Comparing Logistic regression and SVM models for determining the tweet’s time zone

Team 3 – Gianluca Guzzetta, MIAO Linxiao, Jan Stoklasa, Alexander Pavlyuk, Ece Elif Unal

Introduction

In this report we will describe development and results of our Human Cantered Machine Learning (HCML) Course project at Aalto university 2022.

Twitter has been one of the most popular main communication tools that people use to express their feelings. Its speed and ease of use make it first-hand choice for analysis of human trends. Using the content and the emotional orientation of the text, one can match the tweet to a certain trend, which tend to belong to a particular location at that time.

The user location can be hidden easily by VPN and the task Is that we can predict the real user location (in our work his/her time zone) using the text analysis of tweets.

We focus on prediction of user time zone based on tweet’s text analysis. Our aim is to draw a connection between the frequency of certain words and regional trends of that time, which yields the original time zone of the tweet. Motivation for this task can be understanding of similar ML methods, which can help us changing our tweets to hide our real location.

The prediction of the time zones will be made with the models described in the section “Methods”. The dataset and the features that we used will be explained in the “Problem Formulation” section. The model with greater performance along with its scores will be described in “Results” section. The results of this project will be criticized in the “Conclusion” section. Finally, the code is provided in the “Appendix” section.

Problem formulation

The Data Source we used is “Twitter Data" by Shyam R. from Kaggle [1].

Data points are represented by tweets. Each datapoint has 26 features. There are 12,252 datapoints after removing null values (7,798 datapoints are removed). We focus only on two of them: tweet’s text and user’s time zone.

Features are tweets’ texts. These features have a string data type initially, but it is converted to the numeric data type during the data pre-processing.

Labels users’ time zones. Each label is string which describes the time zone initially. During the data pre-processing labels are grouped and encoded to the integers.

Methods

Tweets' texts are chosen as features due to the differences in writing style, local trends, and slang in different parts of the world. So, the correlation between tweet's texts and user's time zones exists.

sklearn.feature_extraction.text.CountVectorizer method is used to count the words in the whole dataset and chose only the 2,000 most frequent ones. This step is required due to the lack of computational resources and irrelevance of using infrequent words in the training process. Next, TF-IDF (Term Frequency — Inverse Document Frequency) method is used for forming new features. Now, there are 2,000 features for each datapoint represented by the TF-IDF of each of the 2,000 chosen words. The features are of the float data type.

The labels are grouped into UTC+X classes manually. Then, only the labels with more than 100 datapoints are chosen for the modelling. Finally, the labels are encoded to the integer format.

The training and the validation sets are formed randomly with 80/20 proportion respectively. This proportion is a rule of thumb for such ML problems. Cross-validation method showed almost the same training and validation errors in each fold. Therefore, the simple and fast random division was chosen.

Logistic regression, SVC with RBF kernel and SVC with polynomial kernel of degrees 2, 3, and 4 are used as models for this project.

The reason to use logistic regression is its simplicity and fast training speed. The linear model was chosen to get the highest possible performance without using much computational resources. One-vs-Rest principle is applied.

SVC models are used as a more powerful models, which can show better results. Nevertheless, these models require more computational resources. 4 different kernels are used to find the one with the highest performance.

During our training, we utilized several hyperparameters for different models to achieve the most diverse yet accurate results. For the SVC with polynomial kernel function, we applied polynomials of 2nd, 3rd, and 4th degree, 0 for the independent term in the kernel function, and the default value of gamma which is $\frac{1}{N \cdot \sigma^2(X)}$ for our model, where σ^2 is the input variance and N represents the number of features. The SVC with RBF kernel is applied with the default value for the regularization parameter which is 1.0 and for gamma the same value as in the polynomial kernel. For the logistic regression, we use l2 loss penalty, 1 for regularization parameter and we have the training algorithm use the one-vs-rest (OvR) scheme.

Logistic loss is used for the logistic regression model training. It is a default loss function, which works well with the logistic regression.

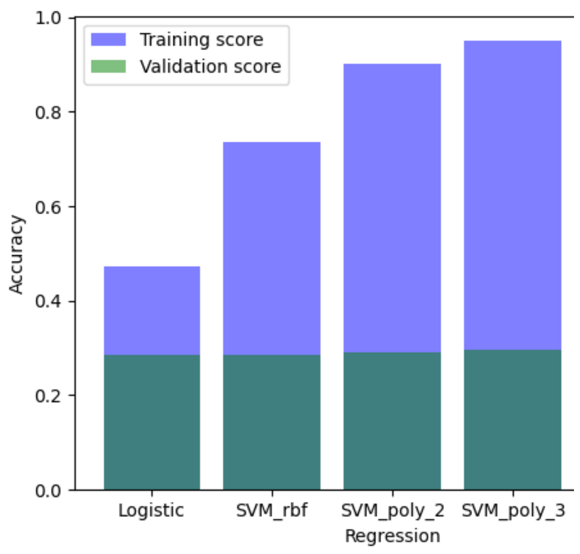
Hinge loss is used for the support vector machine models. This loss finds the largest margin between classes to prevent misclassification. It is a powerful function which is used with SVMs by default.

"Accuracy" is used as a performance measurement. It is simple and understandable. Moreover, it accomplishes the main goal of any classification problem – find out how many datapoints are distributed correctly into classes.

Results

Only SVCs with polynomial kernels show the good training error. Nevertheless, each model shows the same poor performance on the validation set. This means, that there is weak correlation between tweet's text and user's time zone, or the dataset is small. As a rule of thumb, the dataset must be much larger than number of features multiplied by 10. Therefore, more data is required to build a powerful model.

Nevertheless, the SVC model with polynomial kernel of degree 4 shows the best performance. The test set is constructed from the original dataset by randomly taking 10% of datapoints. The test score for the SVC model with polynomial kernel of degree 4 is 26.99%.



Conclusion

Each model shows poor result and cannot be used for user's time zone prediction directly. The reason behind these results is a lack of data. There are numerous unique words, and many of them are crucial in analysis. Therefore, more features and more datapoints along with more computational resources are required in order to achieve meaningful results. There are no doubts that correlation between tweet's text and user's time zone exists and can be found with more data and more computational resources.

So, the following improvements can be made:

1. Collect more data
2. Take more features
3. Use more computational resources.

References:

[1] Dataset:

https://www.kaggle.com/datasets/darkknight98/twitter-data?resource=download

[2] ML textbook:
https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf

[3] ML Python library:

https://scikit-learn.org/stable/index.html

Code:

GitHub link:

https://github.com/AlexPavlyukCode/CS-E407507_ML_project

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import re
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


# Function removes all the punctuation, special characters and capital
letters
def text_cleaning(row):
    text = re.sub('[^a-zA-Z]', ' ', row).lower()
    clean_text = " ".join(text.split())

    return clean_text


# Reading the data file
df = pd.read_csv('tweet_data.csv', encoding='latin-1')
print(f"There are {len(df)} datapoints before filtering.")

# Filtering the dataframe:
#   1. Remove all columns but 2
#   2. Drop all the rows with Nan values
#   3. Renaming columns
df = df.filter(["user_timezone", "text"])
df.dropna(axis=0, inplace=True)
df.reset_index(drop=True, inplace=True)
df.rename(columns={"user_timezone": "timezone"}, inplace=True)
print(f"There are {len(df)} clean datapoints before filtering.")

# Conversion into UTC time zones
df.loc[df["timezone"] == "Cape Verde Is.", "timezone"] = "UTC-1"
df.loc[df["timezone"] == "Newfoundland", "timezone"] = "UTC-3"
df.loc[df["timezone"] == "Atlantic Time (Canada)", "timezone"] = "UTC-3"
df.loc[df["timezone"] == "Brasilia", "timezone"] = "UTC-3"
df.loc[df["timezone"] == "Buenos Aires", "timezone"] = "UTC-3"
df.loc[df["timezone"] == "America/Argentina/Buenos_Aires", "timezone"] =
"UTC-3"
df.loc[df["timezone"] == "Georgetown", "timezone"] = "UTC-4"
df.loc[df["timezone"] == "Caracas", "timezone"] = "UTC-4"
df.loc[df["timezone"] == "Indiana (East)", "timezone"] = "UTC-4"
df.loc[df["timezone"] == "EDT", "timezone"] = "UTC-4"
df.loc[df["timezone"] == "Santiago", "timezone"] = "UTC-4"
df.loc[df["timezone"] == "Mid-Atlantic", "timezone"] = "UTC-4"
df.loc[df["timezone"] == "America/Toronto", "timezone"] = "UTC-4"
df.loc[df["timezone"] == "America/New_York", "timezone"] = "UTC-4"
df.loc[df["timezone"] == "America/Detroit", "timezone"] = "UTC-4"
df.loc[df["timezone"] == "La Paz", "timezone"] = "UTC-4"
df.loc[df["timezone"] == "Eastern Time (US & Canada)", "timezone"] = "UTC-
4"
df.loc[df["timezone"] == "Guadalajara", "timezone"] = "UTC-5"
df.loc[df["timezone"] == "America/Chicago", "timezone"] = "UTC-5"
df.loc[df["timezone"] == "CDT", "timezone"] = "UTC-5"
df.loc[df["timezone"] == "Mexico City", "timezone"] = "UTC-5"
df.loc[df["timezone"] == "Lima", "timezone"] = "UTC-5"
df.loc[df["timezone"] == "Quito", "timezone"] = "UTC-5"
df.loc[df["timezone"] == "Central Time (US & Canada)", "timezone"] = "UTC-
5"
df.loc[df["timezone"] == "Bogota", "timezone"] = "UTC-5"
df.loc[df["timezone"] == "Monterrey", "timezone"] = "UTC-5"
df.loc[df["timezone"] == "America/Boise", "timezone"] = "UTC-6"
df.loc[df["timezone"] == "Arizona", "timezone"] = "UTC-6"
df.loc[df["timezone"] == "CST", "timezone"] = "UTC-6"
df.loc[df["timezone"] == "Chihuahua", "timezone"] = "UTC-6"
df.loc[df["timezone"] == "Mountain Time (US & Canada)", "timezone"] = "UTC-
6"
df.loc[df["timezone"] == "Central America", "timezone"] = "UTC-6"
df.loc[df["timezone"] == "Saskatchewan", "timezone"] = "UTC-6"
df.loc[df["timezone"] == "Mazatlan", "timezone"] = "UTC-6"
df.loc[df["timezone"] == "America/Vancouver", "timezone"] = "UTC-7"
df.loc[df["timezone"] == "Pacific Time (US & Canada)", "timezone"] = "UTC-
7"
df.loc[df["timezone"] == "America/Los_Angeles", "timezone"] = "UTC-7"
df.loc[df["timezone"] == "PDT", "timezone"] = "UTC-7"
df.loc[df["timezone"] == "Tijuana", "timezone"] = "UTC-7"
df.loc[df["timezone"] == "PST", "timezone"] = "UTC-8"
df.loc[df["timezone"] == "Alaska", "timezone"] = "UTC-8"
df.loc[df["timezone"] == "Hawaii", "timezone"] = "UTC-10"
df.loc[df["timezone"] == "Midway Island", "timezone"] = "UTC-11"
df.loc[df["timezone"] == "International Date Line West", "timezone"] =
"UTC-15"
df.loc[df["timezone"] == "Azores", "timezone"] = "UTC"
df.loc[df["timezone"] == "Greenland", "timezone"] = "UTC"
df.loc[df["timezone"] == "Monrovia", "timezone"] = "UTC"
df.loc[df["timezone"] == "Africa/Lagos", "timezone"] = "UTC+1"
df.loc[df["timezone"] == "BST", "timezone"] = "UTC+1"
df.loc[df["timezone"] == "West Central Africa", "timezone"] = "UTC+1"
df.loc[df["timezone"] == "Lisbon", "timezone"] = "UTC+1"
df.loc[df["timezone"] == "Casablanca", "timezone"] = "UTC+1"
df.loc[df["timezone"] == "Edinburgh", "timezone"] = "UTC+1"
df.loc[df["timezone"] == "London", "timezone"] = "UTC+1"
df.loc[df["timezone"] == "Europe/London", "timezone"] = "UTC+1"
df.loc[df["timezone"] == "Dublin", "timezone"] = "UTC+1"
df.loc[df["timezone"] == "Europe/Sarajevo", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Africa/Cairo", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Ljubljana", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Harare", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Bern", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Zagreb", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Cairo", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Budapest", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Stockholm", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Bratislava", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Skopje", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Copenhagen", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Prague", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Madrid", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Amsterdam", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Rome", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Pretoria", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Paris", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Europe/Paris", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Belgrade", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Warsaw", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Berlin", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Paris", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Brussels", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Vienna", "timezone"] = "UTC+2"
df.loc[df["timezone"] == "Europe/Moscow"] = "UTC+3"
df.loc[df["timezone"] == "Baghdad"] = "UTC+3"
df.loc[df["timezone"] == "Moscow"] = "UTC+3"
df.loc[df["timezone"] == "Athens"] = "UTC+3"
df.loc[df["timezone"] == "Europe/Athens"] = "UTC+3"
df.loc[df["timezone"] == "Jerusalem"] = "UTC+3"
df.loc[df["timezone"] == "Bucharest"] = "UTC+3"
df.loc[df["timezone"] == "Istanbul"] = "UTC+3"
df.loc[df["timezone"] == "Helsinki"] = "UTC+3"
df.loc[df["timezone"] == "Nairobi"] = "UTC+3"
df.loc[df["timezone"] == "Africa/Nairobi"] = "UTC+3"
df.loc[df["timezone"] == "Volgograd"] = "UTC+3"
df.loc[df["timezone"] == "Kyiv"] = "UTC+3"
df.loc[df["timezone"] == "Vilnius"] = "UTC+3"
df.loc[df["timezone"] == "GMT+3"] = "UTC+3"
df.loc[df["timezone"] == "Minsk"] = "UTC+3"
df.loc[df["timezone"] == "Tallinn"] = "UTC+3"
df.loc[df["timezone"] == "Riyadh"] = "UTC+3"
df.loc[df["timezone"] == "Riga"] = "UTC+3"
df.loc[df["timezone"] == "Sofia"] = "UTC+3"
df.loc[df["timezone"] == "Kuwait"] = "UTC+3"
df.loc[df["timezone"] == "Baku"] = "UTC+4"
df.loc[df["timezone"] == "Abu Dhabi"] = "UTC+4"
df.loc[df["timezone"] == "Yerevan"] = "UTC+4"
df.loc[df["timezone"] == "Muscat"] = "UTC+4"
df.loc[df["timezone"] == "Kabul"] = "UTC+4:30"
df.loc[df["timezone"] == "Tehran"] = "UTC+4:30"
df.loc[df["timezone"] == "Ekaterinburg"] = "UTC+5"
df.loc[df["timezone"] == "Tashkent"] = "UTC+5"
df.loc[df["timezone"] == "Islamabad"] = "UTC+5"
df.loc[df["timezone"] == "Asia/Karachi"] = "UTC+5"
df.loc[df["timezone"] == "Karachi"] = "UTC+5"
df.loc[df["timezone"] == "IST"] = "UTC+5:30"
df.loc[df["timezone"] == "Mumbai"] = "UTC+5:30"
df.loc[df["timezone"] == "New Delhi"] = "UTC+5:30"
df.loc[df["timezone"] == "Chennai"] = "UTC+5:30"
df.loc[df["timezone"] == "Sri Jayawardenepura"] = "UTC+5:30"
df.loc[df["timezone"] == "Kolkata"] = "UTC+5:30"
df.loc[df["timezone"] == "Dhaka"] = "UTC+6"
df.loc[df["timezone"] == "Almaty"] = "UTC+6"
df.loc[df["timezone"] == "Novosibirsk"] = "UTC+7"
df.loc[df["timezone"] == "Hanoi"] = "UTC+7"
df.loc[df["timezone"] == "Jakarta"] = "UTC+7"
df.loc[df["timezone"] == "Bangkok"] = "UTC+7"
df.loc[df["timezone"] == "Krasnoyarsk"] = "UTC+7"
df.loc[df["timezone"] == "Ulaan Bataar"] = "UTC+8"
df.loc[df["timezone"] == "Urumqi"] = "UTC+8"
df.loc[df["timezone"] == "Kuala Lumpur"] = "UTC+8"
df.loc[df["timezone"] == "Perth"] = "UTC+8"
df.loc[df["timezone"] == "Irkutsk"] = "UTC+8"
df.loc[df["timezone"] == "Singapore"] = "UTC+8"
df.loc[df["timezone"] == "Hong Kong"] = "UTC+8"
df.loc[df["timezone"] == "Taipei"] = "UTC+8"
df.loc[df["timezone"] == "Beijing"] = "UTC+8"
df.loc[df["timezone"] == "Osaka"] = "UTC+9"
df.loc[df["timezone"] == "Tokyo"] = "UTC+9"
df.loc[df["timezone"] == "Seoul"] = "UTC+9"
df.loc[df["timezone"] == "Yakutsk"] = "UTC+9"
df.loc[df["timezone"] == "Darwin"] = "UTC+9:30"
df.loc[df["timezone"] == "Adelaide"] = "UTC+9:30"
df.loc[df["timezone"] == "Australia/Brisbane"] = "UTC+10"
df.loc[df["timezone"] == "Canberra"] = "UTC+10"
df.loc[df["timezone"] == "Sydney"] = "UTC+10"
df.loc[df["timezone"] == "Hobart"] = "UTC+10"
df.loc[df["timezone"] == "Melbourne"] = "UTC+10"
df.loc[df["timezone"] == "Brisbane"] = "UTC+10"
df.loc[df["timezone"] == "Guam"] = "UTC+10"
df.loc[df["timezone"] == "Magadan"] = "UTC+11"
df.loc[df["timezone"] == "Solomon Is."] = "UTC+11"
df.loc[df["timezone"] == "New Caledonia"] = "UTC+11"
df.loc[df["timezone"] == "Fiji"] = "UTC+12"
df.loc[df["timezone"] == "Wellington"] = "UTC+12"
df.loc[df["timezone"] == "Auckland"] = "UTC+12"
df.loc[df["timezone"] == "Samoa"] = "UTC+13"
df.loc[df["timezone"] == "Nuku'alofa"] = "UTC+13"

# Counting datapoints for each time zone
# temp = df.groupby(["timezone"]).count()
# temp = temp.sort_values(by=["text"], ascending=True)
# for i in range(len(np.unique(df["timezone"]))):
#     print(i, temp.iloc[i])
# print()

# Keep all labels with more than 100 datapoints
timezones_to_keep = ["UTC-7", "UTC-4", "UTC-5", "UTC+1", "UTC+2", "UTC-3",
"UTC-6",
                     "UTC+3", "UTC-10", "UTC+8", "UTC-8", "UTC+10"]
df.drop(df[~df["timezone"].isin(timezones_to_keep)].index, inplace=True)
print(f"There are {len(df)} datapoints after filtering.")

# Encoding timezones with integer values starting from 0
le = LabelEncoder()
df["timezone"] = le.fit_transform(df["timezone"])

# Cleaning the text from uppercase letters, special characters and
punctuation
df["clean_text"] = df["text"].apply(lambda f: text_cleaning(f))
df.drop(["text"], axis=1, inplace=True)
```

```
# Take n features:
#   1. Creating CountVectorizer()
#   2. Counting the number of each word in the whole dataset
#   3. Get the vocabulary and put into the "names" variable
#   4. Creating the dataframe from vocabulary and the number of words in
the dataset
#   5. Sorting the dataframe by the number of words in the dataset
#   6. Choosing n_features most frequent words and adding the rest of them
to the stop_words list
count_vectorizer = CountVectorizer()
counter = count_vectorizer.fit_transform(df["clean_text"]).toarray()
word_frequency = counter.sum(axis=0)
names = count_vectorizer.get_feature_names_out()
data = pd.DataFrame({"name": names,
                     "frequency": word_frequency})
data = data.sort_values(by=["frequency"])
n_features = 2000
stop_words = data.head(len(names) - n_features)["name"].to_numpy()
print(f"There are {len(names)} features (unique words) in total, but we get
only {n_features} out of them")

# Convert the text features to the TF-IDF features
# X_train is a rxc matrix, where r is the number of data points and c is
the number of unique words
tf_idf_vectorizer = TfidfVectorizer(stop_words=list(stop_words))
X = tf_idf_vectorizer.fit_transform(df["clean_text"]).toarray()
print(f"The size of each feature matrix is {len(X)} x {len(X[0])}")

# Get the numpy array of labels
y = df["timezone"].to_numpy()
print(f"There are {len(np.unique(y))} classes (timezones)\n")

# Random training/validation split
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y,
test_size=0.1, random_state=3)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,
test_size=0.2, random_state=3)
print(f"The size of test set is {len(X_test)}")
print(f"The size of training set is {len(X_train)}")
print(f"The size of validation set is {len(X_val)}")
print()

# Logistic Regression
log_reg = LogisticRegression(multi_class='ovr')
log_reg.fit(X_train, y_train)

err_train_log_reg = log_reg.score(X_train, y_train)
err_val_log_reg = log_reg.score(X_val, y_val)
print("Logistic regression:")
print(f"Training error: {round(100 * err_train_log_reg, 2)}%")
print(f"Validation error: {round(100 * err_val_log_reg, 2)}%")
print()

# Support Vector Classification with radial-based function as kernel
svm_rbf = SVC(kernel="rbf")
svm_rbf.fit(X_train, y_train)

err_train_svm_rbf = svm_rbf.score(X_train, y_train)
err_val_svm_rbf = svm_rbf.score(X_val, y_val)
print("SVC with rbf kernel:")
print(f"Training error: {round(100 * err_train_svm_rbf, 2)}%")
print(f"Validation error: {round(100 * err_val_svm_rbf, 2)}%")
print()

# Support Vector Classification with polynomial functions as kernel
svm_poly_2 = SVC(kernel="poly", degree=2)
svm_poly_2.fit(X_train, y_train)

err_train_svm_poly_2 = svm_poly_2.score(X_train, y_train)
err_val_svm_poly_2 = svm_poly_2.score(X_val, y_val)
print("SVC with 2nd degree polynomial kernel:")
print(f"Training error: {round(100 * err_train_svm_poly_2, 2)}%")
print(f"Validation error: {round(100 * err_val_svm_poly_2, 2)}%")
print()

svm_poly_3 = SVC(kernel="poly", degree=3)
svm_poly_3.fit(X_train, y_train)

err_train_svm_poly_3 = svm_poly_3.score(X_train, y_train)
err_val_svm_poly_3 = svm_poly_3.score(X_val, y_val)
print("SVC with 3rd degree polynomial kernel:")
print(f"Training error: {round(100 * err_train_svm_poly_3, 2)}%")
print(f"Validation error: {round(100 * err_val_svm_poly_3, 2)}%")
print()

svm_poly_4 = SVC(kernel="poly", degree=4)
svm_poly_4.fit(X_train, y_train)

err_train_svm_poly_4 = svm_poly_4.score(X_train, y_train)
err_val_svm_poly_4 = svm_poly_4.score(X_val, y_val)
err_test_svm_poly_4 = svm_poly_4.score(X_test, y_test)
print("SVC with 4th degree polynomial kernel:")
print(f"Training error: {round(100 * err_train_svm_poly_4, 2)}%")
print(f"Validation error: {round(100 * err_val_svm_poly_4, 2)}%")
print(f"Test error: {round(100 * err_test_svm_poly_4, 2)}%")
print()

# Plotting training and validation scores of each model
training_errors = [err_train_log_reg, err_train_svm_rbf,
err_train_svm_poly_2, err_train_svm_poly_3,
                   err_train_svm_poly_4]
validation_errors = [err_val_log_reg, err_val_svm_rbf, err_val_svm_poly_2,
err_val_svm_poly_3,
                     err_val_svm_poly_4]
regressions = ["Logistic", "SVM_rbf", "SVM_poly_2", "SVM_poly_3",
"SVM_poly_4"]

plt.bar(regressions, training_errors, alpha=0.5, color="b", label="Training
score")
plt.bar(regressions, validation_errors, alpha=0.5, color="g",
label="Validation score")
plt.xlabel("Regression")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

# Tests obtained in experiments

# Test #1
# The size of feature matrix is 12252 x 200
# Training errors for 5 CV folds are: (array([0.25159158, 0.25134672,
0.25159158]), 2)
```

```
# Average training error for 5 CV folds is: 25.15%
# Validation errors for 5 CV folds are: [0.19147894 0.20666014 0.19539667]
# Average validation error for 5 CV folds is: 19.78%
# Training errors for 5 CV folds are: [0.39700031 0.39812264 0.39624566
0.39726586 0.39828606]
# Average training error for 5 CV folds is: 39.74%
# Validation errors for 5 CV folds are: [0.21419829 0.2129743  0.22612245
0.22122449 0.21632653]
# Average validation error for 5 CV folds is: 21.82%

# Test #2
# The size of feature matrix is 12252 x 500
# Logistic regression:
# Training error: 27.98%
# Validation error: 21.66%
#
# SVC:
# Training error: 46.82%
# Validation error: 22.97%

# Test #3
# The size of feature matrix is 10762 x 2000
# Logistic regression:
# Training error: 41.81%
# Validation error: 26.24%
#
# SVC:
# Training error: 63.72%
# Validation error: 28.1%

# Test #4
# There are 20441 features in total, but we get only 2000 out of them
# The size of feature matrix is 6841 x 2000
# There are 3 classes (timezones)
#
# Logistic regression:
# Training error: 63.98%
# Validation error: 39.37%
#
# SVC with rbf kernel:
# Training error: 91.3%
# Validation error: 40.83%
#
# SVC with 3rd degree polynomial kernel:
# Training error: 98.01%
# Validation error: 42.0%

# Test #5
# There are 29156 features in total, but we get only 2000 out of them
# The size of feature matrix is 11775 x 2000
# There are 12 classes (timezones)
#
# Logistic regression:
# Training error: 47.09%
# Validation error: 27.35%
#
# SVC with rbf kernel:
# Training error: 73.16%
# Validation error: 27.86%
#
# SVC with 3rd degree polynomial kernel:
```

```
# Training error: 94.63%
# Validation error: 28.2%

# Test #6
# There are 20050 datapoints before filtering.
# There are 12252 clean datapoints before filtering.
# There are 11775 datapoints after filtering.
# There are 29156 features (unique words) in total, but we get only 2000
out of them
# The size of each feature matrix is 11775 x 2000
# There are 12 classes (timezones)
#
# The size of test set is 1178
# The size of training set is 8477
# The size of validation set is 2120
#
# Logistic regression:
# Training error: 47.19%
# Validation error: 28.44%
#
# SVC with rbf kernel:
# Training error: 73.48%
# Validation error: 28.58%
#
# SVC with 2nd degree polynomial kernel:
# Training error: 90.09%
# Validation error: 29.2%
#
# SVC with 3rd degree polynomial kernel:
# Training error: 94.88%
# Validation error: 29.58%
#
#
# SVC with 4th degree polynomial kernel:
# Training error: 95.49%
# Validation error: 29.25%
# Test error: 26.99%
```