Data Mining Exercise 3

Spring 2021

Teacher : Dr.Farahani
Student : Amin Dehghan Monfared SID : 99422085

Q 1.

A kernel is a method of placing a two dimensional plane into a higher dimensional space, so that it is curved in the higher dimensional space. (In simple terms, a kernel is a function from the low dimensional space into a higher dimensional space.)
However, when we work with a non-linear data SVM finds it difficult to classify the data. The easy solution here is to use the Kernel Trick. A Kernel Trick is a simple method where a Non Linear data is projected onto a higher dimension space so as to make it easier to classify the data where it could be linearly divided by a plane. This is mathematically achieved by the Lagrangian formula using Lagrangian multipliers.
For example we can apply a second-degree polynomial transformation(ϕ) to a two-dimensional training set:

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} {x_1}^2 \\ \sqrt{2}\, x_1 x_2 \\ {x_2}^2 \end{pmatrix}$$

When we write the SVM problem in dual from we have:

$$\underset{\alpha}{\text{minimize}}\ \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}\mathbf{x}^{(i)\top}\mathbf{x}^{(j)} \quad - \quad \sum_{i=1}^{m}\alpha^{(i)}$$
$$\text{subject to} \quad \alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

We have $x^{(i)T}x^{(j)}$, if we apply a transformation to x we will have $\phi(x^{(i)})^T\phi(x^{(j)})$.
Now let's look at what happens to a couple of 2D vectors, **a** and **b**, if we apply this second-degree polynomial mapping and then compute the dot product:

$$\phi(\mathbf{a})^\top\phi(\mathbf{b}) = \begin{pmatrix} {a_1}^2 \\ \sqrt{2}\, a_1 a_2 \\ {a_2}^2 \end{pmatrix}^\top \begin{pmatrix} {b_1}^2 \\ \sqrt{2}\, b_1 b_2 \\ {b_2}^2 \end{pmatrix} = {a_1}^2{b_1}^2 + 2a_1b_1a_2b_2 + {a_2}^2{b_2}^2$$
$$= (a_1b_1 + a_2b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^\top \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = (\mathbf{a}^\top\mathbf{b})^2$$

We can see that The dot product of the transformed vectors is equal to the square of the dot product of the original vectors. Hence we have $\phi(\mathbf{a})^T\phi(\mathbf{b}) = (\mathbf{a}^T\mathbf{b})^2$.
So we see that we don't need to apply the transformation to data we can directly use $(x^{(i)T}x^{(j)})^2$ instead of $\phi(x^{(i)})^T\phi(x^{(j)})$ in dual form equation and get the same results.

We can define K(**a**,**b**) =($\mathbf{a}^T\mathbf{b}$)^2 as a kernel. a kernel is a function capable of computing the dot product $\phi(\mathbf{a})^\top\phi(\mathbf{b})$,based only on the original vectors **a** and **b**, without having to compute (or even to know about) the transformation ϕ**.**

Here are some popular kernels :

- Polynomial kernel
  This kernel is used when data cannot be separated linearly.
  The polynomial kernel has a degree parameter (d) which functions to find the optimal value in each dataset. The d parameter is the degree of the polynomial kernel function with a default value of d = 2. The greater the d value, the resulting system accuracy will be fluctuating and less stable. This happens because the higher the d parameter value, the more curved the resulting hyperplane line.

$$k(x, y) = (x^T y + c)^2$$

- Gaussian kernel

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

- Gaussian radial basis function (RBF)

$$k(x, y) = exp(-\gamma||x - y||^2)$$

- Laplace RBF kernel

$$k(x, y) = \exp\left(-\frac{\|x - y\|}{\sigma}\right)$$

- Sigmoid kernel

$$k(x, y) = tanh(\kappa x^T y + c)$$

- Bessel function of the first kind Kernel

$$k(x, y) = \frac{J_{v+1}(\sigma\|x - y\|)}{\|x - y\|^{-n(v+1)}}$$

- ANOVA radial basis kernel

$$k(x, y) = \sum_{k=1}^{n} \exp(-\sigma(x^k - y^k)^2)^d$$

- Linear splines kernel in one-dimension

$$k(x, y) = 1 + xy + xy\ min(x, y) - \frac{x + y}{2}\ min(x, y)^2 + \frac{1}{3}\min(x, y)^3$$

Each kernel have their advantages and disadvantages but we can't say that for a specific task one kernel always works better. Also each kernel have differents hyperparameters that needs to be tunes for our task, so we should try different kernels and see wich one works better.

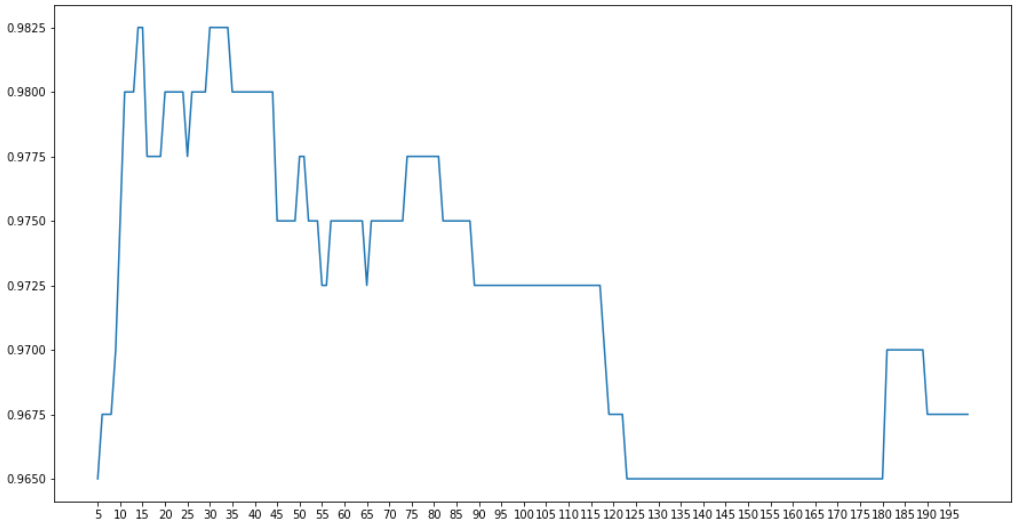
Q 2&3.

We apply svm on the mobile data, here are the results for different kernels:

Kernel	Accuracy
Linear	0.96
Polynomial(deg=2)	0.47
Polynomial(deg=3, best degree)	0.81
RBF	0.90
Sigmoid	0.93

Q 4.

Here we plotted the effect of increasing soft margin parameter on svm, as we see at first by increasing soft margin we get better results, as we allow the model to have more freedom and choose a better line but when we use soft margin too much, model can choose lots of lines and this freedom ends up decreasing our models performance.

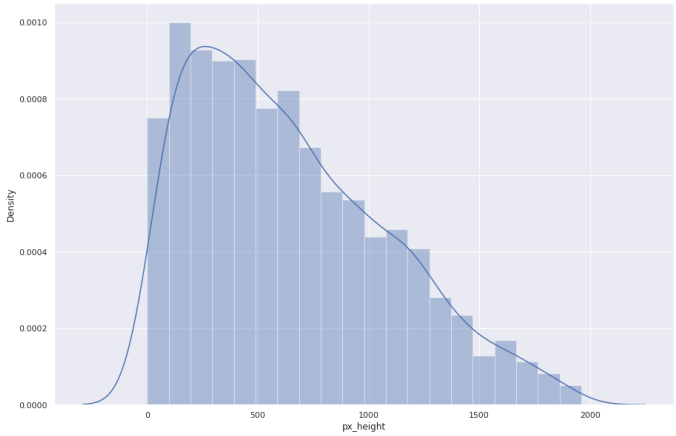


Q 5&6.

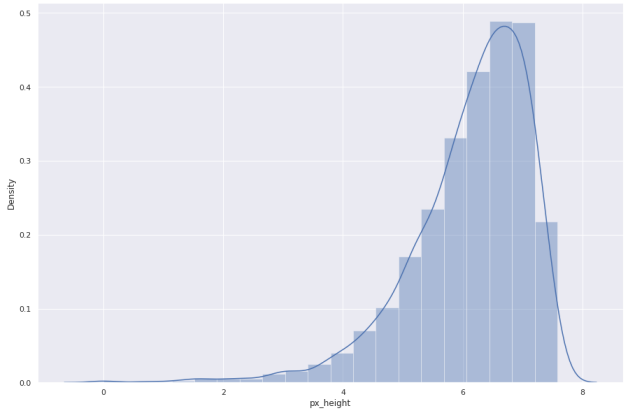
The results of using given feature engineering are shown bellow:

	accuracy
Use binning on battery power	0.96
One hot encoding	0.98
Box-cox on px_height	0.95
Add screen area	0.96
Add mobile Volume	0.96
All	0.95

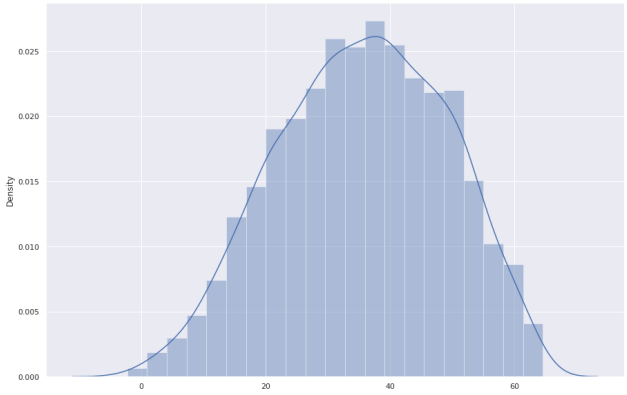
The reason for log transforming your data is to get closer to a normal distribution,in many statistical techniques, we assume that the errors are normally distributed. This assumption allows us to construct confidence intervals and conduct hypothesis tests. By transforming your target variable, we can (hopefully) normalize our errors (if they are not already normal).
px_height feature distribution is like below :



We try to make it more like normal distribution by using log transform, here is the result:



We can see that we failed. So we used box-cox on px_height and got better results.



Here the results is more normal shape. At the core of the Box Cox transformation is an exponent, lambda (λ), which varies from -5 to 5. All values of λ are considered and the optimal value for your data is selected; The “optimal value” is the one which results in the best approximation of a normal distribution curve. The transformation of Y has the form:

$$y(\lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0; \\ \log y, & \text{if } \lambda = 0. \end{cases}$$

Q 7.

Types of decision Trees include:

- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- CART (Classification And Regression Tree)
- CHAID (CHi-squared Automatic Interaction Detector). ...
- MARS: extends decision trees to handle numerical data better.
- Conditional Inference Trees.

Decision Tree implementations differ primarily along these axes:

- the ***splitting criterion*** (i.e., how "variance" is calculated)
- whether it builds models for ***regression*** (continuous variables, e.g., a score) as well as ***classification*** (discrete variables, e.g., a class label)
- technique to eliminate/reduce ***over-fitting***
- whether it can handle ***incomplete data***

The major Decision Tree implementations are:

- **ID3**, or Iterative Dichotomizer, was the first of three Decision Tree implementations developed by Ross Quinlan (Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81-106.)

- **CART**, or *Classification And Regression Trees* is often used as a generic acronym for the term Decision Tree, though it apparently has a more specific meaning. In sum, the CART implementation is very similar to C4.5; the one notable difference is that CART constructs the tree based on a numerical splitting criterion recursively applied to the data, whereas C4.5 includes the intermediate step of constructing *rule set*s.
- **C4.5**, Quinlan's next iteration. The new features (versus ID3) are: (i) accepts both continuous and discrete features; (ii) handles incomplete data points; (iii) solves over-fitting problem by (very clever) bottom-up technique usually known as "pruning"; and (iv) different weights can be applied the features that comprise the training data. Of these, the first *three* are very important--and I would suggest that any DT implementation you choose has all three. The fourth (differential weighting) is much less important
- **C5.0**, the most recent Quinlan iteration. This implementation is covered by a patent and probably, as a result, is rarely implemented (outside of commercial software packages). I have never coded a C5.0 implementation myself (I have never even seen the source code) so I can't offer an informed comparison of C5.0 versus C4.5. I have always been skeptical about the improvements claimed by its inventor (Ross Quinlan)--for instance, he claims it is "several orders of magnitude" faster than C4.5. Other claims are similarly broad ("significantly more memory efficient") and so forth. I'll just point you to studies which report the result of comparison of the two techniques and you can decide for yourself.
- **CHAID** (chi-square automatic interaction detector) actually predates the original ID3 implementation by about six years (published in a Ph.D. thesis by Gordon Kass in 1980). I know every little about this technique.The R Platform has a Package called CHAID which includes excellent documentation
- **MARS** (multi-adaptive regression splines) is actually a term trademarked by the original inventor of MARS, Salford Systems. As a result, MARS clones in libraries not sold by Salford are named something other than MARS--e.g., in R, the relevant function is polymars in the poly-spline library. Matlab and Statistica also have implementations with MARS-functionality

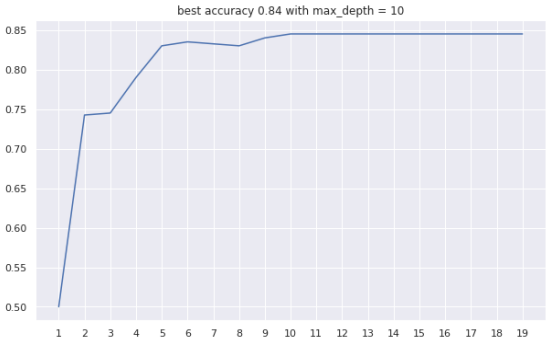
C4.5 is a major step beyond ID3--both in terms of *range* (C4.5 has a far broader use case spectrum because it can handle continuous variables in the training data) and in terms of *model quality*.

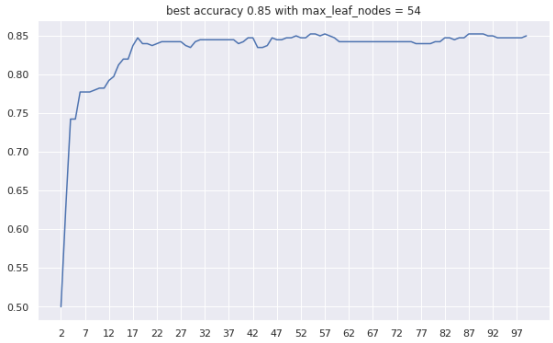
Q 8.

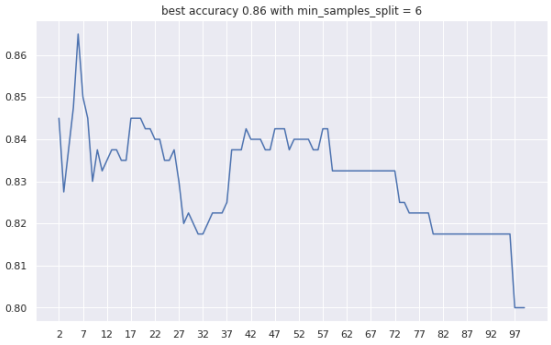
We build a DT using sklearn library in python, using Gini impurity as criteria and use default values for other hyperparameters, and got 0.83 accuracy.

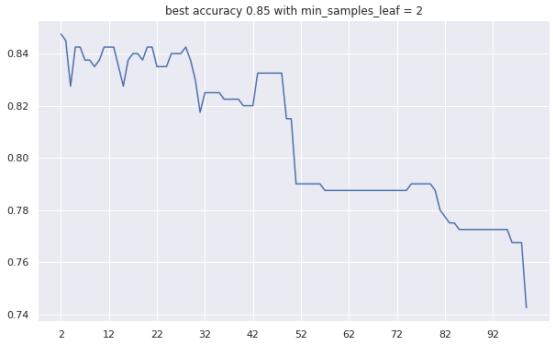
Q 9.

The result of experimenting with different hyperparameters of tree is shown below, experimenting with each hyperparameter, others are set as default value, also the best performance is for each hyperparameter is mention above each figure,









Q 10.

Pruning reduces the size of decision trees by removing parts of the tree that do not provide power to classify instances. Decision trees are the most susceptible out of all the machine learning algorithms to overfitting and effective pruning can reduce this likelihood.
Pruning can be done in two ways :

Pre Pruning

(Early Stopping Rule)

- Minimum no. of sample present in nodes
- Maximum Depth
- Maximum no. of nodes
- Preset Gini Index, Information gain is fixed, which if violated the tree isn't split further

Post Pruning

(Grow the tree and then trim it, replace subtree by leaf node)

- *Reduced Error Pruning* :
  1. Holdout some instances from training data
  2. Calculate misclassification for each of holdout set using the decision tree created
  3. Pruning is done if parent node has errors lesser than child node
- *Cost Complexity or Weakest Link Pruning:*

1. After the full grown tree, we make trees out of it by pruning at different levels such that we have tree rolled up to the level of root node also.
2. We calculate misclassification rate(or Sum of Square residuals for Regression Tree) from test data.
3. Now that we have misclassification(or SSR) for each tree, we add a penalty term to each tree which is penalty coefficient(alpha) times total number of leaves.

$$C_\alpha(T) = R(T) + \alpha|T|$$

C(T) is Cost Complexity Function parametrized by alpha, R(T) is loss function considered(Misclassification Rate or Sum of Square of Residuals), |T| is no. of leaf nodes.

Reason for adding |T|: The most grown tree will most of the times provide lower R(T) as compared to say tree with only root node but that is simply due to large splits.

4. Our task is to find the tree with least cost-complexity function. We do this iteratively by removing subtree which minimizes the reduction in Cost complexity function each time staring from the fully grown tree..

5. Choose alpha by K- cross validation method. Increase of alpha moves our choice of tree to just root node. Average alpha is the final alpha considered for selecting the desired pruned tree.

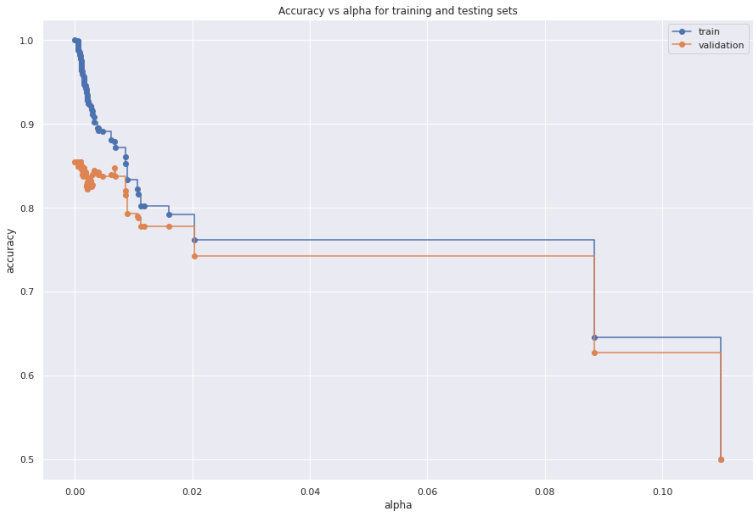
Just some additional points

- Both are Regularization methods in Decision Trees.
- Pre pruning is faster then Post pruning
- Pre pruning goes top to bottom, while post pruning goes bottom up approach

Q 11.

Some of pre pruning techniques are applied when we were experimenting with DT hyperparameters,

By applying *Cost Complexity* method to prune, as we expect our accuracy with drop (so we can say here we didn't have any overfitting problem) but the gap between train performance and validation performance reduces as we increase alpha, her is the figure showing model performances on train and validation data with different alphas.



Q 12.

With Random Forest we get 0.88% as accuracy that is better than DT, The RF algorithm is essentially the combination of two independent ideas: bagging, and random selection of features that both helps the model to have better performance.

Q 13.

- **Versatility.** Random Forest are applicable to a wide variety of modeling tasks, they work well for regression tasks, work very well for classification tasks(and even produce decently calibrated probability scores)
- **Random Forests require almost no input preparation.** They can handle binary features, categorical features, numerical features without any need for scaling.
- **Random Forests perform implicit feature selection** and provide a pretty good indicator of feature importance.
- **Interpretability,** unlike new ML algorithms the rules extracted by Random Forest are explicit and easy to see and understand.
- **Random Forests are very quick to train.** It's a stroke of brilliance when a performance optimization happens to enhance model precision, or vice versa. The random feature

sub-setting that aims at diversifying individual trees, is at the same time a great performance optimization! Tuning down the fraction of features that is considered at any given node can let you easily work on datasets with thousands of features. (The same is applicable for row sampling if your dataset has lots of rows)

- **Random Forests are pretty tough to beat.** Although you can typically find a model that beats RFs for any given dataset (typically a neural net or some boosting algorithm), it's never by much, and it usually takes much longer to build and tune said model than it took to build the Random Forest. This is why they make for excellent benchmark models.
- **It's really hard to build a bad Random Forest!** Since random forests are not very sensitive to the specific hyper-parameters used, they don't require a lot of tweaking and fiddling to get a decent model, just use a large number of trees and things won't go terribly awry. Most Random Forest implementations have sensible defaults for the rest of the parameters.
- **Simplicity.** If not of the resulting model, then of the learning algorithm itself. The basic RF learning algorithm can be written in a few lines of code. There's a certain irony about that. But a sense of elegance as well.
- **Handles Unbalanced Data.** It has methods for balancing error in class population unbalanced data sets. Random forest tries to minimize the overall error rate, so when we have an unbalance data set, the larger class will get a low error rate while the smaller class will have a larger error rate.
- **Low Bias, Moderate Variance.** Each decision tree has a high variance, but low bias. But because we average all the trees in random forest, we are averaging the variance as well so that we have a low bias and moderate variance model.
- **Random Forests can be easily grown in parallel.** The same cannot be said about boosted models or large neural networks.

Q 14.

Q 15.

Yes we can use Random Forest for predicting time series, using it gives us the ability to interpret our model, unlike Deep Learning methods. Compared to other machine learning methods Random Forest is more robust and easier to train as compared to others. Though we should be careful that Random Forest can't process raw time series data, and we should hand feature engineer some features to tell the model the serial relationship between the model. This is done in question 19 and 20 as we tried AdaBoost and Random Forest on Bitcoin data. We use the circular date features like days of week, days of month, month and year to indicate the relation between different data points.

Q 19.

Using AdaBoost on the data that is preprocessed as we said in Q15, on the test set we got mse score of 54391.8135 and 0.37% accuracy as we labeled the predictions within 0.5% range of actual value true,

Q 20.

Using AdaBoost on the data that is preprocessed as we said in Q15, on the test set we got mse score of 5339.7832 and 0.99% accuracy as we labeled the predictions within 0.5% range of actual value true,