



Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

Machine Learning and Data Science

ENCS 5341

Assignment #3:

**KNN, Logistic Regression, SVM, Kernel Methods, and
Ensemble Methods (Boosting and Bagging)**

Mobile Price Classification Dataset.

Prepared by:

Qusay Taradeh

ID: 1212508

Section: 3

Ali Khalil

ID: 1210750

Section: 3

Instructor: **Dr. Ismail Khater**

Date: **Saturday, December 28, 2024**

1. Introduction:

This Assignment aims to predict mobile prices using a dataset scraped from the “Kaggle” website. By building and evaluating a series of classification, regression, Support Vector Machine, and Ensemble models, we aim to identify the most effective model for accurate predictions based on many metrics such as Accuracy, Precision, Recall, F1-Score and AUC-Score. The assignment involves data preprocessing, feature selection, model implementation, and hyperparameter tuning, culminating in a detailed comparison of model performance.

2. Dataset Attributes:

Dataset Name	Instances	Attributes	Dataset Link
Mobile Price Classification	2000	21	Mobile Price Classification Dataset - Kaggle

3. Data Exploration and Preprocessing:

In this section, we are looking to clean the data to avoid missing and inconsistent data as well as redundant ones. So, we can use different strategies to address them.

- **Data information and Description:** In this section, we take info about the dataset, so there are 21 columns and 2000 rows and all of the values were numeric after called description method. Also, there are no missing values in this dataset that are detected by the isnull.sum method. After that, showing the correlation between the classes to identify which classes strongly correlated with our target class which is “price_range”, that is by setting a threshold value as a decision from the correlation heatmap result. Then these columns that is highly correlated with the target column have been selected, and scaled by Standard Scaler to be ready for training or testing.
- **Data Splitting:** The dataset is divided here into training, and testing sets: 80% Train and 20% Test.

4. Building Models, Experiments and Results:

Part 1: K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is an easy-to-understand algorithm used for sorting data into categories (classification) or predicting values (regression). It works by finding the 'k' closest data points to a given input and making predictions based on the most common category (for classification) or the average value (for regression) of these neighbours. The distance between data points is usually measured using methods like Euclidean, Manhattan, or Cosine distance. KNN is simple and easy to use, but it can be slow with large datasets because it needs to calculate the distance to all training samples for each prediction. Also, KNN's performance depends on the choice of 'k' and the distance measurement method used.

In this part, we used the KNN classifier from sklearn library as follows in the steps below:

- **Step 1:** Defining a list of K values from 1 to 50, and distance types in the list of Euclidean, Manhattan, and Cosine KNN metrics.
- **Step 2:** Checking all K values for each distance metric to determine the optimal K with it. That was done using a Cross-Validation score.
- **Step3:** Cross-Validation score has a hyperparameter called 'cv' which is the number of folds, and that's computed by using K-Fold with the number of folds 'n_splits' equals 4 chosen by random, and shuffling was set to shuffle the data before splitting it with random state equals zero to do not affect on the order of folds. Note that each fold is once used as validation and others as training folds. After passing the number of folds, training x and y, KNN classifier, and setting the scoring hyperparameter to 'accuracy' to cross-validation, the mean accuracy result of the current metric and current value of K has been computed and stored in a list. Then after checking all K values, the metric and optimal K value that determine the highest accuracy were stored in a dictionary that has metric as a key and optimal K the value of it.
- **Step 4:** Evaluate KNN after training it with hyperparameters mentioned in the previous step, to calculate Confusion Matrix, Classification Report that contains Precision, Recall, F1-Score and Accuracy, also Area Under the Curve metrics values.
- **Step 5:** Repeating Steps 2, 3 and 4 for each distance metric. Then depending on the maximum accuracy, the optimal metric with an optimal value of K was determined and selected to fit the model with them and obtain the final optimal results of this classifier.

The Confusion Matrix and Classification report **results** for each distance metric classifier with the optimal determined value of K shown as follows:

Confusion Matrix: [[77 27 1 0] [23 52 16 0] [1 31 49 11] [0 1 36 75]] Classification Report: <pre> precision recall f1-score accuracy 0 0.76 0.73 0.75 1 0.47 0.57 0.51 2 0.48 0.53 0.51 3 0.87 0.67 0.76 accuracy 0.63 </pre>	Confusion Matrix: [[89 14 2 0] [20 57 14 0] [1 27 56 8] [0 1 28 83]] Classification Report: <pre> precision recall f1-score accuracy 0 0.81 0.85 0.83 1 0.58 0.63 0.60 2 0.56 0.61 0.58 3 0.91 0.74 0.82 accuracy 0.71 </pre>	Confusion Matrix: [[90 13 2 0] [36 39 14 2] [3 24 38 27] [0 2 18 92]] Classification Report: <pre> precision recall f1-score accuracy 0 0.70 0.86 0.77 1 0.50 0.43 0.46 2 0.53 0.41 0.46 3 0.76 0.82 0.79 accuracy 0.65 </pre>
---	--	---

Figure 1: Euclidean

Figure 2: Manhattan

Figure 3: Cosine

Euclidean Distance: Optimal K = **48**, Manhattan Distance: Optimal K = **48**, and Cosine Distance: Optimal K = **42**. From the figures above we notice that the **highest accuracy** classifier was KNN with hyperparameters (metric = **Manhattan**, n_neighbors = **48**) that's accuracy equals **71%**. Whereas average **Precision** equals **0.71**, **Recall** equals **0.71**, **F1-Score** equals **0.71**, and Area Under the Curve score (One-Versus-Rest) equals **0.89**.

Here is the results table for each distance metric:

Distance\Metric	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Euclidean	63%	0.65	0.63	0.63	0.84

Manhattan	71%	0.71	0.71	0.71	0.89
Cosine	65%	0.62	0.63	0.62	0.83

From the metrics' values above, we conclude that the highest classification performance was for **Manhattan** with **K** equals **48** and this will be used as the optimal choice of our dataset. However, there is small differences between Euclidean and Cosine, such that Cosines' accuracy is 2% greater than Euclidean whereas other metrics' values as Precision, F1-Score, and AUC-Score are a little bit more than Cosines' values.

The value of **K** equals 48 determined based on the accuracy of cross-validation and the accuracy was the best value depending on some points such that the choice **K = 48** for our KNN classifier reflects an effective balance tailored to our dataset's characteristics. With 19 features (selected for their high correlation with the target) and a relatively large training set of 1600 (80% of the dataset) records, **K = 48** ensures that approximately 3% of the training data influences each prediction. This value avoids overfitting to individual data points. Additionally, the dataset's 19-class target distribution benefits from a moderate **K**, which accounts for sufficient neighbours to represent minority classes without being overly influenced by irrelevant ones.

***Note:** This part was done by Qusay Taradeh 1212508

Part 2: Logistic Regression

Logistic Regression is a simple and popular algorithm for sorting data into categories. It works by calculating the chance that a given input belongs to a certain group, using a logistic function that gives results between 0 and 1. The algorithm adjusts the weights of the input features to find the best curve that separates the groups. Its main advantage is its simplicity and easy interpretation. However, it assumes a straight-line relationship between the input features and the outcome, which might not always be true. Additionally, it can have trouble with complex datasets where the groups are not easily separated.

In this part, we used the Logistic Regression Model from sklearn library as follows in the steps below:

- **Step 1:** Create a Logistic Regression Model with a random state equal to 42 that suits the dataset.
- **Step 2:** Defining list of hyperparameters as follows: {**C: inverse of regularization strength** 0.001, 0.01, 0.1, 10, 100, 1000, **max_iter: maximum number of iterations taken by solver** that is in our case is **linear** 1000, 2000, 2500, 5000, **penalty: type of regularization** l1 for L1 and l2 for L2} to be tuned using Grid Search that is imported from sklearn library.
- **Step 3:** Learn a Model using the fit method and find the best hyperparameters tuned using the best_estimator method.
- **Step 4:** Evaluate the Model after training it with hyperparameters mentioned in the previous step, to calculate the Confusion Matrix, a Classification Report that contains Precision, Recall, F1-Score and Accuracy, also Area Under the Curve metrics values.
- **Step 5:** Repeating Steps 2 to 4 but with a change penalty to 'l2' to use L2 regularization such as L1 Regularization the first time.

After training the model and hyperparameters tuning the best hyperparameters and evaluation results respectively shown in the figures below:

```
LogisticRegression(C=10, max_iter=1000, penalty='l1', random_state=42, solver='liblinear')
```

Figure 4: best hyperparameters of L1

```
LogisticRegression(C=1000, max_iter=1000, random_state=42, solver='liblinear')
```

Figure 5: best hyperparameters of L2

Confusion Matrix:				Confusion Matrix:			
[[100 3 2 0]				[[100 3 2 0]			
[0 69 22 0]				[0 69 22 0]			
[0 18 72 2]				[0 18 72 2]			
[0 0 1 111]]				[0 0 1 111]]			
Classification Report:				Classification Report:			
	precision	recall	f1-score		precision	recall	f1-score
0	1.00	0.95	0.98	0	1.00	0.95	0.98
1	0.77	0.76	0.76	1	0.77	0.76	0.76
2	0.74	0.78	0.76	2	0.74	0.78	0.76
3	0.98	0.99	0.99	3	0.98	0.99	0.99
accuracy			0.88	accuracy			0.88
macro avg	0.87	0.87	0.87	macro avg	0.87	0.87	0.87
weighted avg	0.88	0.88	0.88	weighted avg	0.88	0.88	0.88
ROC-AUC Score (Multiclass OVR): 0.9746				ROC-AUC Score (Multiclass OVR): 0.9748			

Figure 6: metrics values of L1

Figure 7: metrics values of L2

From Figures 4 and 5 we notice that the best hyperparameters for both L1 and L2 were approximately the same except C was 10 for the L1 case and 1000 for the L2 case. This indicates that regularization strength in L1 was stronger than L2 such that less value of C leads to more strength of regularization.

Figures 6 and 7 that the metrics' values look like same approximately either using L1 or L2 regularization techniques. The similar results between logistic regression with L1 and L2 regularization show that our dataset works well with logistic regression and isn't greatly affected by the type of regularization. This happened because the features are likely not too correlated **"that we make the threshold a little bit small that selected columns increased"** or sparse, and the dataset has clear patterns that both methods can easily capture. Even though the C values were different, both were well-tuned, leading to similar results. In multiclass classification, where one model is trained for each class, combined metrics like ROC-AUC make small differences less noticeable. Overall, both methods perform well, with L1 focusing on selecting important features and L2 providing more stable results.

The metrics' values were obtained as follows: **Accuracy = 88%, Precision, Recall and F1-Score** each one of them was **0.87**, and **ROC (One-Versus-Rest) = 0.97**.

Performance Comparison with our Optimal KNN

Model\Metric	Accuracy	Precision	Recall	F1-Score	ROC-AUC
KNN	71%	0.71	0.71	0.71	0.89
Logistic Reg.	88%	0.87	0.87	0.87	0.87

From above table we notice that Logistic regression outperforms KNN in all major metrics except for a slight difference in ROC-AUC. It achieves higher accuracy (**88%** compared to KNN's **71%**) and better precision, recall, and F1-score (**0.87** for logistic regression vs. **0.71** for KNN). This shows that logistic regression is better at correctly predicting and generalizing the data. The higher ROC-AUC for KNN (**0.89**) suggests it handles

class separation well, but this advantage doesn't translate into better overall performance. Logistic regression's strong results come from its ability to model linear patterns in the data effectively, making it the better choice for this dataset.

***Note:** This part was done by Qusay Taradeh 1212508

Part 3: Support Vector Machines (SVM)

SVM is a supervised machine learning algorithm used for classification and regression. It aims to find a decision boundary that best separates the data into different classes. There are four key concepts in SVM they are Hyperplane, Support Vectors, Margin, and Kernel Trick.

Linear SVM is used if the data is linearly separable because it will find the hyperplane that best divides the data. Non-linear SVM is used if the data is not linearly separable it will use the kernel trick to map the data to a higher dimensional space where a linear decision boundary is found.

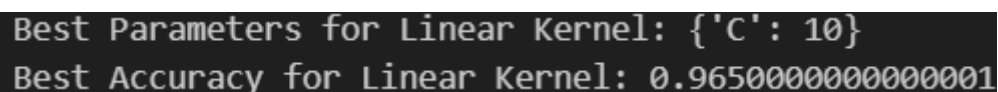
SVM is less prone to overfitting in high dimensional space especially if the correct kernel is used on the data set. SVM works very well in for high dimensional data and Non-Linear data thanks to the kernels it uses.

SVM can be very complex so it will require more computational resources, especially with large datasets. It also requires the right kernel and tuning of its parameters which can be challenging.

In this implementation of SVM, we used libraries from sklearn. We first imported SVC (Support Vector Classification) API which contains the implementation of SVM then we did the following:

Linear Kernel:

First: we used Grid Search to find the best hyperparameters for each kernel so for the linear kernel we found that the best Regularization Parameter (C) based on our data and found it to be 10 with an accuracy of 0.96 for the linear kernel as shown in the figure below:



```
Best Parameters for Linear Kernel: {'C': 10}
Best Accuracy for Linear Kernel: 0.9650000000000001
```

Second: using the best hyperparameters we train our model on the training data and use it to predict the test data.

Third: we then get the evaluation metrics using the predicted data and the actual targets of the test data and we had the following results:

Accuracy = 0.9775, Precision = 0.98, Recall = 0.98, F1-Score = 0.98. ROC-AUC = 0.99.

Polynomial Kernel:

We also repeat the same step of finding the best hyperparameters here but the parameters are more this time so we have Regularization Parameter (C), Polynomial Degree (degree) and Independent Term (coef0) using Grid Search we found that the best hyperparameters are C = 0.1, coef0 = 3, degree = 2 with an accuracy of 0.91 as shown in the figure below.

```
Best Parameters for Polynomial Kernel: {'C': 0.1, 'coef0': 3, 'degree': 2}
Best Accuracy for Polynomial Kernel: 0.9175000000000001
```

Then we trained our model using these parameters and we tested it using the test data to get the following results:

Accuracy = 0.9450, Precision = 0.94, Recall = 0.95, F1-Score = 0.94, ROC-AUC = 0.99.

RBF Kernel:

In this part, we also find the best hyperparameters for the RBF kernel are the Regularization Parameter (C) and Kernel Coefficient (gamma) using grid search they are found to be C = 10 and gamma = 0.01 with an accuracy of 0.91 as shown in the figure below.

```
Best Parameters for RBF Kernel: {'C': 10, 'gamma': 0.01}
Best Accuracy for RBF Kernel: 0.914375
```

Then we trained the model using these hyperparameters and evaluated it using the evaluation metrics to get the following results:

Accuracy = 0.9450, Precision = 0.93, Recall = 0.93, F1-Score = 0.93, ROC-AUC = 0.99.

Comparing the different kernels of SVM:

The following figures show the results of each kernel:

Kernel\Metric	Accuracy	Precision	Recall	F1-Score	AUC-Score
Linear	97%	0.98	0.88	0.98	0.99
Polynomial	94%	0.94	0.95	0.94	0.99
RBF	94%	0.93	0.93	0.93	0.99

From these results, the linear kernel performs the best for this dataset, indicating the data is linearly separable. The polynomial and RBF kernels also performed well but slightly underperformed in accuracy, precision, and recall compared to the linear kernel. This shows the importance of selecting a kernel that aligns with the dataset's characteristics.

Selecting the correct kernel for a model depends on the dataset if it is linearly separable data, as seen in our dataset, the best kernel is the linear one. If it is non-linear data a polynomial kernel would be better although it underperformed above because of the nature of our dataset. RBF kernel also works for complex non-linear data but it requires fine-tuning of gamma to get the optimal performance it also underperformed due to the linearity of the dataset we used in our project.

***Note:** This part was done by Ali Khalil 1210750

Part 4: Ensemble Methods

Ensemble methods combine the predictions of multiple models to improve performance and reduce variance as well as minimize overfitting the two most common ensemble techniques and the ones implemented in this project are:

Boosting (AdaBoost):

This ensemble method combines weak learners sequentially to get a better or stronger learner. Each new model will correct the errors of the previous one. Also, aims for high accuracy.

Bagging (Random Forest):

Random Forest will train multiple models of decision trees independently on data subsets and it will be using voting from all the models to get the prediction to reduce the variance and improve robustness. It is less prone to overfitting compared to boosting.

Then we implemented it in our code using the libraries in sklearn we imported AdaBoostClassifier and RandomForestClassifier to implement both boosting and bagging.

Then we performed a grid search to find the best hyperparameters to train our models and found that for boosting the best number of weak learners (n_estimators) is 150 and the learning rate is 0.1 on the other hand for bagging we found that the best number of weak learners (n_estimators) is 100 and the max_depth is None, min_samples_split is 2, min_samples_leaf is 2, max_features is sqrt

Then we used the evaluation metrics to evaluate both models and got these results:

Method\Metric	Accuracy	Precision	Recall	F1-Score	AUC-Score
Boosting	75%	0.75	0.75	0.75	0.86
Bagging	89%	0.89	0.89	0.89	0.98

From the above results, we find that bagging performed significantly better on our dataset and that's likely because it has high variance and low bias. Because if the bias was high would do better.

From all the metrics we found ensemble method performed better than KNN and logistic regression because they leverage the strength of multiple models (e.g., trees) to reduce bias and variance. KNN and Logistic Regression are single-model approaches that lack mechanisms to adaptively correct their errors although it was worse than support vector machines because ensemble methods (especially Boosting) may overfit the data, particularly if the dataset is small (which our data is) or noisy, which can limit their performance relative to SVMs. In contrast, SVMs are inherently resistant to overfitting due to their margin-maximizing property.

***Note:** This part was done by Ali Khalil 1210750

5. Conclusion:

In this assignment, we implemented a variety of models as KNN with tuning best metric and optimal value of K in our dataset, and how increase or decrease the value of K and metric type make effect on the classification performance. To add to that, in Logistic Regression model changing Regularization technique was not affects classification performance of Logistic Regression Model since our selection of correlated features with target feature, such that if we increased the selection threshold it may be different between regularization techniques usage. We implemented Support Vector Machines (SVM) with different kernels. Linear SVM performed the best, achieving high accuracy and precision, and demonstrated its strength in handling linearly separable data. The Polynomial and RBF kernels performed well too, but they were slightly less effective on this particular dataset, which was more suited to linear separability. For Ensemble Methods, we implemented Boosting (AdaBoost) and Bagging (Random Forest). Bagging (Random Forest) showed superior performance in comparison to Boosting, as it reduced variance and provided a more robust classification. Boosting, though effective at reducing bias, showed signs of overfitting, particularly with the small and noisy dataset used in our experiment.