Name: Atharva Chundurwar ASU ID: 1233380696

Importing Pandas and Reading the Dataset

```
import pandas as pd

df = pd.read_csv(r"titanic/train.csv")

df.head(), df.isna().sum()
```

```
(   PassengerId  Survived  Pclass  \
 0            1         0       3
 1            2         1       1
 2            3         1       3
 3            4         1       1
 4            5         0       3

                                                 Name     Sex   Age
SibSp  \
 0                            Braund, Mr. Owen Harris    male  22.0
1
 1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
 2                             Heikkinen, Miss. Laina  female  26.0
0
 3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
 4                           Allen, Mr. William Henry    male  35.0
0

    Parch            Ticket     Fare Cabin Embarked
 0      0         A/5 21171   7.2500   NaN        S
 1      0          PC 17599  71.2833   C85        C
 2      0  STON/O2. 3101282   7.9250   NaN        S
 3      0            113803  53.1000  C123        S
 4      0            373450   8.0500   NaN        S  ,
 PassengerId      0
 Survived         0
 Pclass           0
 Name             0
 Sex              0
 Age            177
 SibSp            0
 Parch            0
 Ticket           0
 Fare             0
 Cabin          687
 Embarked         2
 dtype: int64)
```

Building a Preprocessing Pipeline

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree,
export_text
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

# Defining the target and features
target = "Survived"
features = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare",
"Embarked"]

X = df[features].copy()
y = df[target].copy()

# Grouping the columns into Numerical and Categorical
num_cols = ["Age", "SibSp", "Parch", "Fare"]
cat_cols = ["Sex", "Embarked", "Pclass"]

# Numerical Columns Transfomer

num_tf = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
])

# Categorical Columns Transformer

cat_tf = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("ohe", OneHotEncoder(handle_unknown="ignore",
sparse_output=False))
])

#

preprocess = ColumnTransformer(
    transformers=[
        ("num", num_tf, num_cols),
        ("cat", cat_tf, cat_cols),
    ],
    remainder="drop"
)

# Decision Tree Pipeline

pipe_dt = Pipeline(steps=[
    ("prep", preprocess),
```

```
    ("clf", DecisionTreeClassifier(random_state=42))
])

# Defining the parameter grid

param_grid = {
    "clf__criterion": ["gini", "entropy", "log_loss"],
    "clf__max_depth": [3, 4, 5, 6, None],
    "clf__min_samples_split": [2, 5, 10],
    "clf__min_samples_leaf": [1, 2, 5]
}

# Defining the Grid Search CV

grid_dt = GridSearchCV(
    estimator=pipe_dt,
    param_grid=param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)

grid_dt.fit(X, y)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('prep',

ColumnTransformer(transformers=[('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median'))]),

['Age',

'SibSp',

'Parch',

'Fare']),

('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('ohe',

OneHotEncoder(handle_unknown='ignore',
```

```
sparse_output=False))]),

['Sex',

'Embarked',

'Pclass'])])),
                                        ('clf',

DecisionTreeClassifier(random_state=42))]),
             n_jobs=-1,
             param_grid={'clf__criterion': ['gini', 'entropy',
'log_loss'],
                         'clf__max_depth': [3, 4, 5, 6, None],
                         'clf__min_samples_leaf': [1, 2, 5],
                         'clf__min_samples_split': [2, 5, 10]},
             scoring='accuracy')

# Taking the best estimator from the grid search

best_dt = grid_dt.best_estimator_
best_dt.get_params()

{'memory': None,
 'steps': [('prep', ColumnTransformer(transformers=[('num',
                                     Pipeline(steps=[('imputer',

SimpleImputer(strategy='median'))]),
                                     ['Age', 'SibSp', 'Parch',
'Fare']),
                                    ('cat',
                                     Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),
                                                     ('ohe',

OneHotEncoder(handle_unknown='ignore',

sparse_output=False))]),
                                     ['Sex', 'Embarked', 'Pclass'])])),
  ('clf',
   DecisionTreeClassifier(criterion='entropy', max_depth=5,
min_samples_leaf=2,
                          min_samples_split=10, random_state=42))],
 'transform_input': None,
 'verbose': False,
 'prep': ColumnTransformer(transformers=[('num',
                                    Pipeline(steps=[('imputer',
```

```
SimpleImputer(strategy='median'))]),
                                    ['Age', 'SibSp', 'Parch', 'Fare']),
                                   ('cat',
                                    Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),
                                                    ('ohe',

OneHotEncoder(handle_unknown='ignore',

sparse_output=False))]),
                                    ['Sex', 'Embarked', 'Pclass'])]),
 'clf': DecisionTreeClassifier(criterion='entropy', max_depth=5,
min_samples_leaf=2,
                       min_samples_split=10, random_state=42),
 'prep__force_int_remainder_cols': 'deprecated',
 'prep__n_jobs': None,
 'prep__remainder': 'drop',
 'prep__sparse_threshold': 0.3,
 'prep__transformer_weights': None,
 'prep__transformers': [('num',
   Pipeline(steps=[('imputer', SimpleImputer(strategy='median'))]),
   ['Age', 'SibSp', 'Parch', 'Fare']),
  ('cat',
   Pipeline(steps=[('imputer',
SimpleImputer(strategy='most_frequent')),
                   ('ohe',
                    OneHotEncoder(handle_unknown='ignore',
sparse_output=False))]),
   ['Sex', 'Embarked', 'Pclass'])],
 'prep__verbose': False,
 'prep__verbose_feature_names_out': True,
 'prep__num': Pipeline(steps=[('imputer',
SimpleImputer(strategy='median'))]),
 'prep__cat': Pipeline(steps=[('imputer',
SimpleImputer(strategy='most_frequent')),
                  ('ohe',
                   OneHotEncoder(handle_unknown='ignore',
sparse_output=False))]),
 'prep__num__memory': None,
 'prep__num__steps': [('imputer', SimpleImputer(strategy='median'))],
 'prep__num__transform_input': None,
 'prep__num__verbose': False,
 'prep__num__imputer': SimpleImputer(strategy='median'),
 'prep__num__imputer__add_indicator': False,
 'prep__num__imputer__copy': True,
 'prep__num__imputer__fill_value': None,
 'prep__num__imputer__keep_empty_features': False,
 'prep__num__imputer__missing_values': nan,
 'prep__num__imputer__strategy': 'median',
```

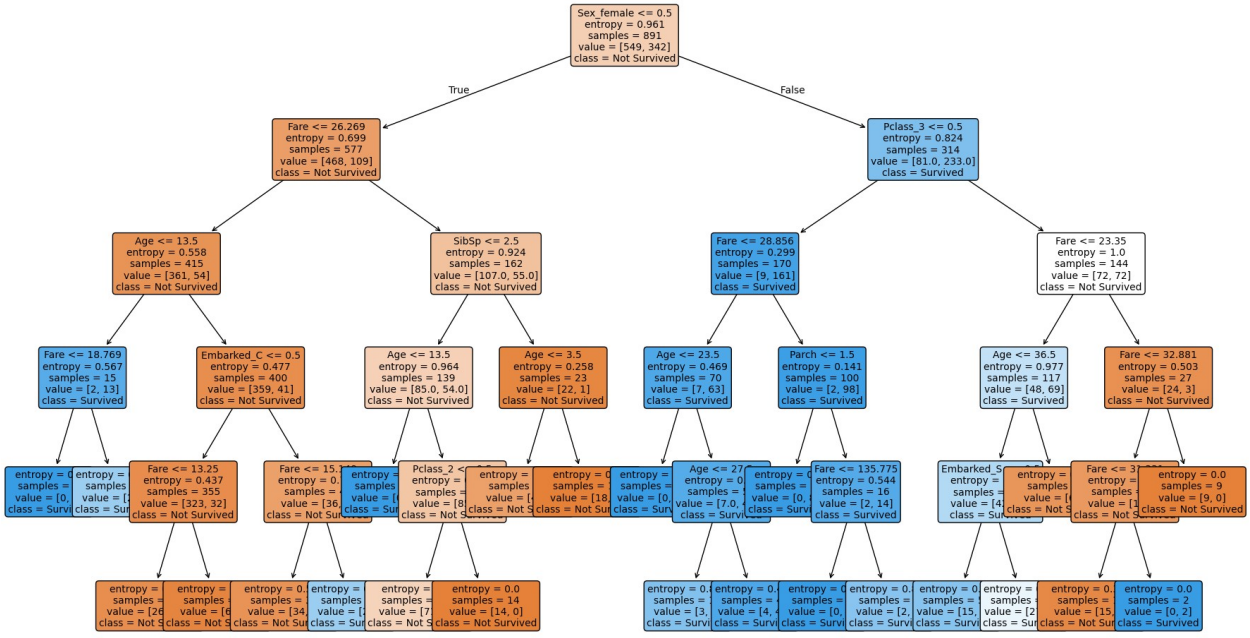
```
 'prep__cat__memory': None,
 'prep__cat__steps': [('imputer',
SimpleImputer(strategy='most_frequent')),
  ('ohe', OneHotEncoder(handle_unknown='ignore',
sparse_output=False))],
 'prep__cat__transform_input': None,
 'prep__cat__verbose': False,
 'prep__cat__imputer': SimpleImputer(strategy='most_frequent'),
 'prep__cat__ohe': OneHotEncoder(handle_unknown='ignore',
sparse_output=False),
 'prep__cat__imputer__add_indicator': False,
 'prep__cat__imputer__copy': True,
 'prep__cat__imputer__fill_value': None,
 'prep__cat__imputer__keep_empty_features': False,
 'prep__cat__imputer__missing_values': nan,
 'prep__cat__imputer__strategy': 'most_frequent',
 'prep__cat__ohe__categories': 'auto',
 'prep__cat__ohe__drop': None,
 'prep__cat__ohe__dtype': numpy.float64,
 'prep__cat__ohe__feature_name_combiner': 'concat',
 'prep__cat__ohe__handle_unknown': 'ignore',
 'prep__cat__ohe__max_categories': None,
 'prep__cat__ohe__min_frequency': None,
 'prep__cat__ohe__sparse_output': False,
 'clf__ccp_alpha': 0.0,
 'clf__class_weight': None,
 'clf__criterion': 'entropy',
 'clf__max_depth': 5,
 'clf__max_features': None,
 'clf__max_leaf_nodes': None,
 'clf__min_impurity_decrease': 0.0,
 'clf__min_samples_leaf': 2,
 'clf__min_samples_split': 10,
 'clf__min_weight_fraction_leaf': 0.0,
 'clf__monotonic_cst': None,
 'clf__random_state': 42,
 'clf__splitter': 'best'}
```

```
# Extract feature names after preprocessing (for readable plot)
ohe =
best_dt.named_steps["prep"].named_transformers_["cat"].named_steps["oh
e"]
cat_names = ohe.get_feature_names_out(cat_cols)
feature_names = list(num_cols) + list(cat_names)

dt_model = best_dt.named_steps["clf"]

plt.figure(figsize=(18, 10))
plot_tree(
    dt_model,
```

```
    filled=True,
    feature_names=feature_names,
    class_names=["Not Survived","Survived"],
    rounded=True,
    fontsize=10
)
plt.tight_layout()
plt.show()
```




```
from sklearn.model_selection import StratifiedKFold, cross_val_score

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

scores = cross_val_score(
    estimator=best_dt,
    X=X, y=y,
    cv=cv,
    scoring="accuracy",
    n_jobs=-1
)

print("Fold accuracies:", [round(s, 4) for s in scores])
print(f"Average accuracy: {scores.mean():.4f}")
print(f"Std dev: {scores.std():.4f}")
```

```
Fold accuracies: [np.float64(0.838), np.float64(0.8258),
np.float64(0.7978), np.float64(0.8202), np.float64(0.8371)]
```

```
Average accuracy: 0.8238
Std dev: 0.0146
```

```
from sklearn.ensemble import RandomForestClassifier

# Creating Random Forest Pipeline

pipe_rf = Pipeline(steps=[
    ("prep", preprocess),
    ("clf", RandomForestClassifier(random_state=42, n_jobs=-1))
])

# Defining parameter grid for Random Forest

param_grid_rf = {
    "clf__n_estimators": [200, 400, 800],
    "clf__max_depth": [None, 6, 10, 14],
    "clf__min_samples_split": [2, 5, 10],
    "clf__min_samples_leaf": [1, 2, 4],
    "clf__max_features": ["sqrt", "log2", 0.5]  # 0.5 = half the
features
}

# Applying Grid Search CV for Random Forest

grid_rf = GridSearchCV(
    estimator=pipe_rf,
    param_grid=param_grid_rf,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
grid_rf.fit(X, y)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('prep',

ColumnTransformer(transformers=[('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median'))]),

['Age',

'SibSp',

'Parch',

'Fare']),
```

```
('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('ohe',

OneHotEncoder(handle_unknown='ignore',

sparse_output=False))]),

['Sex',

'Embarked',

'Pclass'])])),
                                          ('clf',

RandomForestClassifier(n_jobs=-1,

random_state=42))]),
             n_jobs=-1,
             param_grid={'clf__max_depth': [None, 6, 10, 14],
                         'clf__max_features': ['sqrt', 'log2', 0.5],
                         'clf__min_samples_leaf': [1, 2, 4],
                         'clf__min_samples_split': [2, 5, 10],
                         'clf__n_estimators': [200, 400, 800]},
             scoring='accuracy')

best_rf = grid_rf.best_estimator_
print("Best RF params:", grid_rf.best_params_)

Best RF params: {'clf__max_depth': None, 'clf__max_features': 0.5,
'clf__min_samples_leaf': 2, 'clf__min_samples_split': 2,
'clf__n_estimators': 200}

# Applying Stratified K-Fold CV to the best Random Forest model

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
rf_scores = cross_val_score(best_rf, X, y, cv=cv, scoring="accuracy",
n_jobs=-1)

print("RF fold accuracies:", [round(s, 4) for s in rf_scores])
print(f"RF average accuracy: {rf_scores.mean():.4f}")
print(f"RF std dev: {rf_scores.std():.4f}")

RF fold accuracies: [np.float64(0.8603), np.float64(0.8371),
np.float64(0.8202), np.float64(0.8258), np.float64(0.8427)]
RF average accuracy: 0.8372
RF std dev: 0.0140
```

For this task, we can see that the Random Forest has achieved an average accuracy of 83.72% and Decision Tree has received an average accuracy of 82.38%, this tells us that after the similar preprocessing steps we took to clean the data a bit, Random Forest performs a bit better than Decision Tree model because Random Forest tries to reduce the overfitting issue using techniques like bagging and feature sub-sampling.

Decision tree model tends to capture the dataset patterns but overfits but Random forest provides more stable and generalised results.

Therefore while the average accuracy is not that different, we can still conclude that Random Forest is a better algorithm than Decision Tree for this task using the preprocessing pipeline we created.

ASU ID - 1233380696
Name - Atharva Chunduwar

DM - HW 2

Task 2

a) The training error simply tells us about the rate of misclassified samples.

$$\text{Training error} = \frac{\text{total misclassified samples}}{\text{total samples}}$$

For this example, the prediction of the leaf is whichever class count is higher. The other class count is misclassified.

∴ We have 6 leaf nodes: D_0, D_1, E_0, E_1, C_0, C_1.

Nodes	Misclassification #
D_0	5
D_1	6
E_0	2
E_1	6
C_0	5
C_1	5

Total misclassifications = 29.
Total samples = (14+5) + (6+7) + (2+10) + (8+6) + (22+20)
= 100.

∴ Training error rate = $\frac{29}{100}$ = 29 %.

b) For the test instance : {A=0, B=1, C=1, D=1, E=0}.

We start at the root node and traverse the tree according to the labeles for each nodes.

Root (A) → 0 → go left side to B.
Node (B) → 1 → go right side to E.
Node (E) → 0 → go left to leaf node.

We get (+ : 2 , − : 10).

∴ the decision tree will assign this test instance T to "−" class.

Task 3

Q1) We are given 10 samples, out of which we have 4 positive samples and 6 negative ones.

$$Gini = 1 - \Sigma (p_i)^2$$

where p_i is the probability of each class label.

$(p_+) = 4/10 = 0.4 \mid (p_-) = 6/10 = 0.6$

$$\therefore Gini = 1 - [(0.4)^2 + (0.6)^2] = 1 - 0.16 - 0.36$$

∴ Gini = 0.48 before splitting.

Q2) We have 7 true samples and 3 false samples for attribute A.

Now we will calculate $Gini_T$ & $Gini_F$.

For A = true (7 samples) +:4, -:3
so $(h_+) = 4/7$, $(h_-) = 3/7$.

$$\therefore Gini_T = 1 - \left[\left(\frac{4}{7}\right)^2 + \left(\frac{3}{7}\right)^2\right] = \frac{49}{49} - \frac{25}{49}$$

$$= 24/49$$

For A = false (3 samples) +:3, -:0
$\therefore h(+) = 1$, $(h_-) = 0$

$$\therefore Gini_F = 1 - [(1)^2 + (0)^2] = 0$$

Now we will calculate weighted gini after split.

$$\therefore Gini_{split} = \frac{7}{10}(Gini_T) + \frac{3}{10}(Gini_F)$$

$$\therefore Gini_{split} = \frac{7}{10} \cdot \frac{24}{49} = \underline{0.34}.$$

Now, $Gain = Gini_{before} - Gini_{after\ split}$

$\therefore Gain = 0.48 - 0.34 = \underline{0.14}$.

$\therefore$ Gini gain after splitting on attribute A is $\underline{0.14}$.

(4)

Q3) For attribute B, true → 4 samples
false → 6 samples

For B = true, + : 3, − : 1

$$\therefore Gini_T = 1 - \left[\left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2 \right] = 1 - \frac{10}{16}$$

$$= \frac{6}{16} = 0.375$$

For B = false, + : 1, − : 5

$$\therefore Gini_F = 1 - \left[\left(\frac{1}{6}\right)^2 + \left(\frac{5}{6}\right)^2 \right]$$

$$= 0.2778$$

Weighted gini after split

$$Gini_{split} = \frac{4}{10}(Gini_T) + \frac{6}{10}(Gini_F)$$

$$= \frac{4}{10}(0.375) + \frac{6}{10}(0.2778)$$

$$= 0.3167.$$

$$Gini\ gain = Gini_{before} - Gini_{after\ split}$$

$$= 0.48 - 0.3167 = 0.1633$$

∴ Gini gain after splitting on B is 0.1633.

Q4) We pick the splitting attribute which has larger gini gain.

As we calculated, gini gain (A) = 0.137
& gini gain (B) = 0.163

∴ We will choose or decision tree will choose B as the root split as it will give purer children and lower impurity than A.

Task 4

Q1) No decision tree classifiers are not linear classifiers. A linear classifier is the one which tries to separate the data using a straight line. On the other hand, decision tree tries to split the data in step by step manner using different conditions. Therefore, because of these multiple cuts, decision trees can have non-linear patterns in the data and the final boundary is not a straight line.

Q2) Gini index is a better splitting criterion than misclassification error.

The misclassification error criterion only checks how many samples are wrongly classified but it does not show how pure a node is.

But gini index gives importance to this, it becomes smaller as the node get purer.

Thus gini index helps the decision tree find better and cleaner splits.

Task 05

Bagging Weaknesses:

Bagging creates many decision trees using random samples of data and then combines all the results. But a big weakness is that all the trees use same features when splitting.

Because of this, trees may become correlated and result in similar decisions. this reduces the benefit of combining them.

Difference between Bagging & Random Forest

Random forest is improved version of bagging. In random forest, at each split in tree, the algorithm chooses a random subset of features instead of using all features, this makes every tree slightly different.

Why this helps:

The random feature selection of trees in random forest generates less correlated and more diverse trees. As a result the final prediction is more stable and accurate.

(9)

Task 7

Circle equation with (a, b) as center and radius r: $(x_1 - a)^2 + (x_2 - b)^2 - r^2 = 0$

$$x_1^2 - 2ax_1 + a^2 + x_2^2 - 2bx_2 + b^2 - r^2 = 0$$

$$\therefore (x_1^2 + x_2^2) + (-2a)x_1 + (-2b)x_2 + (a^2 + b^2 - r^2) = 0$$

Given the feature map $\phi(x) = (x_1, x_2, x_1^2, x_2^2)$
In this feature space, the expanded circle is a linear equation of the form $w \cdot \phi(x) + b = 0$

Where, $w = (-2a, -2b, 1, 1)$ & $b = (a^2 + b^2 - r^2)$

→ The points present inside the circle satisfy $w \cdot \phi(x) + b < 0$.

→ Points outside circle satisfy $w \cdot \phi(x) + b > 0$.

→ Points on the boundary : $w \cdot \phi(x) + b = 0$.

∴ We can conclude that in the feature space (x_1, x_2, x_1^2, x_2^2), every circular region is separated from the rest of the plane by a hyperplane. Thus circular regions are linearly separable in this space.

(8)

Task 6

$\phi(x_1, x_2) = (z_1, z_2) = (x_1, x_1 x_2)$

Given the points and labels: $z_1 = x_1$, $z_2 = x_1 x_2$

(x_1, x_2)	label	(z_1, z_2)
$(-1, -1)$	−	$(-1, +1)$
$(-1, +1)$	+	$(-1, -1)$
$(+1, -1)$	+	$(+1, -1)$
$(+1, +1)$	−	$(+1, +1)$

From the table, we can see that:

- All positives lie at $z_2 = -1$
- All negatives lie at $z_2 = +1$

∴ We can use the best separating line as halfway between them.

∴ Now we can use:

if $z_2 > 0$ → predicts negative.

if $z_2 < 0$ → predicts positive.

All 4 points lie at distance 1 from the boundary so all four are support vectors.

After mapping (x_1, x_2) to $(x_1, x_1 x_2)$ positives are at $z_2 = -1$ and negatives at $z_2 = +1$.

→ the maximal margin separator is the horizontal line $z_2 = 0$.

→ $w = (0, 1)$, $b = 0$ gives the boundary.

→ Margin = 1 and all the 4 points are support vectors.

(2)

Task 8

Ellipse equation: $c(x_1 - a)^2 + d(x_2 - b)^2 - 1 = 0$

$c x_1^2 - 2acx_1 + ca^2 + dx_2^2 - 2bdx_2 + b^2d - 1 = 0$

The given feature vector: $\phi(x) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$.

If we use expanded ellipse equation, we see that it is in quadratic form in (x_1, x_2). Thus we can express it as $\phi(x)$. $\Rightarrow$ $w \cdot \phi(x) = 0$

here with suitable weights w, x_1, x_2 weight can be 0 for axis aligned ellipses and non-zero for not axis aligned. Thus we can say that any elliptic boundary is a hyperplane in this feature space.

The degree-2 kernel is $K(u, v) = (1 + u \cdot v)^2$
This equals a dot product in quadratic feature above: $(1 + u \cdot v)^2 = \phi(u) \cdot \phi(v)$

For a mapping equivalent to $(1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$ scaling doesn't change linear separability.

Therefore a SVM using $K(u, v) = (1 + u \cdot v)^2$ is same as a linear SVM in the quadratic features $(1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

Github code link:
https://github.com/AtharvaBOT7/CSE572-DM-HW2-1233380696