Al proyecto le agregué 5 clases nuevas.

1) ***Dict***: tiene 4 métodos el método
    - ConvertToDict acepta como parámetro una lista de *strings* y devuelve un diccionario donde la clave de cada elemento es una palabra de la lista y cada valor lo inicio con 0.
    - Dos métodos SortDictionary, uno con claves de tipo *string* y otro con claves de tipo *in*t, cada método ordena los elementos del diccionario según los valores que son de tipo *double.*
    - WordsInVocabulary que acepta una lista de *strings* y un diccionario, no devuelve nada, sino modifica el diccionario y las palabras que coincidan con las de la lista toman valores de 1.
2) ***Coseno***: consta de 5 métodos el método
    - Similitud que devuelve un valor de tipo *double* que representa la similitud de 2 vectores que recibe como parámetros aplicando la fórmula del producto punto y calculando las magnitudes de cada uno, mientras mayor sea el número más semejantes son los vectores.
    - WordsInDoc que recibe una lista de palabras a comparar con el contenido de un documento y devuelve una nueva lista con las palabras de la lista del parámetro que están en él.
    - Positions recibe esas palabras que coincidieron y la lista de todas las palabras en total y devuelve una lista nueva con las posiciones de esas palabras en la lista de todas las palabras.
    - TF_IDFWordsQueryInDoc que recibe la matriz con los TF_IDF,la lista de las posiciones y el número del documento que contenía a esas palabras y devuelve una lista de los TF_IDF de esas palabras en ese documento
    - Snippet que recibe a la palabra con mayor TF_IDF y al documento con el contenido original, el método crea una lista separando las cadenas del texto por un espacio, y aplico el método LevenshteinDistance de la clase Suggestion entre cada porción del texto y esa palabra, la distancia menor será la porción más parecida a la palabra, a esa porción le sumo una cantidad a la derecha y a la izquierda y esa cadena la retorna como el valor del snippet.
3) ***TF_IDF*** :tiene 2 métodos
    - TF que devuelve una matriz ,con los TF de todas las palabras en cada documento
    - TFIDF que devuelve una matriz con los valores TFIDF de todas las palabras en todos los documentos.
4) ***Suggestion*** :que tiene 2 métodos
    - LevenshteinDistance explicado anteriormente
    - ConvertDocArray que me lleva el contenido de un documento a un arrray de string.
5) ***Matrix*** : En el constructor acepta una matriz de tipo double y tiene una propiedad Row con la parte set privada que devuelve la matriz. Tiene .Contiene 9 métodos cada uno realiza la operación algebraica que indica el nombre

- *Suma:* suma 2 matrices
- *Producto*: multiplica 2 matrices
- *Traspuesta*: calcula la traspuesta de una matriz
- *Deter* :Calcula el determinante de orden n de una matriz
- *MatrizXVector*: multiplica una matriz por un vector
- *MatrizXEscalar*:multiplica una matriz por un escalar
- *VectorSuma*: suma 2 vectores
- *VectorMult*: multiplica 2 vectores
- *VectorXescalar*: multiplica un vector por un escalar

En el método Query de la clase ***Moogle*** lo primero que hice es llevar el *query* a una lista en minúsculas y quitarle todos los caracteres que no coincidan con letras o números. Después busco cada documento de la carpeta Content y voy creando las listas con el contenido de cada documento, una con el contenido original y otra con el contenido modificado después de quitarles los caracteres que no sean números y letras, además de la lista de los títulos sin extensiones. Si alguno de los documentos esta vacío se elimina ese documento de cada lista. Después creo la lista de todas las palabras q hay y elimino las que sean iguales y creo las matrices de los TFIDF de cada una .Creo el diccionario Vocabulary con todas las palabras y se lo paso como parámetro al método WordsInVocabulary con las palabras del *query.* Tomo los valores del diccionario y creo una lista con ellos, y voy aplicando el método Similitud con cada columna de la matriz TF_IDF y cada valor se lo agrego a la lista de Scores. Si todos los valores de la lista son 0, es porque todas las palabras aparecen en todos los documentos y en ese caso cálculo la similitud con cada columna de la matrix TF para quedarme con el documento de mayor TF. Después voy agregando cada Score distinto de 0 con el título correspondiente a un diccionario ScoresTitle y eso mismo hago con el diccionario ScoreDocs que contiene como clave los números de orden de los documentos. Le aplico el método SortDictionary a cada uno y la lista de los títulos y los scores las modifico con la lista de los títulos y scores de los diccionarios en orden invertido. Creo una lista invertida DocNumbers también con los números de los documentos. Con un ciclo *for* recorro todos los documentos que coincidan con la lista DocNumbers y en cada uno busco la palabra de mayor de mayor TF_IDF, calculo el snippet y lo agrego a una lista de Snippet. Creo un array item de tipo SearchItem con cada uno de los títulos,snippets y scores correspondientes.Para buscar la sugerencia creo un array con el *query* y un array tester de igual longitud para recorrer cada documento; declaro una variable de apoyo con valor 5.Llevo cada documento a un array y con el tester convertido a *string* texte voy a compararlo con el *query* por el método LevenshteinDistance, me quedo con el string mas similar al *query*, y si no encuentra una similitud <= 5,la sugerencia será una cadena vacía. Por

último le paso el array ítem y la sugerencia como parámetros al objeto de tipo *SearchResult* que será devuelto