CMPE 462: Machine Learning
Term Project

Nazlıcan Aka - 2020400027
Aslı Gök - 2020400189
Dağlar Eren Tekşen - 2020400111

December 22, 2024

Contents

1 Task 1: Data Collection and Feature Extraction **2**
- 1.1 Data Collection Procedure . . . 2
  - 1.1.1 Binary Classification . . . 4
- 1.2 Future Extraction . . . 4
- 1.3 Number of categories, samples, dimensions . . . 4
- 1.4 Data Pre-processing . . . 4
- 1.5 K-Means Clustering . . . 5

2 Task 2: Supervised Learning **7**
- 2.1 Logistic Regression . . . 7
- 2.2 Support Vector Machines . . . 8
  - 2.2.1 Linear Soft-Margin SVM From Scratch . . . 8
  - 2.2.2 Scikit-Learn's SVM Function for Linear and Non-Linear SVMs . . . 8
  - 2.2.3 Hyperparameter Tuning using 5-Fold Cross-Validation . . . 8
- 2.3 Random Forest . . . 9
- 2.4 K-Nearest Neighbors . . . 10
- 2.5 Model Comparison . . . 11
- 2.6 Evaluation Metrics . . . 11
- 2.7 Challenges . . . 12

3 Task 3: Plotting Decision Boundary **13**

4 Individual Contributions **15**

5 CODE **16**

6 REFERENCES **17**

1. Task 1: Data Collection and Feature Extraction

1.1 Data Collection Procedure

We have firstly chosen a dataset from protein sequences. However, the protein sequence dataset was quite long and hard to understand. Unfortunately, the protein dataset was quite complicated and got out of the scope of this course in terms of its difficulty of feature extraction. Therefore, we decided to change our dataset to a less complicated and enjoyable dataset which is movies dataset.

We have investigated many databases.Our expectations from the data are as follows: it should contain a sufficient number of numerical values, be self-explanatory, allow for easy labeling, and, in summary, be suitable for a machine learning project. As a result, we chose the dataset "The Open Movie Database", aka OMDb API[2]. It gives an API key for each different email. The API key can fetch at most 1000 data from the API in a day for free.

Data is stored in JSON format. There are different ways of fetching data, but we used titles of films. We get thousands of film titles from Wikidata. We have used a SPARQL query to fetch data from Wikidata.[1]

Here is our SPARQL query used for fetching the film titles from Wikidata :

```
PREFIX wikibase: <http://wikiba.se/ontology#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX v: <http://www.wikidata.org/prop/statement/>

SELECT ?film ?filmLabel WHERE {
  ?film wdt:P31 wd:Q11424.
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}

LIMIT 4000
```

We submitted this query to wikidata query service. Then, downloaded the query results in .csv format. The resulting .csv file is submitted in the project code files named "query.csv".

Then, we got the titles from the resulting .csv file coming from the query. Titles are under the column "filmlabels", then we send API requests and download the response for each title. As a result, we created the final .csv file with all film data.

Here is an example from our .json format film data:
Request: http://www.omdbapi.com/?t=we+live+in+public
Response:


```
{
    "Title": "We Live in Public",
    "Year": "2009",
    "Rated": "Unrated",
    "Released": "03 Sep 2010",
    "Runtime": "91 min",
    "Genre": "Documentary",
    "Director": "Ondi Timoner",
    "Writer": "Ondi Timoner",
    "Actors": "Josh Harris, Tom Harris, Carlos Alvarez",
    "Plot": "A documentary focusing on the life of dot-com entrepreneur Josh Harris,
    "Language": "English",
    "Country": "United States",
    "Awards": "2 wins & 3 nominations",
    "Poster": "https://m.media-amazon.com/images/M/MV5BMjE4MjE2MzA3MF5BMl5BanBnXkFtZT
    "Ratings": [
        {
            "Source": "Internet Movie Database",
            "Value": "7.1/10"
        },
        {
            "Source": "Rotten Tomatoes",
            "Value": "81%"
        },
        {
            "Source": "Metacritic",
            "Value": "69/100"
        }
    ],
    "Metascore": "69",
    "imdbRating": "7.1",
    "imdbVotes": "2,002",
    "imdbID": "tt0498329",
    "Type": "movie",
    "DVD": "N/A",
    "BoxOffice": "$41,711",
    "Production": "N/A",
    "Website": "N/A",
    "Response": "True"
}
```

1.1.1 Binary Classification

Binary classification is simple and easy to understand as a beginner in machine learning. Also, our data is quite suitable to use for binary classification. Therefore, we chose binary classification to classify the movie data. The classification is to label a film good if the IMDB rating is greater than 7, otherwise label it bad. Because we are going to do supervised learning and unsupervised learning, we need targets. IMDB rating is available for most of the data, but for some data some of them are missing. Because our target is IMDB ratings, we fetched the data only with IMDB ratings.

1.2 Future Extraction

Firstly,we get the data with only meaningful features into a dataframe. They are title, year, runtime, genres, number of actors, languages, number of languages, countries, number of countries, wins, nominations, boxoffice, and finally the IMDB rating.

We removed some features because they are texts and need NLP such as Plot. Also, we removed Ratings, Metascore, and Rotten Tomatoes because they directly indicate the IMDB rating, which would make the learning process very easy.

Additionally, attributes with links are removed.

We did not use any feature extraction technique because the domain is understandable for us, we make some search on attributes and conclude to choose it as a feature or not.

1.3 Number of categories, samples, dimensions

Metric	**Value**
Number of Samples	3280
Categories	0 (Bad), 1 (Good)
Samples per Category	2156 (Bad), 1124 (Good)
Dimensionality of Raw Data	14

1.4 Data Pre-processing

We cleaned the data from unnecessary columns like Language and Country. Handle missing values in numerical features. Year, Runtime, Wins, Nominations, Boxoffice is filled with the median of their respective columns.

Additionally, we scale the numeric features to a range [0, 1] to normalize the data for better model performance.

First we did one-hot encoding for Genres, but there are more than 20 genres, so it increased the dimensionality so much. Then we removed Genres from our features and kept only the number of genres.

Then split the data into training and test sets. The ratio of train/test is 80/20 with a fixed random seed 42 for reproducibility.

1.5 K-Means Clustering

We have used Jaccard coefficient (Mahalanobis distance) as the similarity metric. Here are the results:

- Intra-Class Similarity (Mean): 0.3166154314245598
- Inter-Class Similarity (Mean): 0.279264504756399

Our Intra-Class Similarity mean is small, which means that there is variability within a category. The samples in a class are not so similar to each other, therefore, this adds complexity to the dataset. We tried to increase the intra-class similarity by changing features, but couldn't get satisfactory results.

Our Inter-class similarity mean is small, and its value is close to the intra-class similarity. We potentially have overlapping between clusters. This result adds difficulty to the model as well.

We also have class imbalance: the number of bad movies (2156) is greater than the number of good (1124) movies. This also makes the model more challenging. It is suitable for evaluating the performance of both clustering and classification algorithms. On the other hand, linear models may struggle due to overlapping clusters.

We implemented K-means clustering from scratch, whose implementation is present in the submitted code file.

Here are the plotted clustering results:

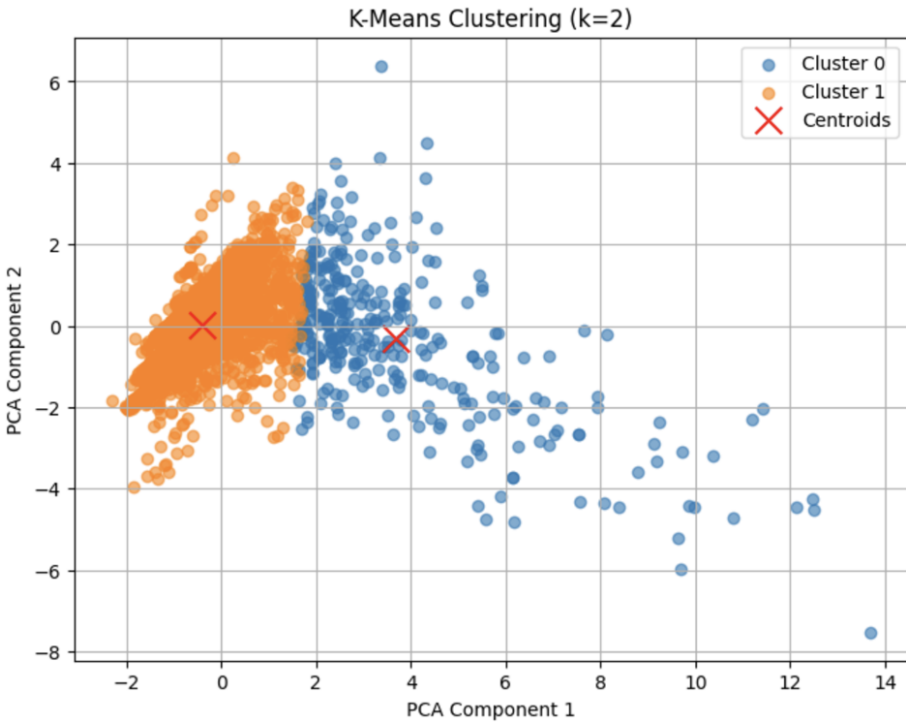

Figure 1.1: Clustering Results

When we compared the clustering assignments and our ground-truth target labels, we saw that the ratio of the number of samples that are classified in the same category for both in the cluster and target columns is 954 / 3280. This ratio (approximately 29 percent) indicates that our dataset is relatively difficult for applying unsupervised methods to perform this classification task.

2. Task 2: Supervised Learning

2.1 Logistic Regression

For this task, first we have trained a logistic regression classifier from scratch.

In the first version of our logistic regression the test accuracy was quite high, we concluded that overfitting occurs ,hence we implemented the logistic regression with regularization and updated the code. We have used L2 norm regularization.

We have defined our stopping criteria according to the loss differences between epochs. If the loss improvement between consecutive epochs is below a small threshold (in our case 1e-6) we stopped the training.

Model	**Train Accuracy**	**Test Accuracy**	**Training Time (s)**
From Scratch Logistic Regression	0.663328	0.675813	0.649038

Table 2.1: Accuracy and Runtime for From Scratch Logistic Regression

To compare the performance of our logistic regression classifier and Scikit-learn's logistic regression classifier, we have imported the Scikit-learn library and used LogisticRegression. We have trained the second model on our training set as well, then got the test/train accuracy and runtime values.

Model	**Train Accuracy**	**Test Accuracy**	**Training Time (s)**
Scikit-Learn's Logistic Regression	0.775697	0.792683	0.157849

Table 2.2: Accuracy and Runtime for From Scikit-Learn's Logistic Regression

Interpretation of these results:

When we looked at the test accuracies of both models we can say that the accuracy is greater in Scikit-learn's Logistic Regression, also note that our models' performance is not so bad but relatively small compared to Scikit-learn.

When we looked at the runtime values of both models, Scikit-learn's logistic regression works in a shorter amount of time compared to ours. It might be due to the efficient methods used in the Scikit-learn's. However, the time difference between the two is not so great since they both work in less than a second.

2.2 Support Vector Machines

2.2.1 Linear Soft-Margin SVM From Scratch

In the implementation of soft-margin SVM from scratch, we have used QP solver (from the `cvxopt` library).

The following expressions are fed to the solver:

Objective Function (P and q):

The goal is to minimize the following quadratic objective function:

$$\frac{1}{2}\mathbf{w}^T\mathbf{w} \quad st \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$

This is represented in the QP solver as:

- $P = \mathbf{X}_y\mathbf{X}_y^T$, where $\mathbf{X}_y = \mathbf{X} \cdot y$
- $q = -\mathbf{1}_m$, where m is the number of data points

Constraints (G, h, A, b):

- G represents the inequality constraints: $-\mathbf{I}_m \leq \alpha_i \leq C$ (soft margin penalty), where α_i are the Lagrange multipliers.
- h corresponds to the bounds for the slack variables: $\alpha_i \geq 0$ and $\alpha_i \leq C$, where C is the regularization parameter.
- A and b enforce the equality constraint $\sum_{i=1}^{m} \alpha_i y_i = 0$, ensuring the SVM's optimal hyperplane is properly constrained.

Solver: The solver is used to solve this quadratic programming problem and find the optimal α_i values. Then, the support vectors are identified by checking which α_i values are greater than a small threshold.

2.2.2 Scikit-Learn's SVM Function for Linear and Non-Linear SVMs

For this task,we used Scikit-Learn's `SVC` class to train both linear and non-linear SVMs:

Linear SVM:

- The default kernel of `SVC` is a linear kernel, which corresponds to a linear decision boundary. We have used this for the Linear SVM implementation.

Non-Linear SVMs:

- We also trained SVMs with non-linear transformations by specifying the kernel parameter as 'poly' (polynomial kernel) or 'rbf' (radial basis function kernel). These kernels allow the SVM to create non-linear decision boundaries.

2.2.3 Hyperparameter Tuning using 5-Fold Cross-Validation

For the hyperparameter tuning using 5-fold Cross-Validation we have imported and used Scikit-Learn's `GridSearchCV`:

Parameter Grid:

- The `param_grid` dictionary specifies the hyperparameters to tune:
  - C: Regularization parameter, with values [1, 10].
  - `kernel`: The kernel type, with options `['poly', 'rbf']`.
  - `gamma`: Kernel coefficient, with options `['scale', 'auto']`.

Grid Search:

The `GridSearchCV` object is used to perform a 5-fold cross-validation search over the specified hyperparameter grid, optimizing for accuracy. This is done using the `fit` method on the training data.

2.3 Random Forest

In random forest implementation, we have imported Scikit-learn's random forest function. We determined the number of trees using 5-fold cross validation. In order to do this, we gave 5 options for the number of trees in a list and traversed this list to find the one with the best result.

Here is the code snippet from our random forest function where we have done the 5-fold cross validation for the number of trees, which is also available in the code file we have submitted:

```
num_trees_options = [10, 50, 100, 200, 500]
best_score = 0
best_num_trees = 0

for num_trees in num_trees_options:
    rf = RandomForestClassifier(n_estimators=num_trees, random_state=42)
    scores = cross_val_score(rf, X_train, y_train, cv=5, scoring='accuracy')
    mean_score = scores.mean()
    if mean_score > best_score:
        best_score = mean_score
        best_num_trees = num_trees
```

Accuracy and Runtime Results:

Model	Best Num Trees	Cross-Validation Accuracy	Train Accuracy	Test Accuracy	Training Time (s)
Random Forest	50	0.793127	0.999129	0.79878	0.227646

Table 2.3: Accuracy and Runtime for Random Forest

2.4 K-Nearest Neighbors

We have implemented the k-NN classifier from scratch, determined the k value using 5-fold cross-validation. In order to do this, we gave 5 options for k value in a list and traversed this list to find the one with the best result. After getting this k value, we have used it in the model.

Here is the code snippet from our knn function where we have done the 5-fold cross validation for the k value , which is also available in the code file we have submitted:

```
def set_k(self, X, y, k_values, cv=5):
    """Determine the best k value using cross-validation."""
    best_k = 0
    best_score = 0
    k_fold = KFold(n_splits=cv, shuffle=True, random_state=42)

    for k in k_values:
        self.k = k
        scores = []

        for train_idx, val_idx in k_fold.split(X):
            self.fit(X[train_idx], y[train_idx])
            predictions = self.predict(X[val_idx])
            score = accuracy_score(y[val_idx], predictions)
            scores.append(score)

        mean_score = np.mean(scores)
        if mean_score > best_score:
            best_score = mean_score
            best_k = k

    self.k = best_k
    return best_k, best_score
```

Accuracy and Runtime Results:

Model	Best k	Cross-Validation Accuracy	Train Accuracy	Test Accuracy	Training Time (s)
K-Nearest Neighbours	3	0.706462	0.840157	0.681911	2.799493

Table 2.4: Accuracy and Runtime for K-Nearest Neighbours

2.5 Model Comparison

Below table summarizes the accuracy and runtime results for all implemented models within Task 2.

Model	Train Accuracy	Test Accuracy	Training Time (s)
From Scratch Logistic Regression	0.663328	0.675813	0.649038
Scikit-Learn's Logistic Regression	0.775697	0.792683	0.157849
SVM from Scratch	0.780923	0.771341	46.349932
Scikit-Learn's SVM	0.790505	0.786585	4752.384431
Random Forest	0.999129	0.79878	0.227646
K-Nearest Neighbours	0.840157	0.681911	2.799493

Table 2.5: Model Performance: Accuracy and Runtime

Comments on the Model Comparison Table:

- When we look at the test accuracy column we saw that Random Forest has the highest value as 0.79878, when we looked at the train accuracy of the same model we saw the value as 0.999129 which is a very high value and indicates that the model is suffering from overfitting. Overfitting is a common problem encountered in tree-based methods. In terms of runtime we can say that it is relatively small.
- The second highest test accuracy belongs to the Scikit-Learn's Logistic Regression, the value is so close to the Random Forest's, on the other hand the train accuracy is a smaller value than in Random Forest's which is a good thing and we don't have the concern of overfitting here. Also it has the smallest runtime over all models, hence we concluded that although it is a simple model it performs nicely in a short amount of time which makes it successful for our binary classification task.
- Scikit Learn's SVM has a very close test accuracy to the Scikit-Learn's Logistic Regression, however it has a relatively larger runtime compared to the rest of the models. When we compared their performances, there is not so much improvement in terms of accuracy and the runtime is too large. Since it is a complex model to implement and the performance gain compared to the runtime is too small, it is not a good choice for this task, we prefer working with simpler models.
- The overall test accuracy is approximately 75 percent, which can be considered as a high value from our perspective since this is the first time we are applying these techniques to such dataset.

2.6 Evaluation Metrics

Metric	Logistic Regression (From Scratch)	Logistic Regression (Scikit-Learn)	SVM From Scratch	Random Forest	KNN
Accuracy	0.675813	0.792683	0.789492	0.798780	0.681911
AUROC	0.622366	0.798795	0.803845	0.848997	0.621616
Avg Precision	0.491284	0.767492	0.785463	0.758173	0.432081
Recall	0.144543	0.474926	0.724224	0.604720	0.427729
F1-Score	0.235012	0.612167	0.727392	0.674342	0.480929

Table 2.6: Comparison of Metrics for Implemented Models

Comments on the Evaluation Metrics[3]:

- Classification Accuracy(ACC)
  This metric represents the ratio of correctly classified samples on the test dataset.
- Area Under the Receiver Operating Characteristic Curve (AUROC)
  This metric represents the ability of the model to distinguish between positive and negative classes across different thresholds.
- Average Precision
  This metric is the average of precision scores for each recall threshold.
- Recall
  This metric represents the portion of all actual positives, how much of them are classified correctly.
- F1-score
  This metric combines precision and recall by calculating their harmonic mean.
- Firstly, we will analyze the evaluation metrics for the Logistic Regression From Scratch, Accuracy is 0.675813, which might seem acceptable, but we have the class imbalance, accuracy isn't a reliable metric on its own.Recall is very low (0.144543), indicating the model struggles to identify the minority class (Good) in our case. F1-Score is also quite low (0.235012) indicating poor performance in balancing precision and recall.
- When we implemented Scikit-Learn's Logistic Regression, we can clearly see that the values of evaluation metrics increased. Which says that it performs better for our binary classification task, Scikit-Learn's Logistic Regression improved the performance which we couldn't done in the from scratch implementation.

2.7 Challenges

The most challenging part of this task was Support Vector Machine part, both the implementation and running the model required so much time. The most time-consuming part was the 5-fold cross validation for finding the best paramaters. Thus, completing the SVM part was difficult. That was a bottleneck for the task, since the remaining part requires a comparison between all classifiers we have trained.

3. Task 3: Plotting Decision Boundary

For this task, we have selected two dimensions (features) as 'Wins' and 'BoxOffice' as they are the most significant features determining the categories. We have two categories: 'Bad' and 'Good', since we are performing a binary classification task.

We trained a logistic regression and linear soft margin SVM, imported from Scikit-Learn, and directly used them for training.

After training, we plotted the data points and decision boundaries for the two models in the same scatter plot. In the plot, the blue region represents samples predicted to be in the 'Bad' category, and the red region represents samples predicted to be in the 'Good' category with the logistic regression classifier.

The green line represents the decision boundary of the Linear Soft-Margin SVM.

Below is the resulting plot:

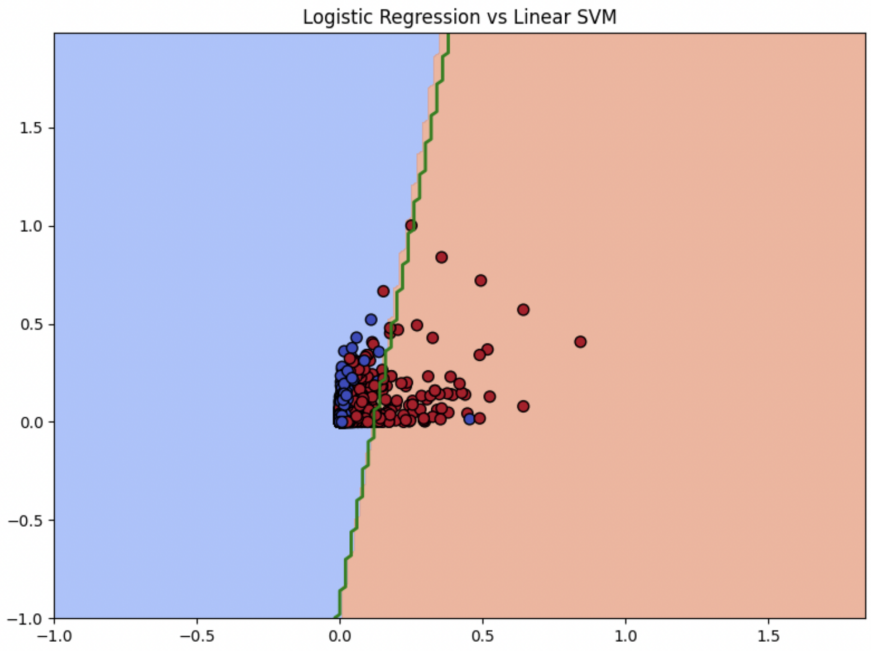

Figure 3.1: Decision Boundaries for Logistic Regression vs SVM

As can be seen from the plot, the decision boundaries for logistic regression and linear soft-margin SVM are so close to each other, we will look at the performances of both models to make inferences.

We compared the accuracy and performance of logistic regression and linear soft margin SVM in the table below:

Model	**Train Accuracy**	**Test Accuracy**	**Training Time (s)**
Logistic Regression	0.7722	0.7724	0.0074
Linear Soft-Margin SVM	0.7713	0.7693	0.3901

Table 3.1: Accuracy and Runtime for Logistic Regression and SVM

Comments on the Table:

- When we looked at the values in the table, we saw that the values are also so close to each other which is consistent with the scatter plot visualization.
- The train and test accuracy values are so close to each other in both models and approximately 77 percent, it seems like no overfitting occurs for both of them.
- When we compare the runtime values, linear soft-margin SVM takes more time, but it is less than a second so there is not a worth-mentioning difference between the runtimes of 2 models.

4. Individual Contributions

- **Aslı Gök:** Implemented SPARQL query to fetch data, Logistic Regression models, Random Forest, documented some sections in the report.
- **Nazlıcan Aka:** Implemented fetch_data.py for OMDB API Call, SVM models, KNN, documented some sections in the report.
- **Dağlar Eren Tekşen:** Implemented K-Means Clustering, plotting decision boundaries, documented some sections in the report.

5. CODE

The code files submitted for the project can be found under this link

6. REFERENCES

[1]: Wikidata Query Service
[2]: Open Movie Database
[3]: Evaluation Metrics Lecture Slides