', 'oem'} an accuracy of 0.70 is obtained, with parameters "brute", "n\_neighbors"=10, and "weight"="distance".

Next, we examine the Naïve-Bayes classifier, which assumes that the features are independent of each other and calculates the probability according to Bayes theorem12 to determine the label. This assumption is often not true in practical applications. According to the correlation heatmap of the dataset7, there seems to be a significant relationship between attributes, and with a large feature-space, the performance of Bayesian classification will not be very high in theory.

There are several Naïve Bayes classification algorithm types in sklearn: BernoulliNB, MultinomialNB , GaussianNB and ComplementNB. We use these four models to train the data. For our dataset, the accuracy of each kind of Naïve Bayes classification are showed in table.

Table 1: Accuracy of Naive Bayes models

|  |  |
| --- | --- |
| model | accuracy |
| BernoulliNB | 0.50 |
| MultinomialNB | 0.43 |
| GaussianNB | 0.53 |
| ComplementNB | 0.59 |

NOTE: this uses Option C, without interpolation of datapoints

In this dataset, ComplementNB has the highest accuracy while GaussianNB takes second place, though they are all below 60%. Gaussian Naive Bayes (prior: Gaussian distribution) can perform better if the distribution of sample features are mostly continuous values. Multinomial Naive Bayes (prior: polynomial distribution) is more appropriate when the distribution of sample features is mostly multivariate & discrete valued. Bernoulli Naive Bayes (prior: Bernoulli distribution) is suitable for discrete binary values or very sparse multivariate discrete values. Complement Naive Bayes is designed to correct the serious assumption of the standard polynomial naive Bayes classifier. It is particularly suited to imbalanced data sets.11

Our working dataset is quite unbalanced, so it made sense that ComplementNB performed the best. As seen below, the data is significantly skewed to the right. Only a few of datapoints have high pricing while most of the instances are under $2000.

Table 2: The numbers for each label in test data

|  |  |  |
| --- | --- | --- |
| Label | Price range | Test number |
| 0 | lower than $300 | 3768 |
| 1 | lower than $700 & higher than $300 | 135 |
| 2 | higher than $700 | 29 |

Even with interpolation, the overall class distributions are quite a bit too imbalanced.

The table represents a portion of the untrained data. This is a bit of a concern, and perhaps the data could be better categorized, i.e. lower than $100 is class 0, $100-200$ class 1, etc. This however may be much more unintuitive in terms of real-life applications. Unfortunately, the definition of ‘cheap’, ‘mid-ranged’ & ‘expensive’ is largely a human construct that should mostly reflect how an individual really values the product.

**Regression**

Predicting the price of a mobile device with regression was the next theme. The function was not assumed linear though if it was, then linear techniques could be applied interchangeably to the data.

Linear regression (LR) attempts to predict a numeric, quantitative output given numeric input. Its inductive bias assumes that the data are linearly correlated, i.e. a linear combination of the features will correspond to some scalar output. With batch LR, one only requires the ‘normal equation’,

which is a ‘closed-form’ solution that performs well on smaller datasets. Since our option ‘C’ in data\_interpolate.py merely drops null rows, a dataset of 700-800 examples would be trained relatively fast via closed-from. However, with options ‘A’ & ‘B’, we maintain over 10,000 instances. Hence a faster gradient descent (stochastic) was also tested in the interest of alleviating computational complexity. Gradient descent (in course slides handout 1), attempts to find weights such that,

where the next step is dependent on the previous weights and derivatives of the ordinary least-squares loss function L.5 Our algorithm is stochastic (random) & also ends upon hitting an arbitrary convergence criterion *(Ref: LR.py:108)*.

Linear Regression was investigated in LR.py. The ‘LinReg’ class was trained via normal-equation (batch) and stochastic gradient descent8. Unfortunately, the results were almost meaningless. The subpar MSE in the range of and R2 of clearly demonstrates that the data was most certainly not a linear function. Even with L1 & L2 regularization , the performance of LR in general was ultimately the worst performing out of all ML models.3

Perhaps the problem could be better modelled in terms of a classification problem, with linear separators.

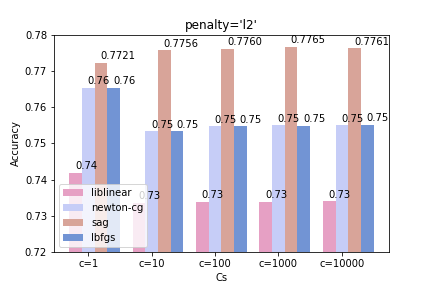
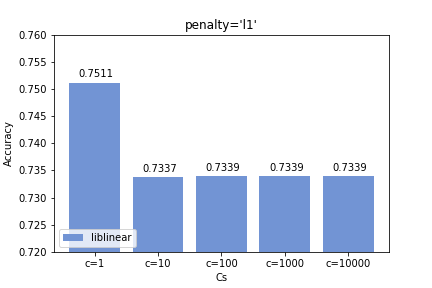
Logistic Regression is such an algorithm for linear, classification-type problems. It can output probabilities for two or more classes. From the course slides (Classification2 handout, pg.102), in a two-class classification problem, a formula for the probability of a certain class Y=1 of an instance X with weights w is given:

In the case of our investigation (Mobile specs vs. Price intervals), we could be able to get Logistic Regression to do well in classification6 by examining the sigmoid function. The continuous mobile phone prices are put into different intervals in our model. In this case, we have 3 classes for prices: , , and . In the case of three-classes, the probability for price Y being in class 1 is,

and the sum of , and is . This model would output the prediction class with the highest score (probability).

The next step is training the model. We decided to try different hyperparameters to get an optimal combination. So, we used the Grid Search module to try different loss functions, normalization coefficients and solvers. In this step we get an accuracy matrix for each parameter combination. Since “newton-cg”, ”sag” and” lbfgs” methods can only use loss function “L1”, there are 25 entries in total in the matrix.

Bar graphs are utilized below to compare the accuracy of different combinations. The test results are divided into two groups by different loss function. The left graph only contains the accuracies for the “liblinear” method with different normalization coefficients . The right graph compares all four methods for different .



According to the results of grid search, we noticed that the “sag” method with penalty “L2” has the highest accuracy. The ‘influences’ of with value 10, 100, 1000 or 10000 are not significant. Therefore, we use to calculate the final accuracy score, 70.05%.

Logistic Regression can deal with classification better than value prediction. The class prediction accuracy has negative correlation with the number of classes. Although we divided the price value into three classes, the accuracy score is only around 70%. Perhaps more complex and advanced data pre-processing could be employed to improve the accuracy of the Logistic Regression model.

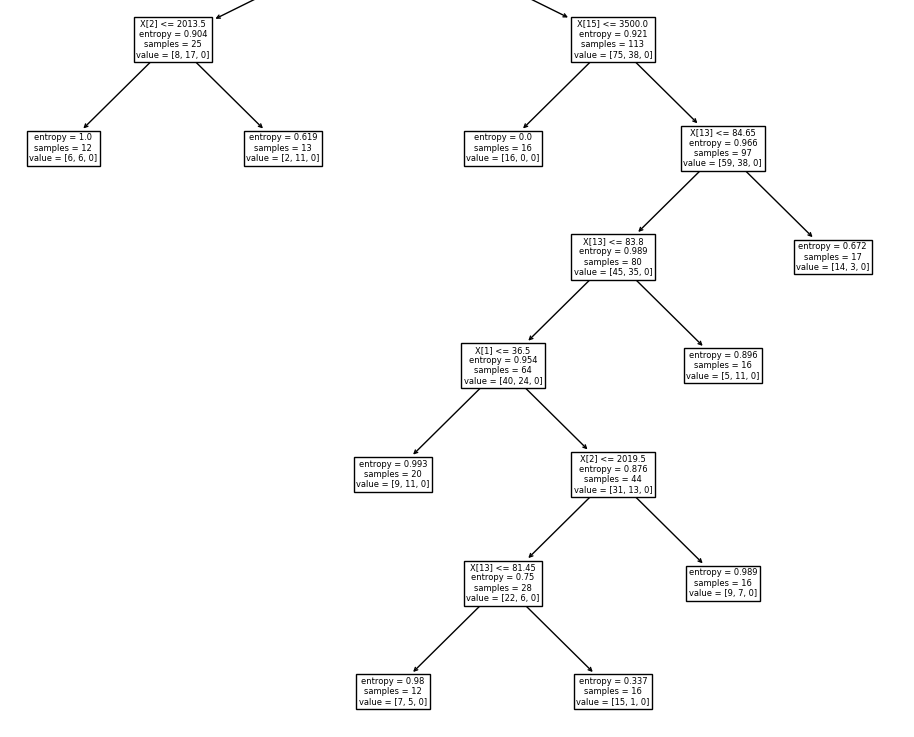
Additionally, some features are continuous, so diving these values into discrete categories may increase the convergence of logistic regression.

**Experimentation: Decision Trees**

The decision was then to examine the nature of splits yield by decision trees and get a better sense of the problem, as regression did not produce a consistent and accurate model. Since decisions trees are quite intuitive as a visual model, this would allow us to better gauge the nature of the relationship between phone features and their prices.

Running the default decision tree yielded an accuracy of 65%. Opting for a randomized hyperparameter search, we reached an accuracy of 68.84%. The complete decision tree may be viewed on our team’s [onedrive](https://unsw-my.sharepoint.com/:f:/g/personal/z5258237_ad_unsw_edu_au/EndZidb_wFZJnpW0Pujr1aMB_kfF_E_w8uLkI_U-kRLpWw?e=B4BXch) folder7.

A smaller branch of the tree is shown below,



As seen above, the entropy at each leaf are relatively high. This raises the concern of potentially ‘bad’ data, i.e. inappropriate sampling techniques leading to poor numeric and categorical values (Ref: GSMArena2020.csv). Prices stored in the GSM databases did not seem to be accounted for inflation, nor can the prices themselves be properly validated due to the large number of instances & missing features.

**Experimentation: Support Vector Machines**

Linear classifiers as seen in the Logistic Regression section appeared to work somewhat decently as a start. Perhaps a linear SVM classifier (‘one-vs-one’) would outperform Logistic Regression due to its flexibility with non-linear kernels and accuracy on smaller datasets with many features. Since we have a dataset of 700-800 instances with 20 features (without interpolation), it made some sense to evaluate the performance of an SVM against Logistic Regression.

Note that a one-versus-one classifier compares two classes out of the 3-5 classes available such that we have a total of individual SVM classifiers where . Predicting a class involves comparing each individual classifier’s score and selecting the class with the highest score.

The default SVM classifier uses a ‘radial-basis’ kernel (gaussian) that utilizes the following mapping:

This kernel may have an infinite number of dimensions, and so would theoretically work fittingly even on very high dimensional (>1 million) data, though it may not scale well. The idea of this kernel is to decrease its value as distance decreases and be intuitive for measuring similarity between points.

A ‘One-versus-One’ SVM classifier on a pipeline with a standard scaler resulted in an accuracy of 76%. The recall & f1 scores for class 0 were also reasonable as well as the confusion matrix that reveals the number of correctly & incorrectly classified examples for 3-class & 5-class classification.7

In general, it was clear that transforming the problem from a regression one to a classification one yield far better results, though the highest accuracy attainable was still up to debate. Perhaps there was a model, e.g. deep learning and boosting that would better map to the data and achieve higher accuracy? Though the data may in fact be quite noisy or sampled improperly to begin with, meaning that the problem was not one about choosing the right model but understanding where the flaws in the dataset lay.

**Conclusion and Discussion**

The results of investigation a mobile device’s tech specs and their physical valuation did not meet our initial expectations. The initial hypothesis was that the relationship was linear or some nonlinear function that could be tested in a streamlined manner through testing a variety of machine learning algorithms.

Through feature selection and cluster analysis, features that did not contribute meaningfully to the output were removed and features that were as easily explained by other features could also be dropped. The resulting feature space was compact enough for classifiers like Logistic Regression, Decision Trees and SVM’s to work on efficiently. However, the results were not optimal as accuracies usually did not surpass 80% on average. Only the random forests and ensembles of DT’s seemed to produce the best results of 87% as an upper bound. Even neural networks with 100’s of hidden layers could not produce a meaningful model capable of accurate predictions (See NeuralLearning.py).

Either the function is too complex to be modelled or, more reasonably, the data had not been sampled in a statistically appropriate manner or adjusted to fit a more uniform standard. Factors such as inflation in pricing may not have been accounted for by databases like GSM and so the resulting data are not in an appropriately scaled form. Features such as ‘RAM’, regional differences between models may not have been sampled appropriately or even included in the dataset. Given the disclaimer “We cannot guarantee that the information on this page is 100% correct” by GSM themselves, it is clear that the dataset operated on would not produce meaningful results when passed into Machine Learning algorithms regardless of the type. Once again, the idea of “garbage in, garbage out” appears to hold true when dealing with the machine learning framework.

**Index and References**

*Plots & diagrams*

-LR: [Coefficients & Results](https://unsw-my.sharepoint.com/:f:/g/personal/z5258237_ad_unsw_edu_au/EgNsnM-WxzFOsImrcxjfTLIBY7mgU0LLDhD3pZe0pEw2-A?e=8TEiKK)

-[Feature-pair plots](https://unsw-my.sharepoint.com/:f:/g/personal/z5258237_ad_unsw_edu_au/Eq8vD47ZlxhHiVYTg9e5Q_MBmMOoAcTipycpt7mfWu-x8w?e=o4uH9a): warning: large number of plots.

-[Figures](https://unsw-my.sharepoint.com/:f:/g/personal/z5258237_ad_unsw_edu_au/EoV3gma-dpVGm38ApOgBDyABuHOQbnPdgmup13yvThqbbw?e=nuDX46)

*References & Notes*

1. <https://www.kaggle.com/arwinneil/gsmarena-phone-dataset> (gsm2017)

2. <https://github.com/MannAgrawal/gsmarena-webscrapping> ()

3. Some errors with indices were encountered with our LinReg L1 gsm2020gradient descent algorithm, but the baseline algorithm from sklearn still resulted in very poor performance.

4. Performed on gsm2017-2018 datasets.

5. COMP9417 Handout slides on Gradient Descent.

6. Though, ‘better’ performance in two-class classification.

7. Link to team onedrive folder: <https://unsw-my.sharepoint.com/:f:/g/personal/z5258237_ad_unsw_edu_au/EndZidb_wFZJnpW0Pujr1aMB_kfF_E_w8uLkI_U-kRLpWw?e=B4BXch>.

8. ‘Hands on Machine Learning with [SK-Learn]”, Guero, ‘Stochastic Gradient Descent, Wikipedia.

9. COMP9417 Lecture slides, MIT lecture slides.

10. Docs, Scipy, Sk-learn, TF, Keras, et al.

11. <https://scikit-learn.org/stable/modules/naive_bayes.html> Complement NB.

12. [Bayes theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem#Derivation)

Asim, M. & Khan, Z., 2018. Mobile Price Class prediction using Machine Learning Techniques. *International Journal of Computer Applications,* 179(29).

Nasser, I. M. & Al-Shawwa, M., 2019. ANN for Predicting Mobile Phone Price Range. *IJAISR,* 3(2), pp. 1-6.

Saiteja, A., 2019. *Mobile Price Prediction Using Machine Learning Classification Techniques,* s.l.: s.n.