Optimizing Search Engine Relevance

By

Samaneh Karimi

LinkedIn: https://www.linkedin.com/in/samane-karimi-52178459/

Outline

- ❖ Summary
  - Preprocessing
  - Techniques and models used
  - Evaluation results
- ❖ Conclusion and final recommendations
- ❖ Python implementation and visual walk-through of the outputs

Summary of my approach

1. Analyzing raw data
2. Preprocessing
3. Model selection
4. Implementing the models and evaluation

Step1) Analyzing raw data

Step 1) Analyzing raw data

- **Goal: Knowing the characteristics of the data**

- Balanced or imbalanced?

Class	1 (Relevant)	0 (Non-relevant)
#Instance (ratio)	34987 (44%)	45060 (56%)

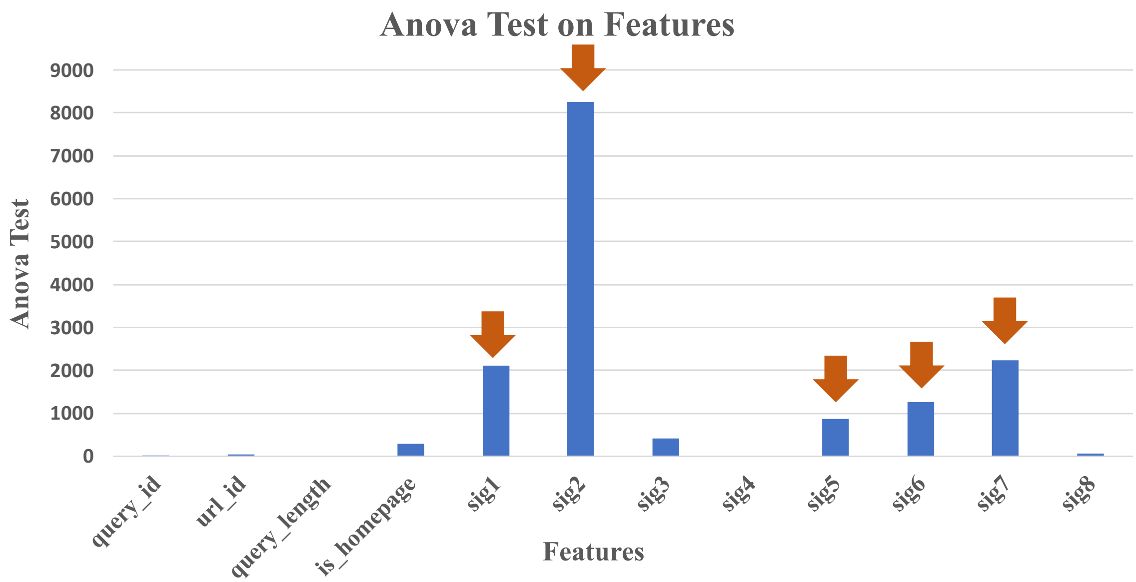
=> balanced (no need for using imbalanced data handling techniques)

- Analyzing feature importance:
  - Using ANOVA Test

Step 1) Analyzing the raw data (Cont.)

- Top 5 features based on ANOVA test:

1. sig2
2. sig7
3. sig1
4. sig6
5. sig5



Step 2) Preprocessing

- Handling missing values
- Duplicate removal
- Feature scaling
- Standardization
- Feature selection (dimensionality reduction)

Step 2) Preprocessing

- **Handling missing values**
  - No missing value was found in the dataset.
- **Duplicate removal**
  - The redundant data may appear both in the training and testing sets and cause inaccurate learning of the model.
  - No duplicate was found in the dataset.
- **Feature scaling**
  - The majority of machine learning algorithms perform much better when dealing with features that are on the same scale.
- **Standardization:**
  - Centering the feature columns at mean 0 with standard deviation 1 so that the feature columns have the same parameters as a standard normal distribution (zero mean and unit variance).

Step 2) Preprocessing (Cont.)

- **Feature Selection (Dimensionality Reduction)**
  - The process of reducing the number of input variables when developing a predictive model to both reduce the computational cost of modeling and to improve the performance of the model.

  - **Two approaches tested in this project:**
  1. Statistics for filter-based feature selection methods:
     - Since we have numerical input and binary annotations in our dataset => ANOVA correlation coefficient is used.
  2. Principal Component Analysis (PCA)
     - finds directions of maximal variance of data

Step 3) Model selection

- Random Forest
- Support Vector Machines (SVM)
- Deep Learning (MLP)

Step 3) Model selection
Random Forest

- The **random forest** approach is a ensemble (bagging) method where **deep trees**, fitted on bootstrap samples, are combined to produce an output with lower variance. For each test data, the output class would be the mode of the classes of the individual trees.

- My reasons for choosing it:
  - It runs efficiently on large databases.(✔)
  - Since it takes the **advantage** of ensemble learning, reduces variance and thus helps us avoid overfitting.

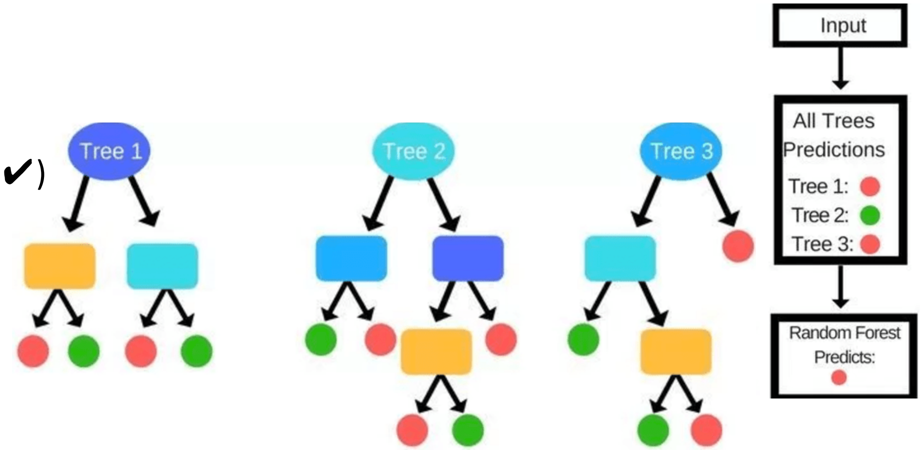
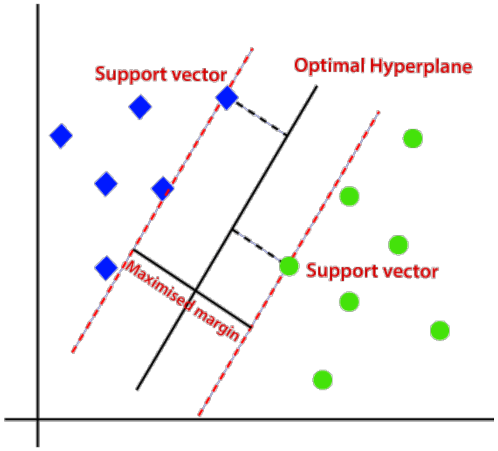


Image Ref: https://syncedreview.com/2017/10/24/how-random-forest-algorithm-works-in-machine-learning/

Step 3) Model selection

Support Vector Machines (SVM)

- SVM is a supervised machine learning algorithm that uses hyper planes to separate out different classes.



- Two types:
  - Linear
  - Non-linear (Kernel-based)

- My reasons for choosing it:
  - SVMs are suitable for numerical features (✔)
  - SVMs can handle non-linearly separable data (kernel trick)(✔)

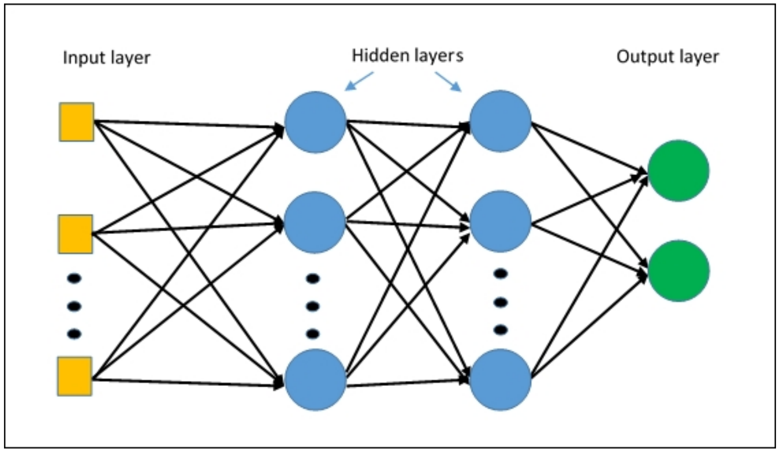
Image Ref: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm

Step 3) Model selection
Deep Learning (MLP)

- Multilayer Perceptrons (MLPs) is a feedforward neural network that consists of multiple fully-connected layers.

- My reasons for choosing it is that

  MLPs are suitable for*:
  - Classification prediction problems (✔)
  - Tabular datasets (✔)



Ref*: https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/
Image Ref: https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron

Step 4) Implementing the models and evaluation

- Evaluation Protocol
- Experiments

The Evaluation Protocol

- K-fold cross validation
  - To avoid overfitting
  - To avoid wasting data as the validation set

- Tuning hyper-parameters
  - I implemented Grid Search using a hyper-parameter optimization tool (sklearn.model_selection.GridSearchCV)

- Evaluation measures
  - Precision, Recall, F1 and Accuracy

Experiments

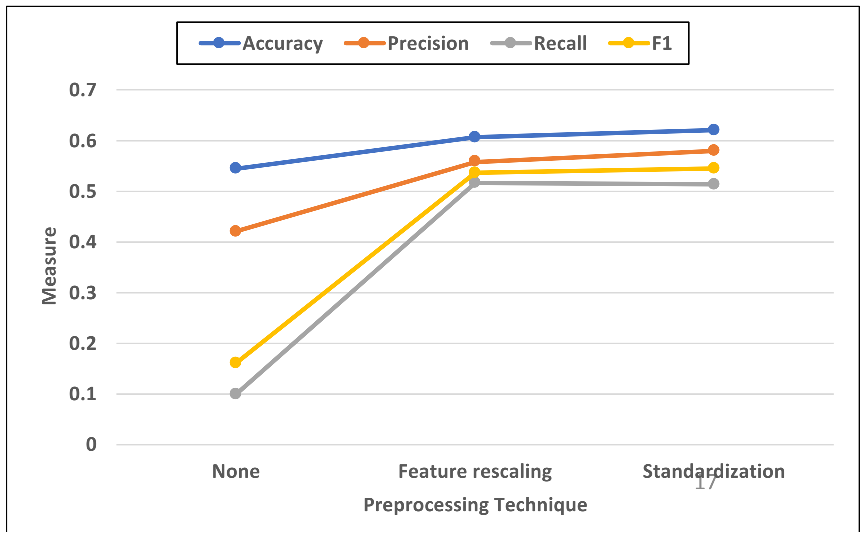
- The impact of feature scaling and standardization
- The impact of dimension reduction techniques
- Hyper-parameters tuning for the classification methods
- Comparing the performance of the classification methods on the test set

The impact of feature scaling and standardization

- Classification Method:
  - SVM
- Feature rescaling using min-max normalization
- Standardization: Centering the feature columns at mean 0 with standard deviation 1

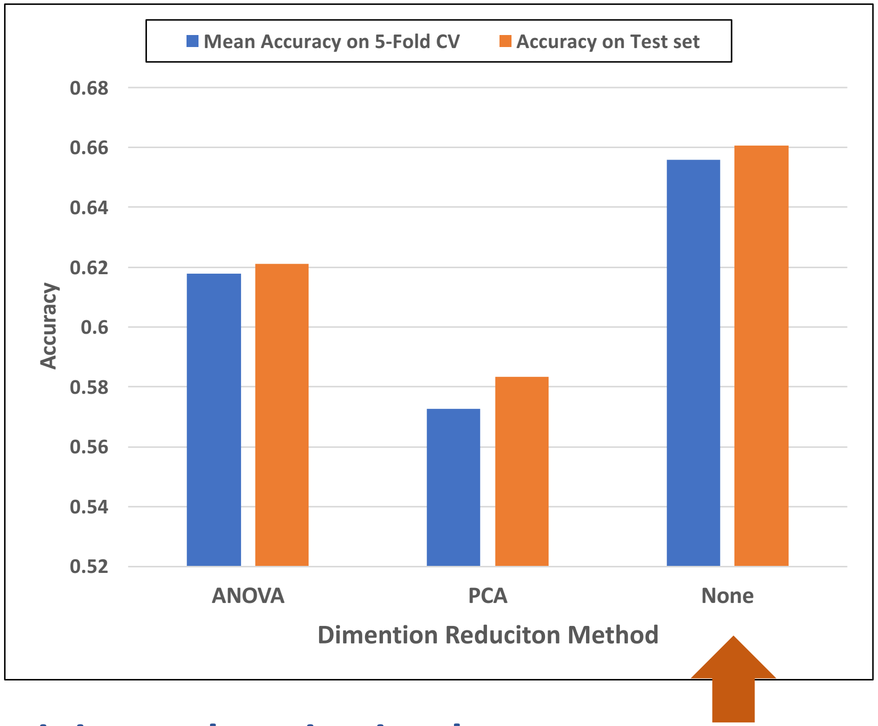
❖ **Standardization has the highest impact on SVM's performance**

❖ **Standardization also decreased SVM's training time significantly.**



The impact of dimension reduction techniques

- Experimental setup:
  - Classification Method:
    - Random Forest
  - 5-fold Cross Validation
  - Evaluation Measure:
    - Training over 5-olds: Mean accuracy
    - Test set: Accuracy using the optimal classifier



❖ **Results: The best performance on both training and testing is when <u>no dimension reduction technique</u> is used.**

Hyper-parameters tuning
Random Forest

- Experimental setup:
  - Grid Search over all combinations of the Random Forest parameters as follows
    - 'bootstrap': [True, False]
    - 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None]
    - 'max_features': ['auto', 'sqrt']
    - 'min_samples_leaf': [1, 2, 4]
    - 'min_samples_split': [2, 5, 10]
    - 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]
  - 5-fold Cross Validation
  - Tuned for both precision and recall measures
- Best parameters set found on development set:
  - For Precision :{'bootstrap': False, 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 800}
  - For Recall:{'bootstrap': False, 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 800}

Random Forest performance on the test set tuned based on precision and recall

- Experimental setup:
  - The Random Forest classifier is trained on the training set in two ways:
    - using optimal parameter values tuned for precision (Prec-based)
    - using optimal parameter values tuned for recall (Recall-based).
  - The training and test sets are obtained using Sklearn **train_test_split** module that makes random partitions for the two subsets.
  - sklearn.metrics.classification_report is used for reporting the model's performance on the test set.

Random Forest performance on the test set tuned based on precision and recall (Cont.)

- Performance on the test set using precision-based optimal parameter

Measure / Class	Precision	Recall	F1-measure	Support
Class 0 (Non-Relevant)	0.67	0.81	0.73	13503
Class 1 (Relevant)	0.66	0.48	0.55	10511

- Performance on the test set using recall-based optimal parameter

Measure / Class	Precision	Recall	F1-measure	Support
Class 0 (Non-Relevant)	0.67	0.79	0.72	13503
Class 1 (Relevant)	0.66	0.50	0.57	10511

Hyper-parameters tuning MLP

- Experimental setup:
  - Grid Search over all combinations of the MLP parameters as follow
    - 'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)]
    - 'activation': ['tanh', 'relu']
    - 'solver': ['sgd', 'adam']
    - 'alpha': [0.0001, 0.05]
    - 'learning_rate': ['constant','adaptive']
  - 5-fold Cross Validation
  - Tuned for both precision and recall measures
- Best parameters set found on development set:
  - For Precision: {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'adam'}
  - For Recall: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'adam'}

MLP performance on the test set
tuned based on precision and recall

- Performance on the test set using precision-based optimal parameter

Class \ Measure	Precision	Recall	F1-measure	Support
Class 0 (Non-Relevant)	0.66	0.80	0.72	13591
Class 1 (Relevant)	0.64	0.46	0.53	10423

- Performance on the test set using recall-based optimal parameter

Class \ Measure	Precision	Recall	F1-measure	Support
Class 0 (Non-Relevant)	0.57	1.00	0.73	13591
Class 1 (Relevant)	0.82	0.02	0.05	10423

Hyper-parameters tuning
SVM

- Experimental setup:
  - Grid Search over all combinations of the SVM parameters as follow
    - {'kernel': ['rbf'], 'gamma': [1e-3, 1e-4]}
    - {‘kernel’:[‘linear], 'C': [1, 10, 100, 1000]}
  - 5-fold Cross Validation
  - Tuned for both precision and recall measures
- Best parameters set found on development set:
  - For Precision: {'gamma': 0.001, 'kernel': 'rbf'}
  - For Recall: {'gamma': 0.001, 'kernel': 'rbf'}
  - ❖ **The optimal parameter values for precision and recall are the same**

SVM performance on the test set tuned based on precision and recall

- Performance on the test set using precision/recall-based optimal parameter

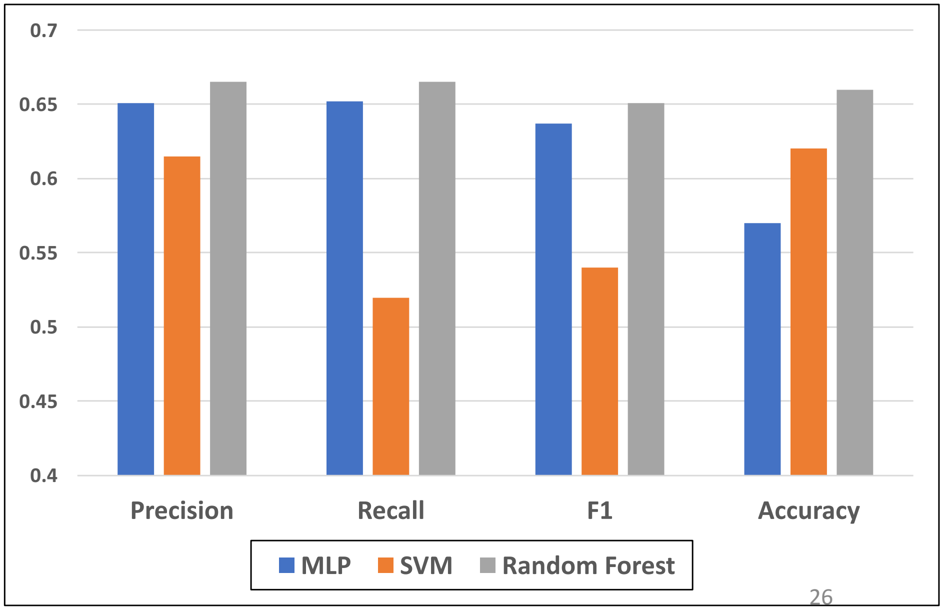
Measure / Class	Precision	Recall	F1-measure	Support
Class 0 (Non-Relevant)	**0.65**	**0.70**	**0.68**	**13562**
Class 1 (Relevant)	**0.57**	**0.52**	**0.54**	**10452**

Comparing the performance of the classification methods on test set

- All models' performances on the test set using the optimal parameters tuned for precision

- Random Forest outperforms other two models in terms of all measures.

- In terms of running time, Random Forest trained faster than MLP and SVM.



Conclusion and final recommendations

- Standardization decreased the training time of the models.
- The top 5 important features in the dataset according to ANOVA test are sig2, sig7, sig1, sig6 and sig5 respectively.
- PCA and ANOVA feature reduction methods do not improve the classification performance on this dataset.
  - Due to the small number of features (13) compared to the number of instances (80047), dimension reduction increases bias and can cause under-fitting.
- The performance of Random Forest and MLP on the test set is better than SVM in terms of Precision, Recall, F1 and accuracy.
- The Random Forest model trained faster than MLP and SVM.

Conclusion and final recommendations

- The per-class evaluation of all three methods show that the relevant document detection (Class 1) is a harder problem compared to identifying non-relevant documents.
- The relevance optimization problem is similar to the problem of document retrieval in response to query. Therefore, the ranking of documents would be a useful information in our dataset.
- The forth method used in this project is a learning to rank algorithm called SVMRank. The idea of learning the ranking of the documents for each query is an effective approach for the search engine optimization problem.

Python implementation and visual walk-through of the outputs

- The python script attached to this report can be executed to regenerate the results reported on the slides.
- In the first step for running the program, you may read the ReadMe file or enter the following command to see the instructions:
  - ➢ python3 RelevancePrediction.py –h
- You can pass the following arguments with your command when running the code:
  - -d: The dimension reduction method. (available options are "PCA" and "AnovaTest")
  - -m: The classification model. (available options are "SVM", "MLP" and "RandomForest")
- Example Command:
  - ➢ python3 RelevancePrediction.py –d "AnovaTest" –m "RandomForest"

Python implementation and visual walk-through of the outputs (Cont.)

- The first function called after entering the command is setup()

- It takes the arguments from input and set the corresponding variables.

```
############################## Setup Function ##############################
def setups(argv):

    if len(argv) == 0:
        print('You must pass some parameters. Use \"-h\" to help.')
        return
    if len(argv) == 1 and argv[0] == '-h':
        f = open('ReadMe.txt', 'r')
        print(f.read())
        f.close()
        return
    inputFileName = "Dataset.csv"#default value
    outputFolderName = "Results"#default value
    DimRedMethod = "None"#default value
    ClassificationMethod = "RandomForest"#default value

    opts, args = getopt.getopt(sys.argv[1:],"d:m:")
    for opt,arg in opts:
        if opt == '-d':
            DimRedMethod = arg
        elif opt == '-m':
            ClassificationMethod = arg
        else:
            print("Usage: %s -d DimRedMethod -m ClassificationMethod" % sys.argv[0])

    ########## Step One: Preprocessing ###########
    print("...............Preprocessings...............")
    preprocessedInputDir = preprocessing(inputFileName)
    print("Done!")

    ###########  SVM Classification ###########
    if "SVM" in ClassificationMethod:
        print(".............SVM Classification..............")
        SVMClassification(preprocessedInputDir,outputFolderName,DimRedMethod)
        print("Done!")


    ###########  RandomForest Classification ###########
    elif "RandomForest" in ClassificationMethod:
        print(".........Random Forest Classification..........")
        RandomForestClassification(preprocessedInputDir,outputFolderName,DimRedMethod)
        print("Done!")
```

Python implementation and visual walk-through of the outputs (Cont.)

- Based on the arguments, the next functions are called.

- The next called function is preprocessing.

```python

############################## Preprocessings Function ##############################
def preprocessing(inputFileDir):

    ################# Duplicate Removal ###############
    print("start preprocessing!")
    print("inputFileDir: "+str(inputFileDir))
    dataframe = pd.read_csv(inputFileDir, sep=",")
    dataframe.drop_duplicates(subset =['query_id','url_id'], keep = 'first', inplace = True)

    ################# Feature Scaling ################
    array = dataframe.values
    # separate array into input and output components
    X = array[:,[2,4,5,6,7,8,9,10,11]]
    Y = array[:,-1]
    scaler = MinMaxScaler(feature_range=(0, 1))
    rescaledX = scaler.fit_transform(X)

    ################# Standardization ###############
    # Centering the feature columns at mean 0 with standard deviation
    scaler = StandardScaler().fit(rescaledX)
    StandardizedX = scaler.transform(rescaledX)

    preprocX = np.concatenate((array[:,0:2],StandardizedX[:,[0]],array[:,[3]],StandardizedX[:,1:9],array[:,[12]]), axis=1)
    preprocessedFileDir = ''.join("Preprocessed_"+inputFileDir)
    with open(preprocessedFileDir,"w") as my_csv:
        csvWriter = csv.writer(my_csv,delimiter=',')
        csvWriter.writerow(['query_id','url_id','query_length','is_homepage','sig1','sig2','sig3','sig4','sig5','sig6','sig7','s
        csvWriter.writerows(preprocX)
    return preprocessedFileDir
```

Python implementation and visual walk-through of the outputs (Cont.)

- Based on the arguments, the next functions are called.

- The next called function is preprocessing.
- In preprocessing function,
  - The duplicate instances are removed.
  - Features are rescaled using min-max normalization.
  - Then, features are standardized.

```
############################## Preprocessings Function ##############################
def preprocessing(inputFileDir):

    ################# Duplicate Removal ###############
    print("start preprocessing!")
    print("inputFileDir: "+str(inputFileDir))
    dataframe = pd.read_csv(inputFileDir, sep=",")
    dataframe.drop_duplicates(subset =['query_id','url_id'], keep = 'first', inplace = True)

    ################## Feature Scaling ################
    array = dataframe.values
    # separate array into input and output components
    X = array[:,[2,4,5,6,7,8,9,10,11]]
    Y = array[:,-1]
    scaler = MinMaxScaler(feature_range=(0, 1))
    rescaledX = scaler.fit_transform(X)

    ################## Standardization ################
    # Centering the feature columns at mean 0 with standard deviation
    scaler = StandardScaler().fit(rescaledX)
    StandardizedX = scaler.transform(rescaledX)

    preprocX = np.concatenate((array[:,0:2],StandardizedX[:,[0]],array[:,[3]],StandardizedX[:,1:9],array[:,[12]]), axis=1)
    preprocessedFileDir = ''.join("Preprocessed_"+inputFileDir)
    with open(preprocessedFileDir,"w") as my_csv:
        csvWriter = csv.writer(my_csv,delimiter=',')
        csvWriter.writerow(['query_id','url_id','query_length','is_homepage','sig1','sig2','sig3','sig4','sig5','sig6','sig7','s
        csvWriter.writerows(preprocX)
    return preprocessedFileDir
```

Python implementation and visual walk-through of the outputs (Cont.)

- After preprocessing, according to the desired classification method set in the arguments, one of the

three classification functions are called.

  - SVM
  - Random Forest
  - MLP

```python

    opts, args = getopt.getopt(sys.argv[1:],"d:m:")
    for opt,arg in opts:
        if opt == '-d':
            DimRedMethod = arg
        elif opt == '-m':
            ClassificationMethod = arg
        else:
            print("Usage: %s -d DimRedMethod -m ClassificationMethod" % sys.argv[0])

    ########### Step One: Preprocessing ############
    print("...............Preprocessings................")
    preprocessedInputDir = preprocessing(inputFileName)
    print("Done!")

    ###########  SVM Classification ############
    if "SVM" in ClassificationMethod:
        print("..............SVM Classification..............")
        SVMClassification(preprocessedInputDir,outputFolderName,DimRedMethod)
        print("Done!")


    ###########  RandomForest Classification ############
    elif "RandomForest" in ClassificationMethod:
        print(".........Random Forest Classification..........")
        RandomForestClassification(preprocessedInputDir,outputFolderName,DimRedMethod)
        print("Done!")


    ###########  MLP Classification ############
    elif "MLP" in ClassificationMethod:
        print("..........MLP Classification..........")
        MLPClassification(preprocessedInputDir,outputFolderName,DimRedMethod)
        print("Done!")


if __name__ == "__main__":
    setups(sys.argv[1:])
```

SVM Classification

- First, dimension reduction
is applied (if set in command options)
- Then, hyper-parameter tuning starts
by defining the values of SVM parameters
to be searched using Sklearn GridSearchCV
- The GridSearch is performed using
5-fold cross validation over training set
for both Precision and Recall
- After finding the optimal parameter
values, the model is trained on the
training set using the tuned parameter values
and tested on the testing set.

```python

########################## SVM Classification ############################
def SVMClassification(preprocessedInputDir,outputFolderName,DimRedMethod):

    dataframe = pd.read_csv(preprocessedInputDir, sep=",")
    array = dataframe.values
    X = array[:,0:12]
    Y = array[:,-1]

    #writing to outPutFile
    outFile = open(outputFolderName+"/SVMResults.txt","w")

    #splitting into train and test data
    X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.3)

    #### Dimension Reduction using PCA ####
    if(DimRedMethod == "PCA"):
        pca = PCA(.95)
        pca.fit(X_train)
        X_train = pca.transform(X_train)
        X_test = pca.transform(X_test)

    #### Dimension Reduction using Anova test ####
    if (DimRedMethod == "AnovaTest"):
        test = SelectKBest(score_func=f_classif, k=5)
        fit = test.fit(X_train, y_train)
        X_train = fit.transform(X_train)
        X_test = fit.transform(X_test)

    ##### SVM Parameter Tuning #####
    # Set the parameters by cross-validation
    tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],'C': [1, 10, 100, 1000]},
                        {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
    scores = ['precision', 'recall']
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        #clf = GridSearchCV(svm.NuSVC(), tuned_parameters, scoring='%s_macro' % score)
        #clf = GridSearchCV(svm.LinearSVC(), tuned_parameters, scoring='%s_macro' % score)
        clf = GridSearchCV(SVC(), tuned_parameters, scoring='%s_macro' % score,n_jobs=3)
        clf.fit(X_train, y_train)
        print()
        print("Best parameters set found on development set:")
        print()
        print(clf.best_params_)
        print()
        print("Grid scores on development set:")
        print()
        means = clf.cv_results_['mean_test_score']
        stds = clf.cv_results_['std_test_score']
        for mean, std, params in zip(means, stds, clf.cv_results_['params']):
            print("%0.3f (+/-%0.03f) for %r"
                  % (mean, std * 2, params))
        print()

        print("Detailed classification report:")
        print()
        print("The model is trained on the full development set.")
        print("The scores are computed on the full evaluation set.")
        print()
        y_true, y_pred = y_test, clf.predict(X_test)
        print(classification_report(y_true, y_pred))
        print()
        outFile.write("the model using optimal parameter value on the test set"+"\n")
        outFile.write(classification_report(y_true, y_pred))
    outFile.close()
```

MLP Classification

- The whole process is the same as SVM, unless the MLP classification module and also MLP parameters for tuning are different.
- Three different network architectures has been tested with different number of layers and neurons.
- Two activation functions are tested including tanh and ReLU
- The result are written in a file in a direcotry called ‘Results'

```
#################### Fully Connected MLP Classification ########################
def MLPClassification(preprocessedInputDir,outputFolderName,DimRedMethod):
    dataframe = pd.read_csv(preprocessedInputDir, sep=",")
    array = dataframe.values
    X = array[:,0:12]
    Y = array[:,-1]
    X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.3)
    #writing to outPutFile
    outFile = open(outputFolderName+"/MLPResults.txt","w")

    #### Dimension Reduction using PCA ####
    if(DimRedMethod == "PCA"):
        pca = PCA(.95)
        pca.fit(X_train)
        X_train = pca.transform(X_train)
        X_test = pca.transform(X_test)
        print("After PCA")
    #### Dimension Reduction using Anova test ####
    elif (DimRedMethod == "AnovaTest"):
        test = SelectKBest(score_func=f_classif, k=5)
        fit = test.fit(X_train, y_train)
        X_train = fit.transform(X_train)
        X_test = fit.transform(X_test)

    ##### MLP Parameter Tuning #####
    tuned_parameters = {
        'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
        'activation': ['tanh', 'relu'],
        'solver': ['sgd', 'adam'],
        'alpha': [0.0001, 0.05],
        'learning_rate': ['constant','adaptive'],
    }
    scores = ['precision', 'recall']
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        clf = GridSearchCV(MLPClassifier(), tuned_parameters, scoring='%s_macro' % score,n_jobs=3)
        clf.fit(X_train, y_train)
        print()
        print("Best parameters set found on development set:")
        print()
        print(clf.best_params_)
        print()
        print("Grid scores on development set:")
        print()
        means = clf.cv_results_['mean_test_score']
        stds = clf.cv_results_['std_test_score']
        for mean, std, params in zip(means, stds, clf.cv_results_['params']):
            print("%0.3f (+/-%0.03f) for %r"
                  % (mean, std * 2, params))
        print()
        print("Detailed classification report:")
        print()
        print("The model is trained on the full development set.")
        print("The scores are computed on the full evaluation set.")
        print()
        #Run the model using optimal parameter value on the test set
        y_true, y_pred = y_test, clf.predict(X_test)
        print(classification_report(y_true, y_pred))
        print()
        outFile.write("the model using optimal parameter value on the test set"+"\n")
        outFile.write(classification_report(y_true, y_pred))
    outFile.close()
```

Random Forest Classification

- Due to the large number of hyper-parameters in Random Forest, in order to reduce the running time, I utilized parallel processing by increasing the value of n_jobs parameter in Sklearn.

```python
#################### Random Forest Classification ########################
def RandomForestClassification(preprocessedInputDir,outputFolderName,DimRedMethod):
    dataframe = pd.read_csv(preprocessedInputDir, sep=",")
    array = dataframe.values
    X = array[:,0:12]
    Y = array[:,-1]

    #writing to outPutFile
    outFile = open(outputFolderName+"/RandomForestResults.txt","w")

    #splitting into train and test data
    X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.3)

    #### Dimension Reduction using PCA ####
    if(DimRedMethod == "PCA"):
        pca = PCA(.95)
        pca.fit(X_train)
        X_train = pca.transform(X_train)
        X_test = pca.transform(X_test)

    #### Dimension Reduction using Anova test ####
    if (DimRedMethod == "AnovaTest"):
        test = SelectKBest(score_func=f_classif, k=5)
        fit = test.fit(X_train, y_train)
        X_train = fit.transform(X_train)
        X_test = fit.transform(X_test)

    ##### RF Parameter Tuning #####
    #hyper parameter tuning. Selecting best n_estimators
    tuned_parameters = {'bootstrap': [True, False],
    'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
    scores = ['precision', 'recall']
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        clf = GridSearchCV(RandomForestClassifier(), tuned_parameters, scoring='%s_macro' % score, n_jobs=3)
        clf.fit(X_train, y_train)
        print()
        print("Best parameters set found on development set:")
        print()
        print(clf.best_params_)
        print()
        print("Grid scores on development set:")
        print()
        means = clf.cv_results_['mean_test_score']
        stds = clf.cv_results_['std_test_score']
        for mean, std, params in zip(means, stds, clf.cv_results_['params']):
            print("%0.3f (+/-%0.03f) for %r"
                  % (mean, std * 2, params))
        print()
        print("Detailed classification report:")
        print()
        print("The model is trained on the full development set.")
        print("The scores are computed on the full evaluation set.")
        print()
        #Run the model using optimal parameter value on the test set
        y_true, y_pred = y_test, clf.predict(X_test)
        print(classification_report(y_true, y_pred))
        print()
        outFile.write("the model using optimal parameter value on the test set"+"\n")
        outFile.write(classification_report(y_true, y_pred))
    outFile.close()
```

Thanks for your attention