Machine Learning (CS60050)

Assignment-2

Parth Jindal, Pranav Rajput

Group : 59

Abstract

This report contains an analysis for a spam-filter trained on the **Spam/Ham Dataset**.The following tasks are covered in the report:

1. Build a Spam Filter classifier using KNearestNeighbour algorithm
2. Find the accuracy over a random 80/20 split
3. Compare the performance across different distance metrics
4. Vary k value for all metrics and observe underlying trends

Index

Data Analysis

The Spam-dataset is a categorical dataset classifying a mail into 2 categories **[‘ham’, ‘spam’]** The frequency table for each class is given below:

class	N	%
ham	3672	71.01 %
spam	1499	28.99 %

The dataset consists of 5171 mails with each mail having a text body as its single attribute. To vectorize/tokenize the text associated, TF-IDF vectorizer is used to build a vector representation for all the mail-bodies.

TF-IDF vectorizer

Term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

$$tf(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

$t: term \;\; d: document, f_{t,d}: \; frequency\ count\ of\ term\ t\ in\ document\ d$

Note: Using tf-idf vectorization produces sparse vector encoding for mail-texts. Other vectorization techniques (based on Neural-nets) can also be used in place of tf-idf.

Accuracy of the k-NN classifier

The following is the statistical report achieved on using a k-NN classifier on a random 80/20 split of dataset.

Configuration:

k: 30

metric: cosine-similarity

tf-idf vectorization max-features: 1000

Metric	**Result**
Accuracy	96.04%
Precision	93.91%
Recall	93.02%
F1	93.46%

The 95% CI for accuracy: 94.85 % - 97.22 %

The following is the confusion matrix observed for the k-NN classifier on the said task:

Confusion Matrix

True \ Prediction	**Ham**	**Spam**
Ham	701	19
Spam	22	293

Comparison of distance/similarity metrics

The following distance metrics were used to train k-NN classifiers:

Cosine similarity:

$$similarity(a, b) = \frac{a \bullet b}{||a|| \times ||b||}$$

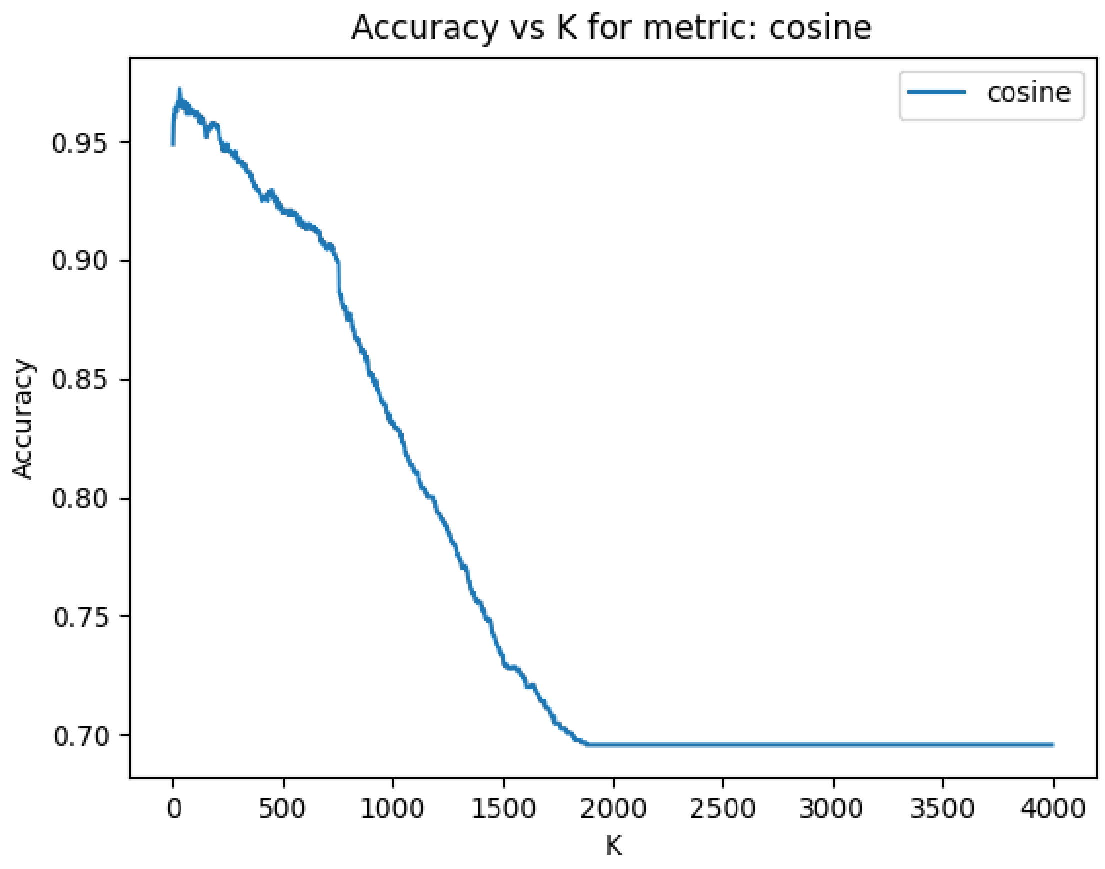
Cosine similarity measures how close two vectors are in the Euclidean space by their angular distance. The best accuracy using cosine similarity was achieved for k = **31,** accuracy = 97.20 %.

Metric	Result
Accuracy	97.20%
Precision	95.25%
Recall	95.56%
F1	95.40%

True \ Prediction	Ham	Spam
Ham	705	15
Spam	14	301

We can clearly see cosine similarity performing well on the prediction task with high precision and recall scores as well. With TF-IDF producing highly sparse vector representations, cosine similarity is easily able to distinguish mails, sharing no important keywords with their similarity being 0. (orthogonal vectors).

As observed in the graph for *Accuracy vs k* for cosine metric, The accuracy reaches its maximum for **k = 31** after which it starts to reduce and plateaus around **k = 1500** at which accuracy is **69.57 %**



<u>**Euclidean Distance:**</u>

$$Distance(a, b) = \sum_{i=0}^{n} (a_i - b_i)^2$$

Euclidean distance simply measures their spatial distance in the euclidean space. The best accuracy using euclidean distance was achieved for k = **31,** accuracy = 97.20 %.

Metric	**Result**
Accuracy	97.20%
Precision	95.25%
Recall	95.56%
F1	95.40%

True \ Prediction	Ham	Spam
Ham	705	15
Spam	14	301

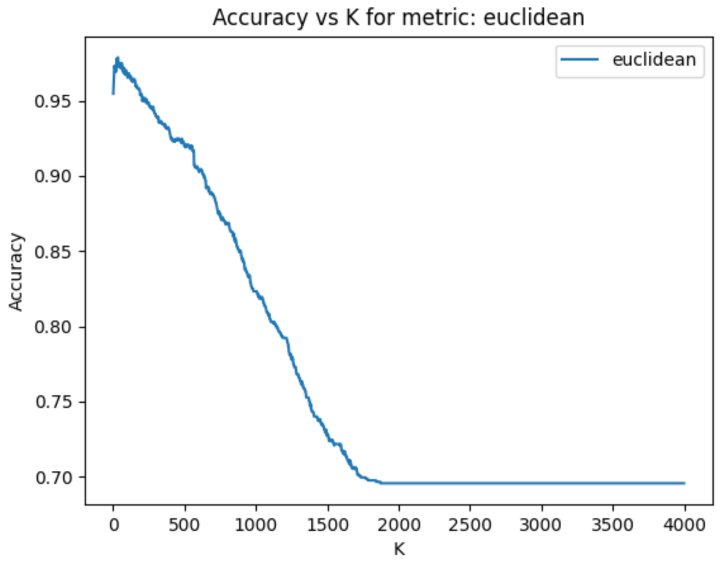
Note how the statistics achieved for cosine similarity and euclidean distance are the same. In fact with fair assumptions under the tf-idf vectorization (sparse vector representations, all non-negative attribute associations, **normalized vector representations**), Both these metrics will produce the nearest-neighbour rankings, hence the classification is same for both. Proof:

Let u, v be two unit vectors $\in R^{n}$.

Cosine-similarity$(u, v) = \sum_{i=0}^{n} (u_i * v_i)$

Euclidean distance $(u, v)^2 = 2 - 2 * (\sum_{i=0}^{n} (u_i * v_i))$

As seen, both these metrics rank the vectors in the same order under the assumptions taken above. As observed in the graph for *Accuracy vs k* for euclidean metric, The accuracy reaches its maximum for **k = 31** after which it starts to reduce and plateaus around **k = 1500** at which accuracy is **69.57 %**



Manhattan Distance:

$$Distance(a, b) = \sum_{i=0}^{n} |(a_i - b_i)|$$

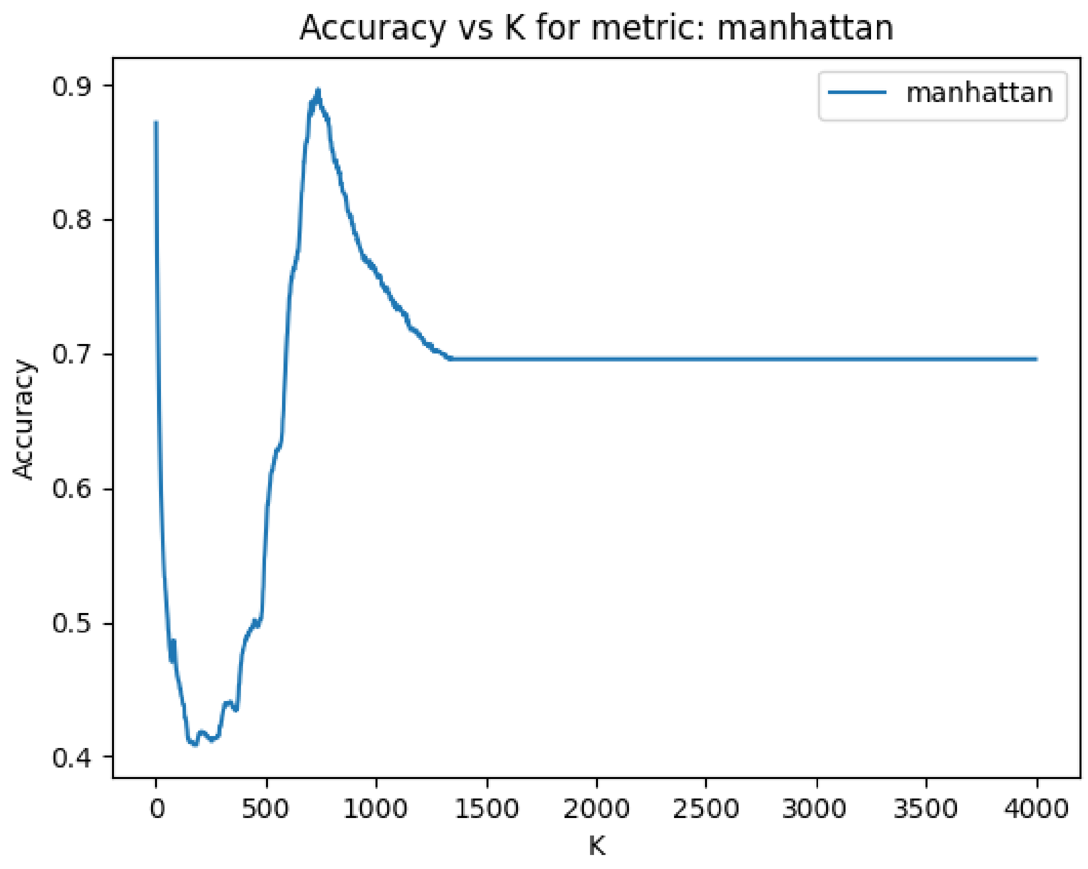
Manhattan distance measures the distance between two vectors in the euclidean space if travelled along the unit basis vectors. The best accuracy using cosine similarity was achieved for k = **737,** accuracy = 89.66%.

Metric	**Result**
Accuracy	89.66%
Precision	81.90%
Recall	84.76%
F1	83.31%

True \ Prediction	**Ham**	**Spam**
Ham	661	59
Spam	48	267

As observed, Manhattan distance performs worse than other metrics in fairly all statistical measures. The best accuracy for manhattan distance is also achieved at a very high k value strongly suggesting how bias inherent in dataset due to class-imbalance is coming to play.

As observed in the graph for *Accuracy vs k* for manhattan metric, The accuracy reaches its maximum for **k = 737** after which it starts to reduce and plateaus around **k = 1500** at which accuracy is **69.57 %**



Conclusions

As observed, euclidean and cosine metrics perform the same on the given dataset achieving high accuracy, precision and recall scores as evident from the confusion matrix, whereas manhattan metric performs worse than them and also has very low precision and recall score suggesting at other factors at play in classification apart from the KNN classifier itself.

Steps to run the code:

1. Inside the submission directory, use pip install -e requirements.txt to install the dependencies
2. Run python main.py

Procedure and Implementation details

The following packages have been used in building the decision tree:

Numpy: for reading and writing arrays

Pandas: for reading csv dataset files

Matplotlib: for creating graphs and plotting

Major Classes:

KNN

This is the main class to build the KNearestNeighbour classifier

x_train: training-dataset

y_train: training-labels

n_neighbours: no. of neighbours to be used during classification

weighted: whether to use weighted decision making

metric: the distance metric to be used (can be a either of the three above or a user-defined function)

method: brute or cached (in cached mode, predictions can be made directly from the dst_mtrx provided without computing it again)

dst_mtrx: optional distance matrix for cached method

Functions:

predict(): Predicts the labels for the given input data

neighbours(): Computes the distance matrix and sorted-indices for n nearest-neighbours and returns the same.

_compute_chunked_distance() : Computes the distance matrix by creating chunks of input data of appropriate size to fit in memory and joins all the chunks and returns the distance matrix

_predict_one(): Predicts the class label for a single training input (helper function)

MailDataset

Class to load the mail dataset stored in the csv file.

root: root-directory for the csv file
transform: Callable function that can be used to apply transformations to the dataset if needed (used in vectorization)

Vectorizer

Wrapper class for TF-IDF vectorizer with data marshalling and other wrapping.

Major functions:

main(): Main function with cmdline arguments to run a single metric and produce statistical report for the same.

compare_metrics: Wrapper function to compute the varying accuracy with k value for given list of metrics and produce a graph for all.

Helper functions:

get_cosine_score: computes the cosine score for a given set of vectors X with respect to all vectors present in Y

minkowski_distance: computes the pth order distance between two sets of vectors X and Y.

split_dataset : function to randomly split the given dataset in 80:20 ratio for training : validation. Returns the training dataset and validation dataset

shuffle_dataset : function to shuffle the dataset and target labels. Returns the shuffled dataset**.**

precision_score : function to compute the precision score for a set of labels given as input. Returns the precision score.

recall_score : function to compute the recall score for a set of labels given as input. Returns the recall score.

get_metrics : function to return the attribute values for an instance given as input present in the dataset. Returns the attribute values for the instance

confusion_matrix : function to compute the confusion matrix for a set of labels given as input. Returns the confusion matrix.

f1_score : function to compute the F1 score for a set of labels given as input. Returns the F1 score.