ML For LLMS

July 19, 2024

[11]:
```
import pandas as pd
import numpy as np
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from tqdm import tqdm
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
nltk.download('stopwords', quiet= True)
import warnings
warnings.filterwarnings("ignore")
```

Below we create a dataset by merging "final" and "total". This new dataset contains the following columns:

-question (the questions that the LLMs were asked)
-context (the context that was given to the LLMs)
-actual_answer (the correct answer to the question)
-predicted_answer_roberta (the answer provided by the roberta LLM)
-predicted_answer_t5 (the answer provided by the t5 LLM)
-predicted_answer_distilbert (the answer provided by the distilbert LLM)
-best_model (the LLM that provided the best answer to a question)

Some of the answers contain whitespaces, so we will remove those. Then, we creat a new column, called 'answered_correctly', whose value is 1 if at least one of the LLMs provided a correct answer to a goven question and 0 otherwise.

[12]:
```
df = pd.read_csv('/Users/sandinatatu/Desktop/final.csv', usecols =['question',␣
 ↪'context','actual_answer'])
```

```
df2 = pd.read_csv('/Users/sandinatatu/Desktop/total.csv',␣
 ↪usecols=['question','predicted_answer_roberta',␣
 ↪'predicted_answer_distilbert', 'predicted_answer_t5', 'best_model'])
df = df.merge(df2, on = 'question', how='inner').reset_index()

df['predicted_answer_t5'] = df['predicted_answer_t5'].str.strip()
df['predicted_answer_distilbert'] = df['predicted_answer_distilbert'].str.
 ↪strip()
df['predicted_answer_roberta'] = df['predicted_answer_roberta'].str.strip()
df['actual_answer'] = df['actual_answer'].str.strip()


matches = (df['predicted_answer_roberta'] == df['actual_answer']) | \
          (df['predicted_answer_distilbert'] == df['actual_answer']) | \
          (df['predicted_answer_t5'] == df['actual_answer'])

df['answered_correctly'] = matches.astype(int)

df.head()
```

[12]:

	index	question	context	actual_answer	predicted_answer_roberta	predicted_answer_distilbert	predicted_answer_t5	best_model	answered_correctly
0	0	Which magazine was started first Arthur's Maga…	Radio City (Indian radio station): Radio City …	arthurs magazine	first for women	first for women	arthurs magazine	t5	1
1	1	The Oberoi family is part of a hotel company t…	Ritz-Carlton Jakarta: The Ritz-Carlton Jakarta…	delhi	delhi	delhi	delhi	t5	1
2	2	Musician and satirist Allie Goertz wrote a son…	Lisa Simpson: Lisa Marie Simpson is a fictiona…	president richard nixon	president richard nixon	allison beth	members of his own family	roberta	1
3	3	What nationality was James Henry Miller's wife?	Moloch: or, This Gentile World: Moloch: or, Th…	american	australian	british	english	t5	0
4	4	Cadmium Chloride is slightly soluble in this c…	Cadmium chloride: Cadmium chloride is a white …	alcohol	methyl alcohol	methyl alcohol	alcohol	t5	1

Since we will be using the TF-IDF matrix of the "question" column to obtain the features for our ML model, we first need to clean the data. To do this, we create the following function which:

-lowers all of the characters in a given string
-replaces "","/", and"-" characters with spaces (since these characters might appear between words)
-removes non-alphanumeric characters
-removes whitespaces
-tokenizes all of the words in the string
-removes all of the stopwords from the tokenized string (words such as "in", "the", or "on")
-conducts stemming on the words (e.g.: "running" necomes "run", "easily" becomes "easili")

[13]:
```
def preprocess_context(context):

    #lowercase everything
    context = context.lower()

    #remove punctuation
    context = re.sub(r"[\\/-]", ' ', context)
    context = re.sub(r'[^a-zA-Z0-9\s]', '', context)

    #remove whitespace
    context = context.strip()

    #tokenize the words (transform the context into an array of words)
    tokens = word_tokenize(context)

    #remove stopwords from the tokens
    swords = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in swords]

    #conduct stemming (extracting the base form from words)
    ps = PorterStemmer()
    tokens = [ps.stem(token) for token in tokens]

    return tokens
```

We then apply this function to the "question" column in the dataframe and the join all of the "cleaned" words together.

[14]:
```
#preprocess all contexts
df['question'] = df['question'].apply(preprocess_context)
df['question'] = df['question'].apply(' '.join)
```

Then, we split our data into training and testing.

[15]:
```
X = df[['question']].copy()
y=df[['answered_correctly']].copy()
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 42)

X_train = pd.DataFrame(X_train, columns = X.columns)
X_test = pd.DataFrame(X_test, columns = X.columns)
```

0.0.1 Random Forest

Below we fit a random forest model to our training data using cross-validation. To do this, we:

-initialize a pipeline that containd a TFidfVectorizer and a RandomForestClassifier
-initialize a param_grid, controlling for 'classifier___n_estimators' (how many trees we will have in the forest), 'classifier___max_depth:' (max_depth of each tree in a forest), 'classifier___min_samples_leaf' (the min number of samples required to be at each leaf node), 'classifier___min_samples_leaf' (the max number of leaf nodes in a tree). -fit this param_grid to the training data using cross-validation

[17]:
```
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=300)),
    ('classifier', RandomForestClassifier())
])

param_grid = {
    'classifier__n_estimators': [1000, 1500],
    'classifier__max_depth': [80, 100],
    'classifier__min_samples_leaf': [1, 2],
    'classifier__max_leaf_nodes': [80, 100]
}


grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train['question'], y_train)
```

[17]:
```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('tfidf',
                                        TfidfVectorizer(max_features=300)),
                                       ('classifier',
                                        RandomForestClassifier())]),
             param_grid={'classifier__max_depth': [80, 100],
                         'classifier__max_leaf_nodes': [80, 100],
                         'classifier__min_samples_leaf': [1, 2],
                         'classifier__n_estimators': [1000, 1500]},
             scoring='accuracy')
```

Now that we identified the model which works best, we save it in "best_model". We also save the best parameters of this model in "best_params".

We also save the "mean_test_scores" and "std_test_score" for every single one of our models. We then print these for every single one of our models.

Finally, we use the best_params on the entire training set and then predict on the test set. We also compute accuracy on the test set.

[18]:
```
# Get the best parameters and model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Print accuracy for each fold
cv_results = grid_search.cv_results_
mean_test_scores = cv_results['mean_test_score']
std_test_scores = cv_results['std_test_score']
for mean, std, params in zip(mean_test_scores, std_test_scores,␣
 ↪cv_results['params']):
    print(f"Mean accuracy: {mean:.3f} (± {std:.3f}) for parameters: {params}")
# Print the best parameters to ensure they exist
print("Best Parameters:", best_params)

# Refit the pipeline on the entire training set with the best parameters
best_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=300)),
    ('classifier', RandomForestClassifier(
        n_estimators=best_params.get('classifier__n_estimators', 100),
        max_depth=best_params.get('classifier__max_depth', None),
        min_samples_leaf=best_params.get('classifier__min_samples_leaf', 1),
        max_leaf_nodes=best_params.get('classifier__max_leaf_nodes', None)
    ))
])

best_pipeline.fit(X_train['question'], y_train)

# Make predictions on the test set
y_pred = best_pipeline.predict(X_test['question'])

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nRandom Forest Accuracy on Test Set:", accuracy)
```

```
Mean accuracy: 0.608 (± 0.002) for parameters: {'classifier__max_depth': 80,
'classifier__max_leaf_nodes': 80, 'classifier__min_samples_leaf': 1,
'classifier__n_estimators': 1000}
Mean accuracy: 0.605 (± 0.006) for parameters: {'classifier__max_depth': 80,
'classifier__max_leaf_nodes': 80, 'classifier__min_samples_leaf': 1,
'classifier__n_estimators': 1500}
Mean accuracy: 0.605 (± 0.006) for parameters: {'classifier__max_depth': 80,
'classifier__max_leaf_nodes': 80, 'classifier__min_samples_leaf': 2,
'classifier__n_estimators': 1000}
Mean accuracy: 0.608 (± 0.003) for parameters: {'classifier__max_depth': 80,
'classifier__max_leaf_nodes': 80, 'classifier__min_samples_leaf': 2,
```

```
'classifier__n_estimators': 1500}
Mean accuracy: 0.610 (± 0.002) for parameters: {'classifier__max_depth': 80,
'classifier__max_leaf_nodes': 100, 'classifier__min_samples_leaf': 1,
'classifier__n_estimators': 1000}
Mean accuracy: 0.610 (± 0.002) for parameters: {'classifier__max_depth': 80,
'classifier__max_leaf_nodes': 100, 'classifier__min_samples_leaf': 1,
'classifier__n_estimators': 1500}
Mean accuracy: 0.609 (± 0.002) for parameters: {'classifier__max_depth': 80,
'classifier__max_leaf_nodes': 100, 'classifier__min_samples_leaf': 2,
'classifier__n_estimators': 1000}
Mean accuracy: 0.610 (± 0.002) for parameters: {'classifier__max_depth': 80,
'classifier__max_leaf_nodes': 100, 'classifier__min_samples_leaf': 2,
'classifier__n_estimators': 1500}
Mean accuracy: 0.608 (± 0.003) for parameters: {'classifier__max_depth': 100,
'classifier__max_leaf_nodes': 80, 'classifier__min_samples_leaf': 1,
'classifier__n_estimators': 1000}
Mean accuracy: 0.608 (± 0.003) for parameters: {'classifier__max_depth': 100,
'classifier__max_leaf_nodes': 80, 'classifier__min_samples_leaf': 1,
'classifier__n_estimators': 1500}
Mean accuracy: 0.608 (± 0.002) for parameters: {'classifier__max_depth': 100,
'classifier__max_leaf_nodes': 80, 'classifier__min_samples_leaf': 2,
'classifier__n_estimators': 1000}
Mean accuracy: 0.608 (± 0.002) for parameters: {'classifier__max_depth': 100,
'classifier__max_leaf_nodes': 80, 'classifier__min_samples_leaf': 2,
'classifier__n_estimators': 1500}
Mean accuracy: 0.609 (± 0.002) for parameters: {'classifier__max_depth': 100,
'classifier__max_leaf_nodes': 100, 'classifier__min_samples_leaf': 1,
'classifier__n_estimators': 1000}
Mean accuracy: 0.610 (± 0.002) for parameters: {'classifier__max_depth': 100,
'classifier__max_leaf_nodes': 100, 'classifier__min_samples_leaf': 1,
'classifier__n_estimators': 1500}
Mean accuracy: 0.609 (± 0.002) for parameters: {'classifier__max_depth': 100,
'classifier__max_leaf_nodes': 100, 'classifier__min_samples_leaf': 2,
'classifier__n_estimators': 1000}
Mean accuracy: 0.610 (± 0.002) for parameters: {'classifier__max_depth': 100,
'classifier__max_leaf_nodes': 100, 'classifier__min_samples_leaf': 2,
'classifier__n_estimators': 1500}
Best Parameters: {'classifier__max_depth': 80, 'classifier__max_leaf_nodes':
100, 'classifier__min_samples_leaf': 1, 'classifier__n_estimators': 1500}

Random Forest Accuracy on Test Set: 0.6174827472888597
```

0.0.2 Gaussian Naive Bayes

[19]:
```
from sklearn.naive_bayes import GaussianNB
```

Prior to fitting a Gaussian Naive Bayse model to the data, we need to transform the tf-idf matrix to a dense matrix. We do this through the function defined below.

Then, just as in the case of the RandomForest, we define a pipeline. This pipeline first computes the tf-idf matrix of the data, then transform this matrix to a dense format, and then finally fits a GaussianNB model to the data.

For the param_grid in this case, we define differet possible values for the var_smoothing. We identify the best var_smoothing through cross_validation.

[20]:
```
import numpy as np
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import GaussianNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import GridSearchCV

class DenseTransformer(TransformerMixin, BaseEstimator):
    def fit(self, X, y=None, **fit_params):
        return self

    def transform(self, X, y=None, **fit_params):
        return np.asarray(X.todense())

# Define the pipeline
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=1500)),
    ('to_dense', DenseTransformer()),
    ('classifier', GaussianNB())
])

# Define the hyperparameter grid
param_grid = {
    'classifier__var_smoothing': [1e-10, 1e-9, 1e-8, 1e-7, 1e-6]
}

# Setup the GridSearchCV
grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid, cv=3,␣
 ↪scoring='accuracy')

# Convert y_train to a NumPy array and reshape to a 1D array
y_train_array = y_train.values.ravel()

# Fit the model
grid_search.fit(X_train['question'], y_train_array)

# Get the best parameters
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print(f"Best Parameters: {best_params}")
```

```
print(f"Best Cross-Validation Score: {best_score}")
```

```
Best Parameters: {'classifier__var_smoothing': 1e-06}
Best Cross-Validation Score: 0.5541161763461012
```

[21]:
```
best_params_nb = grid_search.best_params_
best_model_nb = grid_search.best_estimator_

cv_results_nb = grid_search.cv_results_
nb_mean_test_scores = cv_results_nb['mean_test_score']
nb_std_test_scores = cv_results_nb['std_test_score']

for mean, std, params in zip(nb_mean_test_scores, nb_std_test_scores,␣
 ↪cv_results_nb['params']):
    print(f"Mean accuracy: {mean:.3f} (± {std:.3f}) for parameters: {params}")

# Print the best parameters to ensure they exist
print("Best Parameters:", best_params_nb)

# Refit the pipeline on the entire training set with the best parameters
best_pipeline_nb = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=300)),
    ('to_dense', DenseTransformer()),
    ('classifier', GaussianNB(
       var_smoothing=best_params_nb.get('classifier__var_smoothing'),
    ))
])

best_pipeline_nb.fit(X_train['question'], y_train_array)

# Make predictions on the test set
y_pred = best_pipeline_nb.predict(X_test['question'])

# Evaluate the model
accuracy_nb = accuracy_score(y_test.values.ravel(), y_pred)

print(f"Test Accuracy: {accuracy_nb:.3f}")
```

```
Mean accuracy: 0.546 (± 0.005) for parameters: {'classifier__var_smoothing':
1e-10}
Mean accuracy: 0.548 (± 0.005) for parameters: {'classifier__var_smoothing':
1e-09}
Mean accuracy: 0.550 (± 0.004) for parameters: {'classifier__var_smoothing':
1e-08}
Mean accuracy: 0.552 (± 0.004) for parameters: {'classifier__var_smoothing':
1e-07}
Mean accuracy: 0.554 (± 0.003) for parameters: {'classifier__var_smoothing':
1e-06}
```

```
Best Parameters: {'classifier__var_smoothing': 1e-06}
Test Accuracy: 0.575
```