Who is the real Winner?

Vetcha Pankaj Nath

professor: Indranil Saha

Abstract—*India, the world's largest democracy, holds regular state and union territory (UT) elections that play a crucial role in shaping the country's political landscape.This study explores the educational backgrounds of election winners in India.The primary objective is to develop a machine learning model to predict the education level of the winning candidates based on various parameters*

Contents

1. Introduction

In this report, we present a machine learning model for predicting the Education Level of the Winners of the State Elections in India using a *multi-class classification* approach. The dataset is sourced from the Election Commission of India website and contains various features related to the election winners across different states and union territories (UTs).

2. Dataset

2.1. Dataset Description

The dataset contains the following features:

1. **ID** - Serial ID for the candidate.
2. **Candidate** - Name of the winning candidate.
3. **Constituency** ∇ - Constituency from where the candidate won.
4. **Party** - Political Party to which the candidate belongs.
5. **Criminal Case** - Total number of criminal cases on the candidate.
6. **Total Assets** - Total assets declared by the candidate.
7. **Liabilities** - Liabilities declared by the candidate.
8. **Education** - Education Level of the candidate. (The target variable)

2.2. Data Preprocessing

2.2.1. Converting Liabilities and Total Assets

```python
# Define conversion factors for different units to Crore
conversion_factors = {
    'Crore+': 1,
    'Lac+': 0.01,
    'Thou+': 0.0001,
    'Hund+': 0.00001,
    "0": 0,
}

# Function to convert values to Crore
def convert_to_crore(value, unit):
    factor = conversion_factors.get(unit, None)
    if factor is not None:
        return value * factor
    else:
        raise ValueError("Conversion factor for unit '{}' is not defined.".format(unit))

# Apply conversion to selected column
def convert_column(value):
    parts = value.split()
    amount = float(parts[0])
    unit = parts[-1]
    return convert_to_crore(amount, unit)

# Convert 'Total Assets' column to Crore
trainData['Total Assets'] = trainData['Total Assets'].apply(convert_column)
testData['Total Assets'] = testData['Total Assets'].apply(convert_column)

# Convert 'Liabilities' column to Crore
trainData['Liabilities'] = trainData['Liabilities'].apply(convert_column)
testData['Liabilities'] = testData['Liabilities'].apply(convert_column)

# Scale the 'Total Assets' and 'Liabilities' columns between 0 and 1
trainData["Total Assets"]=trainData["Total Assets"]/trainData["Total Assets"].max()
testData["Total Assets"]=testData["Total Assets"]/testData["Total Assets"].max()

trainData["Liabilities"]=trainData["Liabilities"]/trainData["Liabilities"].max()
testData["Liabilities"]=testData["Liabilities"]/testData["Liabilities"].max()
```

Figure 1. Code

transformation

before

	Total Assets	Liabilities
0	211 Crore+	2 Crore+
1	1 Crore+	0
2	7 Crore+	22 Lac+
3	9 Crore+	24 Lac+
4	2 Crore+	61 Lac+

after

	Total Assets	Liabilities
0	0.166535	0.002270
1	0.000789	0.000000
2	0.005525	0.000250
3	0.007103	0.000272
4	0.001579	0.000692

Figure 2. Code

2.2.2. One Hot Encoding states and parties using mapping

```python
# Store the unique values of 'state' and 'Party' columns
total_states = trainData["state"].unique()
total_parties = trainData["Party"].unique()

# Create total_states columns
for state in total_states:
    trainData[state] = (trainData["state"] == state).astype(bool)
    testData[state] = (testData["state"] == state).astype(bool)

# Create total_parties columns
for party in total_parties:
    trainData[party] = (trainData["Party"] == party).astype(bool)
    testData[party] = (testData["Party"] == party).astype(bool)
```

Figure 3. Code

transformation

before

	state	Party
0	TAMIL NADU	DMK
1	MADHYA PRADESH	BJP
2	KARNATAKA	INC
3	BIHAR	BJP
4	WEST BENGAL	BJP

after

	TAMIL NADU	MADHYA PRADESH	KARNATAKA	BIHAR	WEST BENGAL	...	Sikkim Krantikari Morcha	JD(U)	JMM	JD(S)	Tipra Motha Party
0	True	False	False	False	False	...	False	False	False	False	False
1	False	True	False	False	False	...	False	False	False	False	False
2	False	False	True	False	False	...	False	False	False	False	False
3	False	False	False	True	False	...	False	False	False	False	False
4	False	False	False	False	True	...	False	False	False	False	False

Figure 4. Code

2.2.3. Encoding Education column of trainData

```
mapper = {'Others':0 ,'Literate': 1, '5th Pass': 2, '8th Pass': 3, '10th Pass': 4, '12th Pass': 5,
           'Graduate': 6, 'Post Graduate': 7, 'Graduate Professional': 8, 'Doctorate': 9}
reverse_mapper = {v: k for k, v in mapper.items()}
trainData['Education'] = trainData['Education'].map(mapper)
```

Figure 5. Code

transformation

before

	ID	Education
0	0	8th Pass
1	1	12th Pass
2	2	Post Graduate
3	3	Post Graduate
4	4	8th Pass

after

	ID	Education
0	0	3
1	1	5
2	2	7
3	3	7
4	4	3

Figure 6. Code

3. Method 1

Using the K-Nearest Neighbors (KNN) algorithm from the Scikit-learn library to perform a classification task.

```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.01, random_state=42)

# Define the parameter
param = {
    'n_neighbors': range(1, 100)
}

# Create the model object
model_knn = KNeighborsClassifier()

# Get the test set F1-scores for each n_neighbors value
test_f1_scores = []
for n in param['n_neighbors']:
    model_knn.set_params(n_neighbors=n,n_jobs=-1)
    model_knn.fit(X_train, y_train)
    y_pred = model_knn.predict(X_test)
    test_f1_scores.append(f1_score(y_test, y_pred, average='weighted'))
```

Figure 7. Code

Summary

Manually iterating over a range of potential values for the 'n_neighbors' parameter in the K-Nearest Neighbors (KNN) classifier
1. fitting the model on `X_train`
2. making predictions on `X_test`
3. and calculating the F1-score for each value

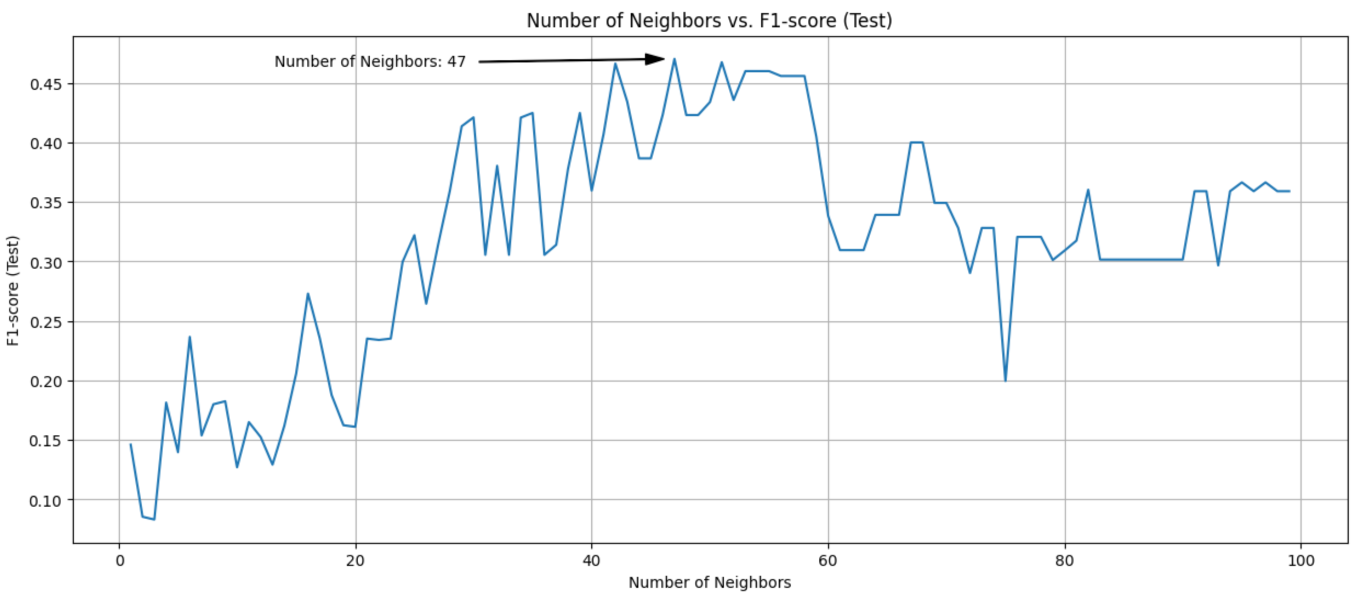

Figure 8. Your plot caption here

3.1. Hyper-parameters

- `n_neighbours=47`
- `average=`'*weighted*' for testing f1_score

4. Method 2

Reimplimented "*A Simple Approach to Ordinal Classification*" [1] using LogisticRegression rom the Scikit-learn library to perform a classification task.

4.1. Further One Hot Encoding "Education" column

```python
# Get unique entries in 'Education' column
unique_entries = sorted(trainData['Education'].unique())

# Create new columns based on unique entries
for i, entry in enumerate(unique_entries[:-1]):  # We exclude the last entry as there's no greater value
    trainData[f'Education_gt_{entry}'] = (trainData['Education'] > entry).astype(int)
```

Figure 9. Code

before

	Education
0	3
1	5
2	7
3	7
4	3

after

	Education_gt_0	Education_gt_1	Education_gt_2	Education_gt_3	Education_gt_4	Education_gt_5	Education_gt_6	Education_gt_7	Education_gt_8
0	1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0
2	1	1	1	1	1	1	1	0	0
3	1	1	1	1	1	1	1	0	0
4	1	1	1	0	0	0	0	0	0

Figure 10. Code

Summary

We generate new Pseudo Class columns in the 'trainData' DataFrame based on the unique entries in the 'Education' column. These new columns indicate whether the 'Education' value is greater than each unique entry i.e. If the 'Education' value is greater than the current entry, the corresponding cell in the new column is set to 1.

For a particular cell in $Education\_gt\_i = 1$ implies that , that cell has 'Education' value greater than i

4.2. Training individual Models for each pseudo class created

```python
from sklearn.linear_model import LogisticRegression

# Selecting features and target variable
features = trainData.copy()

temparr1 = [f'Education_gt_{entry}' for entry in unique_entries[:-1]]
temparr2 = temparr1 + ['ID','Candidate','Constituency ∇', 'Party', 'state', 'Education']

features.drop(temparr2, axis=1, inplace=True)
target = trainData[temparr1]

# List to store models
models = []
f1_scores = []

# Create and train a model for each 'Education_gt_' column
for i, entry in enumerate(unique_entries[:-1]):
    # Get target column
    target_col = f'Education_gt_{entry}'

    # Create a logistic regression model
    model = LogisticRegression(max_iter=1000)

    # Fit the model
    model.fit(features, target[target_col])

    # Store the model
    models.append(model)
```

Figure 11. Code

4.3. Predicting Probabilities for each pseudo class

```python
test_features = testData.copy()

temparr3 = ['ID','Candidate','Constituency ∇', 'Party', 'state']

test_features.drop(temparr3, axis=1, inplace=True)

# Create a DataFrame to store probabilities
probabilities = pd.DataFrame()

# Predict probabilities for each model
for i, model in enumerate(models):
    # feature names from the data used for predictions
    X_final = test_features

    # Predict probabilities
    proba = model.predict_proba(X_final)

    # Get the probability of the positive class
    proba = proba[:, 1]

    # Store the probabilities in the DataFrame
    probabilities[f'Education_gt_{unique_entries[i]}'] = proba
```

Figure 12. Code

	Education_gt_0	Education_gt_1	Education_gt_2	Education_gt_3	Education_gt_4	Education_gt_5	Education_gt_6	Education_gt_7	Education_gt_8
0	0.994839	0.994247	0.991913	0.979038	0.785092	0.550997	0.343892	0.137773	0.010802
1	0.996634	0.996926	0.996936	0.948478	0.853159	0.631902	0.382654	0.185620	0.019809
2	0.994542	0.993968	0.991883	0.978846	0.798083	0.572906	0.305315	0.175801	0.010504
3	0.996634	0.996926	0.996936	0.948476	0.853163	0.631896	0.382634	0.185638	0.019810
4	0.997524	0.996769	0.997040	0.990566	0.933511	0.765933	0.513721	0.198188	0.008913

Figure 13. Code

4.4. Predicting best Class from Pseudo Class probabilities

```python
# Create a new DataFrame 'final' with one more column than 'probabilities'
final_cols = list(probabilities.columns) + [f'Education_gt_{unique_entries[-1]}']
final = pd.DataFrame(index=probabilities.index, columns=final_cols)

# Set values for the first column
final.iloc[:, 0] = 1 - probabilities.iloc[:, 0]

# Set values for the intermediate columns
for i in range(1, len(probabilities.columns)):
    final.iloc[:, i] = probabilities.iloc[:, i-1] - probabilities.iloc[:, i]

# Set values for the last column
final.iloc[:, -1] = probabilities.iloc[:, -1]

# Create a final Predictions DataFrame
final_df=pd.DataFrame()
final_df["ID"]=testData["ID"] # changed
final_df["Education"] = final.idxmax(axis=1).to_frame()

# Reverse the mapper dictionary
reverse_mapper = {v: k for k, v in mapper.items()}

# Modify the keys in the reverse_mapper dictionary
modified_mapper = {f'Education_gt_{k}': v for k, v in reverse_mapper.items()}

# Map the entries of the 'Education' column using the modified mapper
final_df['Education'] = final_df['Education'].map(modified_mapper)
```

Figure 14. Code

final

	Education_gt_0	Education_gt_1	Education_gt_2	Education_gt_3	Education_gt_4	Education_gt_5	Education_gt_6	Education_gt_7	Education_gt_8	Education_gt_9
0	0.005161	0.000592	0.002334	0.012875	0.193945	0.234095	0.207105	0.206119	0.126971	0.010802
1	0.003366	-0.000292	-0.00001	0.048457	0.095319	0.221257	0.249248	0.197033	0.165811	0.019809
2	0.005458	0.000574	0.002085	0.013036	0.180763	0.225177	0.267591	0.129514	0.165297	0.010504
3	0.003366	-0.000292	-0.00001	0.048459	0.095314	0.221267	0.249263	0.196995	0.165828	0.01981
4	0.002476	0.000754	-0.000271	0.006474	0.057056	0.167578	0.252212	0.315533	0.189275	0.008913

final_df

	ID	Education
0	0	12th Pass
1	1	Graduate
2	2	Graduate
3	3	Graduate
4	4	Post Graduate

Figure 15. Code

5. Data Insights

5.1. Relation between Parties and Criminal Cases

Percentage distribution of top 30% candidates (based on Criminal Cases in decreasing order) in various parties.

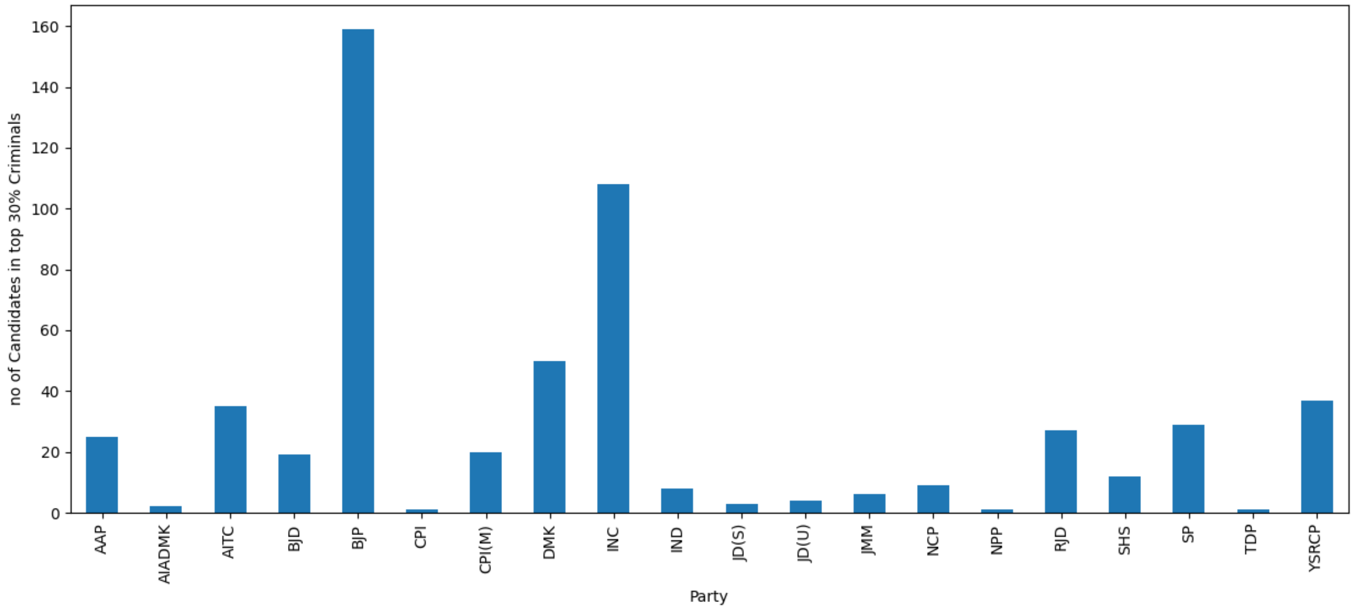

Figure 16. Your plot caption here

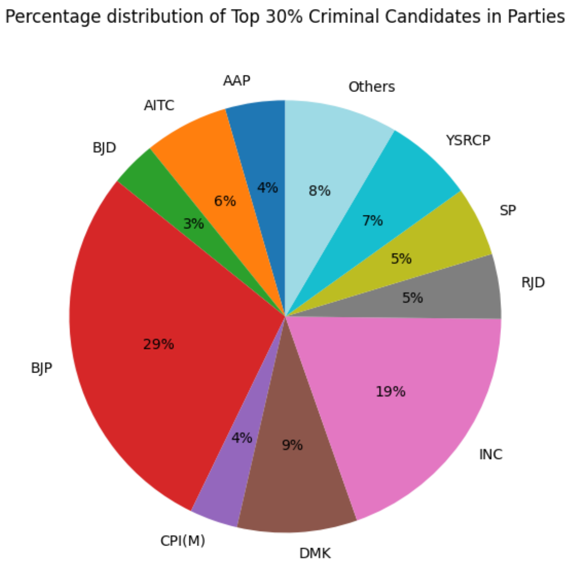

Figure 17. Your plot caption here

5.2. Relation between Parties and Total Assets

Percentage distribution of top 30% candidates (based on Total Assets in decreasing order) in various parties.

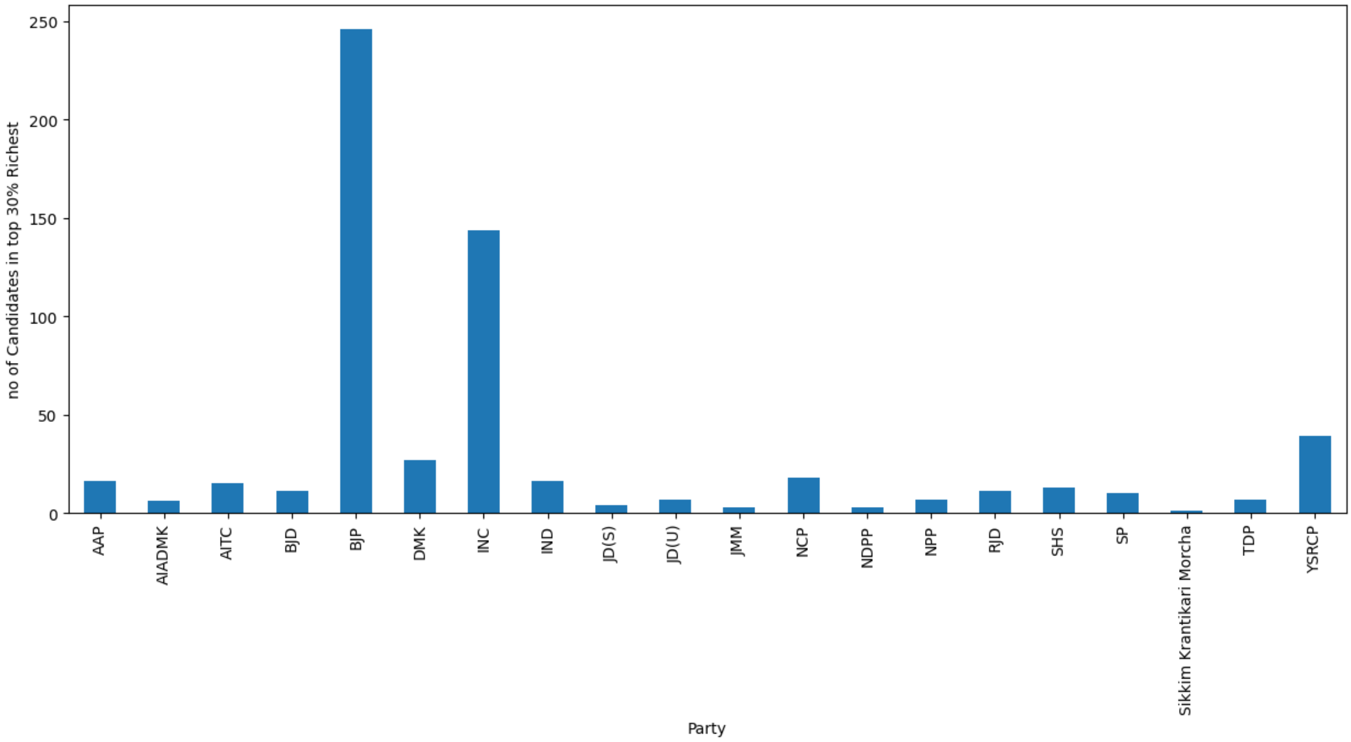

Figure 18. Your plot caption here

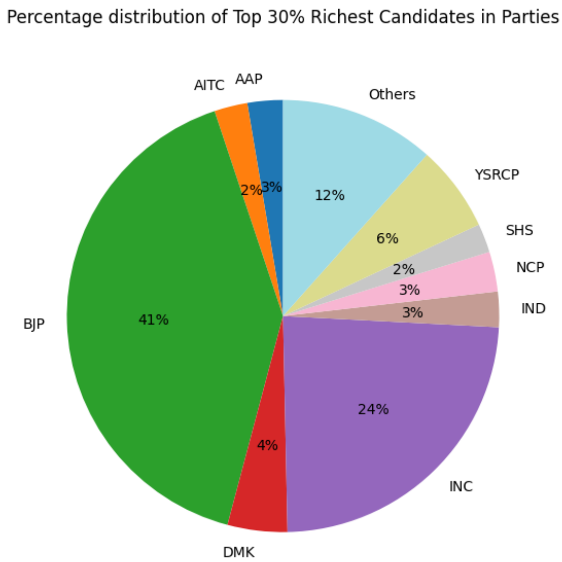

Figure 19. Your plot caption here

5.3. Relation between state and Education

distribution of education levels for each state/UT:

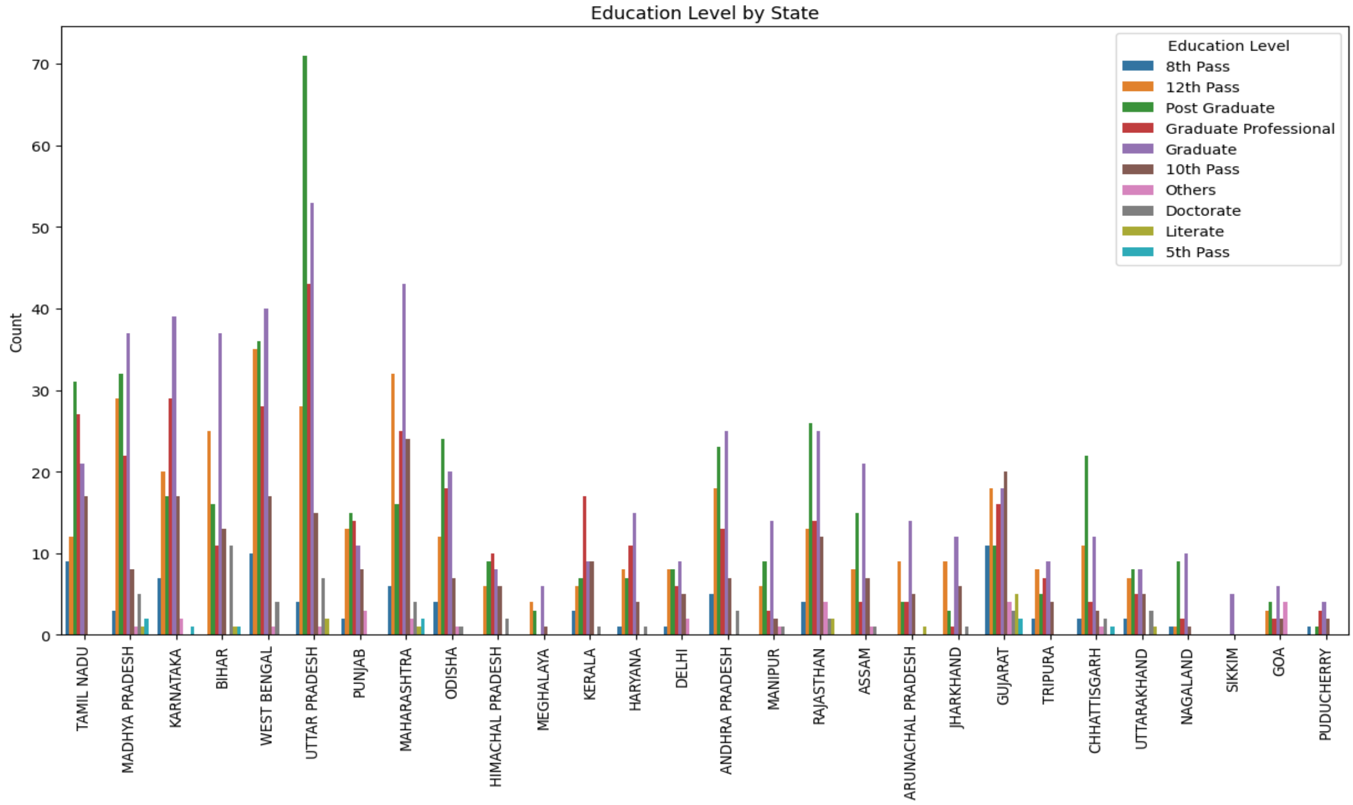

Figure 20. Your plot caption here

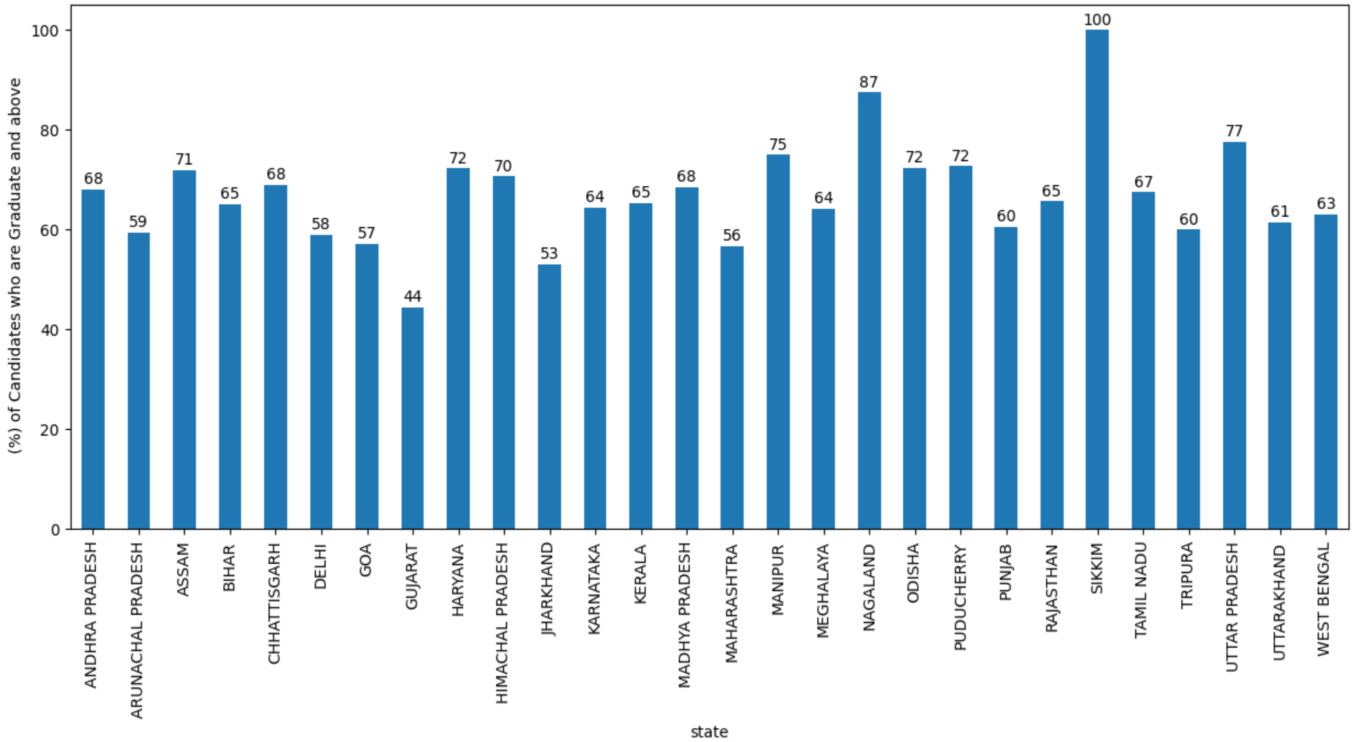

Figure 21. Your plot caption here

5.4. Relation between Party and Education

distribution of education levels for each state/UT:

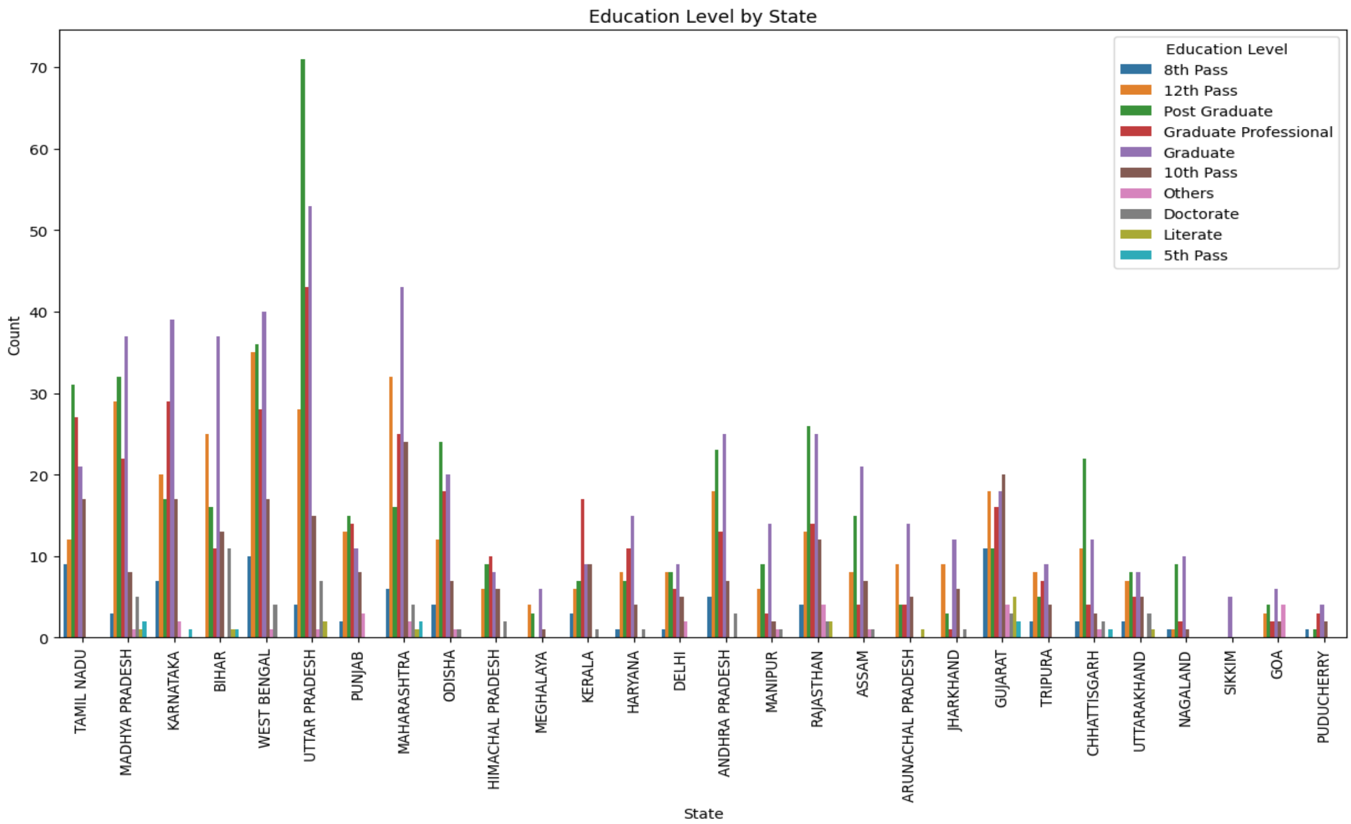

Figure 22. Your plot caption here

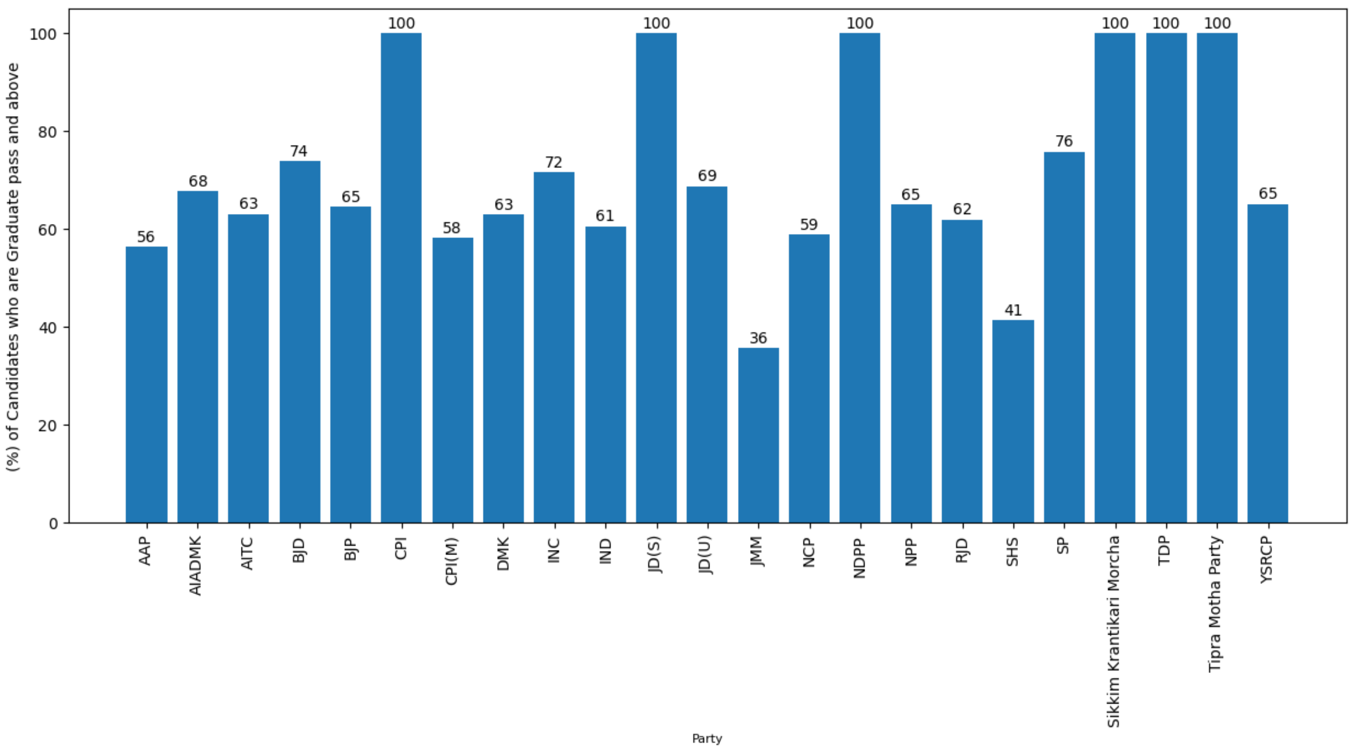

Figure 23. Your plot caption here

6. Results

- **Public f1_score:** 0.24334
- **Private f1_score:** 0.23607
- **Public Leaderboard Rank:** 82
- **Private Leaderboard Rank:** 85

GitHub Repo
https://github.com/might-guy106/CS253-Assignment-3
pankajnath1724@gmail.com

References

[1] E. Frank and M. Hall, "A Simple Approach to Ordinal Classification", 2001. DOI: http://old-www.cms.waikato.ac.nz/~eibe/pubs/ordinal_tech_report.pdf.

[2] Anthropic, "Claude: An AI assistant", 2023. [Online]. Available: https://www.claude.ai.

[3] OpenAI, "Chatgpt: Conversational ai", 2022. [Online]. Available: https://openai.com/blog/chatgpt/.