Project in Bioinformatics 236524

Introduction

In this project I studied RNA Binding Proteins (RBP) and used Machine Learning algorithms to predict their binding site location. Proteins are large biomolecules responsible for most of the functionality of all biological systems including the human body. Biological systems vary from single cells to all living organisms and as such, proteins diverse vastly by their functionality. To name a few, there are enzymes that catalyze chemical reactions, structural proteins that are mainly used for their stiffness and rigidity and signaling proteins like DNA binding proteins or this project focus, RNA binding proteins.

RBPs are an essential part of many cellular processes and understanding it's biological mechanism could be key in future research. Like the name suggests, these proteins bind to RNA and "read" its genetic information to know what role (or what function) they are assigned to do. If we consider the fact that RNA is an extremely long thread of information, RBPs need to navigate along the RNA to find the information they are looking for, and so they do, RBPs find their way to the relevant site on the RNA strand consistently and precisely, and bind to it. This mechanism is still unknown, and the goal is to shed more light on this mechanism. In this project we will shortly review a current model that predicts an RBP binding site using a statistical approach and then we will try and predict the binding site using an SVM model.

A brief overview of RBPmap[1]

RBPmap analyzes a given proteins motif and uses this information to calculate predictive binding sites along a given RNA sequence. A motif of a protein is a sequence of nucleotides that are associated with the protein, meaning that the protein usually binds to this sequence in the RNA. For every protein there may be a few different sequences that the protein binds to, in various rates. For this reason, there is a variety of representations that can be used to describe a proteins motif. There are two representations that can be used as an input to RBPmap- A consensus sequence and a PSSM matrix. Consensus sequence of a motif is simply the most frequent sequence the protein binds to, this is a less informative representation of the motif. A more informative representation is a position-specific scoring matrix (PSSM) which calculates the log odds probability of each nucleotide occurrence in each position of the sequence based on a background model (which is for most cases a uniform distribution over all four nucleotides in every position). Using the motif representation and the given RNA sequence RBPmap computes the putative binding sites in a few steps.

Firstly, RBPmap maps the given RNA sequence to genomic coordinates and uses this information to categorize the sequence to different genomic regions: intronic regions, internal exons, exons in 5’ and 3’ UTR regions, non-coding RNA and mid-intron regions. Next, RBPmap calculates a score for each sequence using the consensus sequence or the PSSM matrix according to user input. RBPmap compares all the scores in the sequence (matching all the overlapping sites in the sequence) to a background model and calculates Z-scores and

[1] Paz I, Kosti I, Ares M Jr, Cline M, Mandel-Gutfreund Y. (2014) RBPmap: a web server for mapping binding sites of RNA-binding proteins. Nucleic Acids Res., 2014.

P-values. We then filter out all sites using two different thresholds: significant threshold used for classifying putative binding sites on the sequence and suboptimal threshold. The suboptimal threshold is used for a weighted rank score for all putative binding sites. From experimental results, we know that proteins motif tend to cluster around the binding site, we use this to make a prediction. By using the closest suboptimal sites to a putative binding site, we rank them by their score and take a weighted average of their score according to their rank. This score is compared to a region-specific background model, and we receive Z-scores and P-values once again. These values are the final scores of the putative binding sites. Binding sites with P-values lower than 0.05 are predicted binding sites. Optionally, RBPmap allows users to add an additional step based on conservation filtering. This filtering might improve prediction due to regulatory regions that are evolutionary conserved, this means that we might favor binding sites with high conservation scores.

Overview of SVM model prediction

Firstly, we retrieved our datasets from ENCODE consortium[2], Gene Yeo, UCSD lab. We downloaded the call sets from the ENCODE portal (Sloan et al. 2016) (https://www.encodeproject.org/) with the following identifiers: ENCSR432XUP, ENCSR321PWZ, ENCSR661ICQ, ENCSR366YOG, ENCSR570WLM, ENCSR489ABS, ENCSR724RDN, ENCSR795CAI. We used the bed files from the datasets and labeled each sequence by using the experimental score given to it. This score indicates the likelihood of a certain protein (the protein it was tested for in the lab) to bond to this sequence. Once labeled, we used the genomic coordinates to convert sequences to nucleotide strings using Bed2Fasta tool from MEME Suit[3]. We created strings of 200 nucleotide length around the center of the sequence given from the experiment results. We used these strings along side their labels to train and test an SVM model (we used Sci-kit learn[4] SVM model) that predicts whether a certain protein binds to a given sequence.

To use an SVM model, we need to transform the strings into numeric values that can be used as input features of the SVM model. A trivial transformation where we simply swap A to 1, C to 2, G to 3 and T to 4 wont work as this gives bigger values to arbitrary letters. Instead, we want to use a representation where each letter gets an equally sized value. We can use vectors on the unit circle for instance or maybe use one-hot encoding where the letter A transforms to 1000 and the letter C transforms to 0100 etc. We tested both representations and found out that the one-hot encoding representation gives better results in terms of

[2] ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. Nature. 2012 Sep 6;489(7414):57-74. doi: 10.1038/nature11247. PMID: 22955616; PMCID: PMC3439153.

Luo Y, Hitz BC, Gabdank I, Hilton JA, Kagda MS, Lam B, Myers Z, Sud P, Jou J, Lin K, Baymuradov UK, Graham K, Litton C, Miyasato SR, Strattan JS, Jolanki O, Lee JW, Tanaka FY, Adenekan P, O'Neill E, Cherry JM. New developments on the Encyclopedia of DNA Elements (ENCODE) data portal. Nucleic Acids Res. 2020 Jan 8;48(D1):D882-D889. doi: 10.1093/nar/gkz1062. PMID: 31713622; PMCID: PMC7061942.

Hitz BC, Jin-Wook L, Jolanki O, Kagda MS, Graham K, Sud P, Gabdank I, Strattan JS, Sloan CA, Dreszer T, Rowe LD, Podduturi NR, Malladi VS, Chan ET, Davidson JM, Ho M, Miyasato S, Simison M, Tanaka F, Luo Y, Whaling I, Hong EL, Lee BT, Sandstrom R, Rynes E, Nelson J, Nishida A, Ingersoll A, Buckley M, Frerker M, Kim DS, Boley N, Trout D, Dobin A, Rahmanian S, Wyman D, Balderrama-Gutierrez G, Reese F, Durand NC, Dudchenko O, Weisz D, Rao SSP, Blackburn A, Gkountaroulis D, Sadr M, Olshansky M, Eliaz Y, Nguyen D, Bochkov I, Shamim MS, Mahajan R, Aiden E, Gingeras T, Heath S, Hirst M, Kent WJ, Kundaje A, Mortazavi A, Wold B, Cherry JM. The ENCODE Uniform Analysis Pipelines. bioRxiv [Preprint]. 2023 Apr 6:2023.04.04.535623. doi: 10.1101/2023.04.04.535623. PMID: 37066421; PMCID: PMC10104020.

[3] Timothy L. Bailey, James Johnson, Charles E. Grant, William S. Noble, "The MEME Suite", Nucleic Acids Research, 43(W1):W39-W49, 2015.

[4] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011

prediction (see figures 1 and 2 below). We also found that regularization parameter that was best for prediction was 10^{-3}. Additionally, we also tried to change the rate of positive labled samples against negative labeled samples. We tried to train a model where half the labels were positive and another model where a fifth of the labels were positive. We concluded that when the model trains on a half positive dataset the results are better (see figure 3). We received these results for a single protein, SRSF1, and applied the same for the rest of the datasets. We used ROC curves to measure prediction results, we also cross validated training set to find optimal regularization parameter for prediction. To validate our results we also trained an SVM model on a randomly labeled dataset to see if the model learned anything and we also compared results to RBPmap predictions (see figure 3).

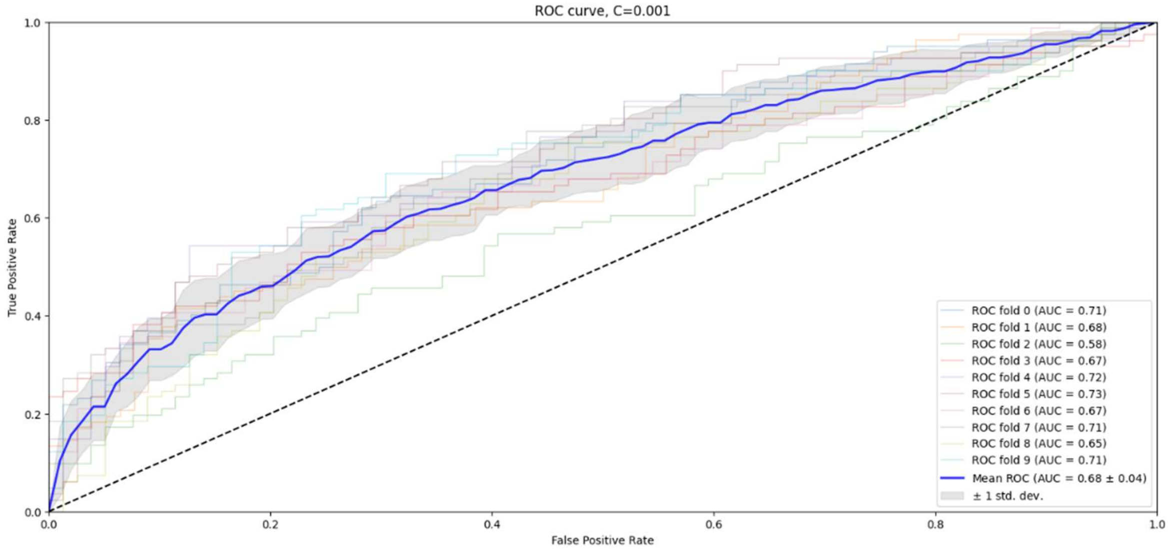

Figure 1- ROC curves of an SVM model using unit vectors representation for different folds in cross validation. Blue curve is the mean ROC curve of all folds and the gray zone around it is one standard deviation from mean. Regularization parameter here is 10^{-3}.

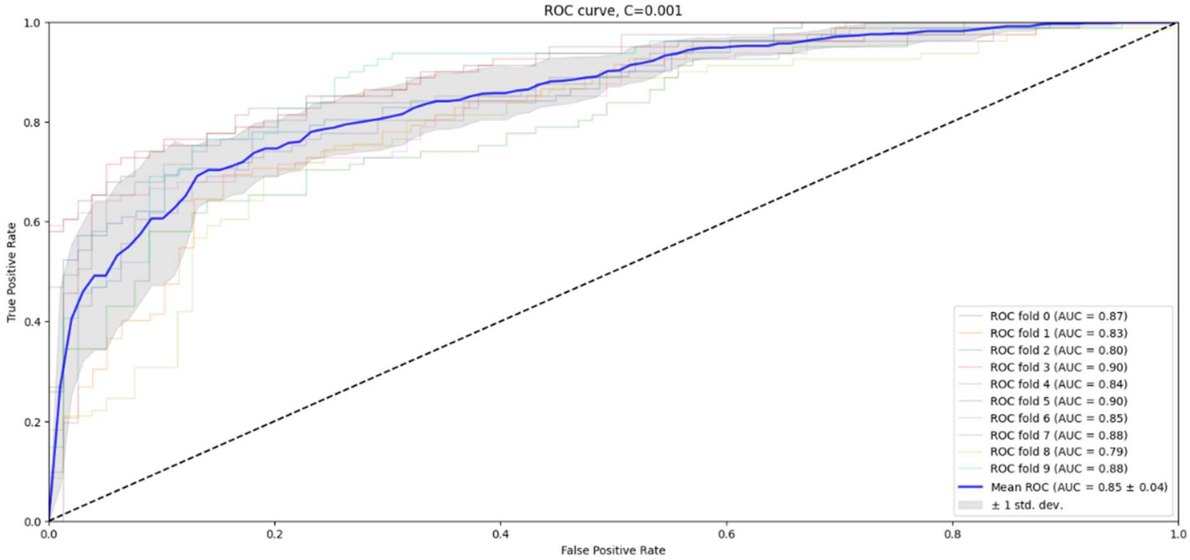

Figure 2- ROC curves of an SVM model using one-hot encoding for different folds in cross validation. Blue curve is the mean ROC curve of all folds and the gray zone around it is one standard deviation from mean. Regularization parameter here is 10^{-3}.

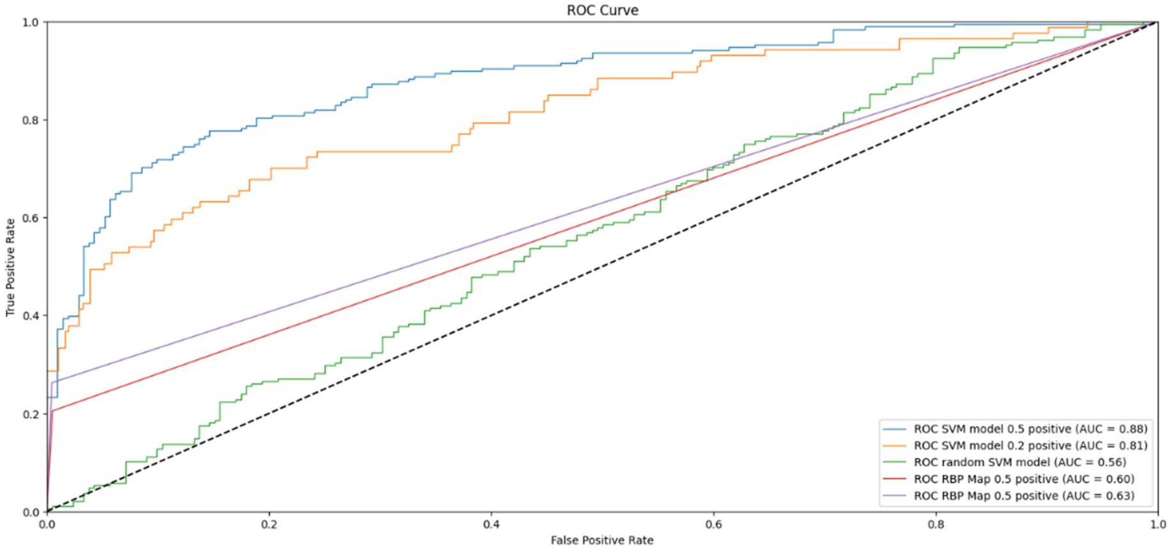

Figure 3- ROC curves for different models, an SVM model trained on a half positive dataset, an SVM model trained on a fifth positive dataset, an SVM model trained random labeled datasets and two curves corresponding to RBPmap predictions.

Next, we experimented with more proteins: SRSF1, PUM2, QKI, RBM5 and HNRNPL. For each protein we used an SVM model with one-hot encoding, optimized regularization parameter and a dataset where half its samples are positive. We compared results against RBPmap and in general we see that the SVM model predicts quite well, at least as good as RBPmap and in some cases significantly better. Current results of RBPMap were better than before because we used high stringency threshold instead of the default threshold. Below we can observe some results.

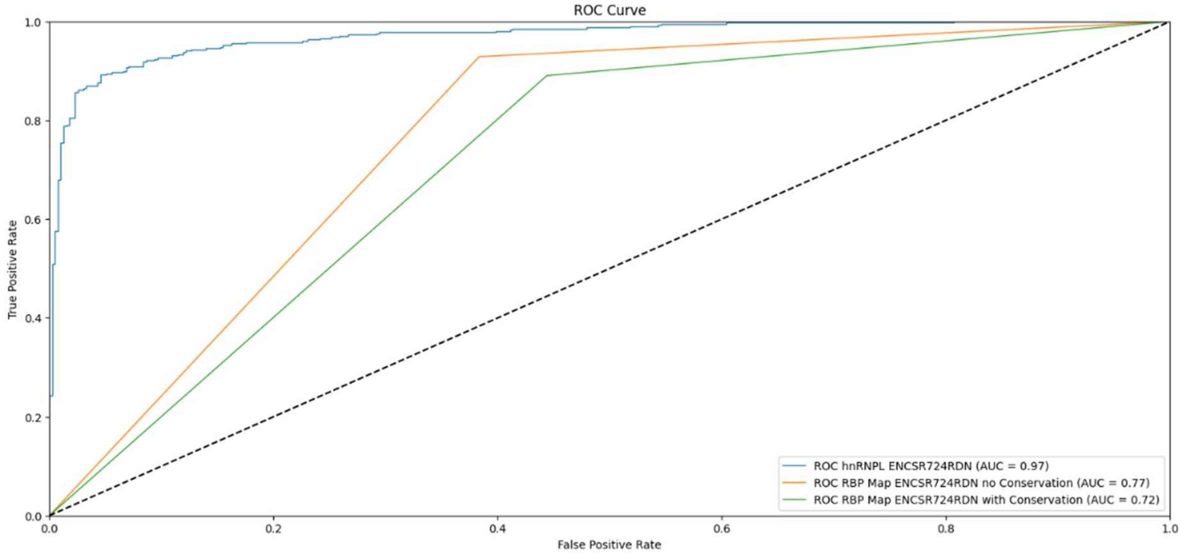

Figure 4- ROC curves of 3 different models predicting on hnRNPL. SVM model in blue, RBPmap without conservation in orange and RBPmap with conservation in green.

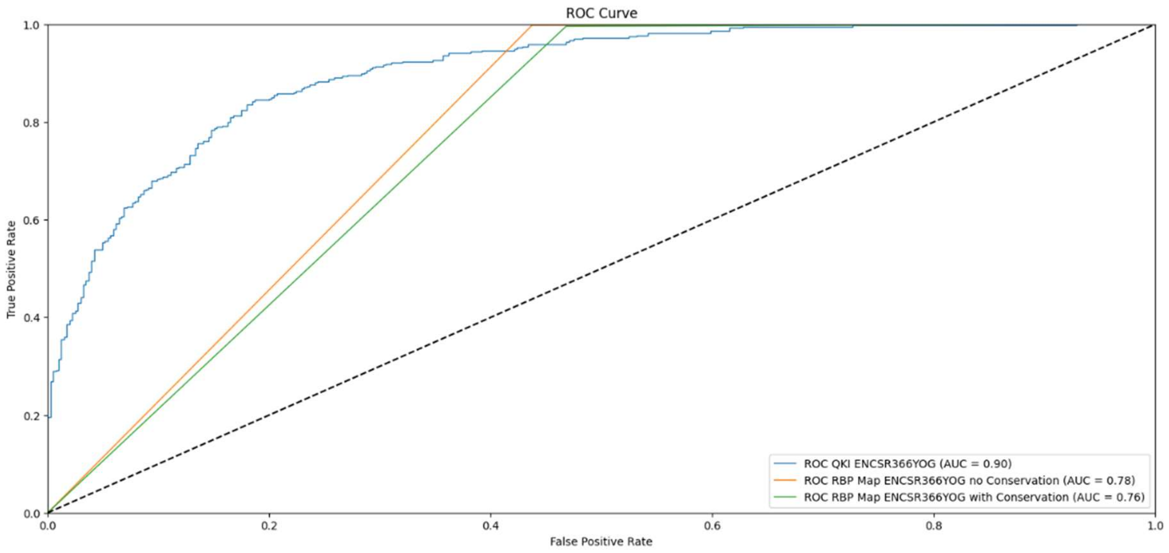

Figure 5- ROC curves of 3 different models predicting on QKI. SVM model in blue, RBPmap without conservation in orange and RBPmap with conservation in green.

Lastly, we observed the models fitted weights to try and analyze what the model learned. This task proved more difficult than the rest because it is not a trivial puzzle- understanding what the model learned, especially when using models with many features. We have tried to analyze the data in various ways, but we are currently trying to understand the model in a way that might prove instrumental for future purposes. Below is an example of a single SVM model we fitted and its coefficients along the position on the sequence.

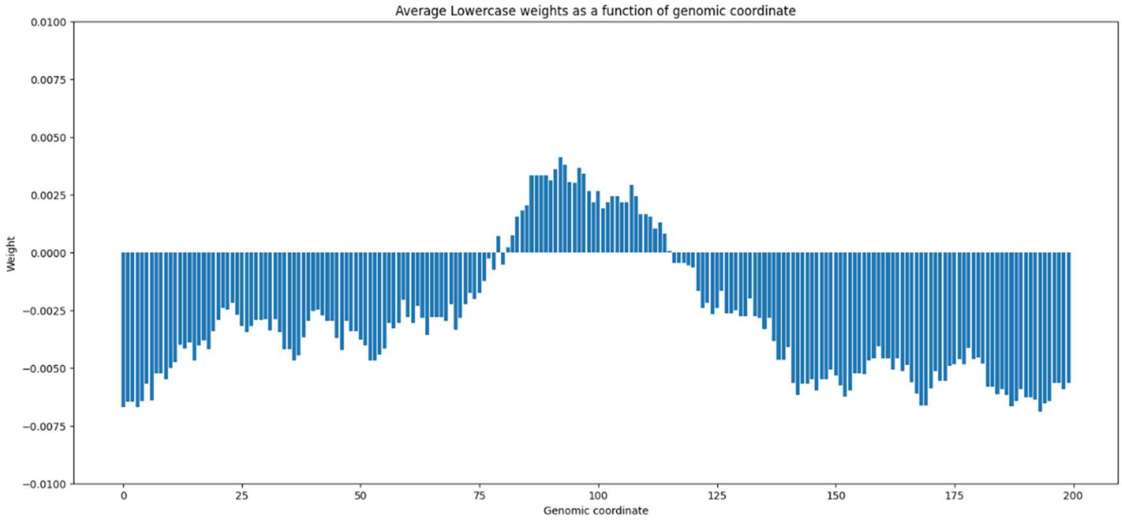

Figure 6- The weight of each location in the sequence. We average over all 4 positional weights corresponding to A, C, G and T. Lowercase in the title refers to the fact that we have 8 different letters, A and a, C and c etc. The case of the letter tells us if it is an exon region or an intron region (uppercase=exon region).

Conclusion

In this project we addressed a fundamental problem in biology and tried to explore it in a new approach. It is worth mentioning that there are many works that try to do similarly, predict binding sites using machine learning approaches. Although, the focus in this project is not to successfully predict a protein's binding site but to successfully understand what the model learned and use this to get a more profound understanding of the biological mechanism. We were first introduced with motif-based predictions that give good predictions to some extent, yet it is quite clear that the predictions are not perfect and mislabel occasionally. We then decided to look at the data with a fresh perspective, neglecting motif-based predictions in the hopes of finding new alternatives for binding sites predictions. The prediction results using the SVM model seemed promising as the model predicts binding sites with high accuracy. In addition, it seems reasonable to suggest that the model learned a decision rule that has nothing to do with the motif because we never gave the model the proteins motif as a feature. In contrast to these promising results, we are still not sure how to interpret these results and this is yet to be uncovered.

For future research directions, there many things we can try. First, before all other options, it is worth trying to understand what the SVM model learned yet again as the results clearly show that something has been learned. Secondly, based on biological reasoning it may be proved useful to use the secondary structure of the sequence. The secondary structure tells us if the sequence is single stranded in a loop or double stranded linearly. We know that proteins tend to bond to loops more often so we think that adding secondary structure features might benefit in prediction. Another approach is to try and train deep learning models that are not feature based. The advantage here is that we don't need to guess the features initially. Also, for some models like transformers, there is a lot of work done on model interpretation, meaning that it may be easier to understand what a transformer learned using standard techniques. Lastly, it may be worth trying to predict binding sites using pretrained models like DNABERT or the nucleotide transformer that were trained on a wide range of domain knowledge in the field of biology and are likely to give profound conclusions regarding binding site predictions.

Code

```
import numpy as np

def prepare_bed_file(input_filepath_positive,
input_filepath_negative, output_filepath='output.txt',
num_of_positive_samples=2000, shuffle=True):
    positive_file = open(input_filepath_positive, 'r')
    positive_lines = positive_file.readlines()
    num_of_positive_samples = min(num_of_positive_samples,
len(positive_lines))
    positive_file.close()
    positive_lines = sort_and_clean_data(positive_lines)

    negative_file = open(input_filepath_negative, 'r')
    negative_lines = negative_file.readlines()
    negative_file.close()
    negative_lines = sort_and_clean_data(negative_lines)

    top_lines = positive_lines[:num_of_positive_samples]
    bottom_lines =
np.flip(np.flip(negative_lines)[:num_of_positive_samples])
    dataset_lines = np.concatenate([top_lines, bottom_lines])
    if shuffle:
        order = np.array(range(len(dataset_lines)))
        np.random.shuffle(order)
        dataset_lines = [dataset_lines[index] for index in order]

    new_file = open(output_filepath, 'w')
    new_file.writelines(dataset_lines)
    new_file.close()


def sort_and_clean_data(lines):
    lines_to_write = []
    q_values = []
    for line in lines:
        elements = line.split('\t')
        location = int(elements[1]) + (int(elements[2]) -
int(elements[1])) // 2
        elements[1] = str(location-100)
        elements[2] = str(location + 100)
        q_values.append(float(elements[7]))
        new_line = '\t'.join(elements)
        lines_to_write.append(new_line)
    order = np.flip(np.argsort(q_values))
    sorted_lines_to_write = np.array([lines_to_write[index] for index
in order])
    return sorted_lines_to_write

def prepare_data(bed_filepath, fasta_filepath,
output_filepath='output.txt', threshold=0):
    bed_file = open(bed_filepath, 'r')
    bed_lines = bed_file.readlines()
    bed_file.close()

    fasta_file = open(fasta_filepath, 'r')
    fasta_lines = fasta_file.readlines()
    fasta_lines = [line for line in fasta_lines if not
line.startswith(">")]
    fasta_file.close()

    new_file = open(output_filepath, 'w')

    for bed_line, fasta_line in zip(bed_lines, fasta_lines):
        bed_elements = bed_line.split('\t')
        label = int(float(bed_elements[7]) > threshold)
        new_line = str(label) + '\t' + fasta_line
        new_file.write(new_line)
    new_file.close()


def label_rbp_map_results(input_rbp_filepath, input_ds_filepath,
input_bed_filepath, output_filepath='output.txt', threshhold=1e-4):
    bed_file = open(input_bed_filepath, 'r')
    bed_lines = bed_file.readlines()
    bed_lines = [bed_lines[i].split('\t')[1] for i in
range(len(bed_lines))]
    bed_file.close()

    ds_file = open(input_ds_filepath, 'r')
    ds_lines = ds_file.readlines()
    true_labels = [ds_lines[i].split('\t')[0] for i in
range(len(ds_lines))]
    ds_file.close()

    rbp_file = open(input_rbp_filepath, 'r')
    rbp_lines = rbp_file.readlines()
    rbp_file.close()
    rbp_filtered_lines = [line for line in rbp_lines if
```

```
                          (line.startswith(("chr", "No motifs
found")) or line[0].isdigit())]
    coordinates = []
    labels = []
    min_p_value = 1
    for line in rbp_filtered_lines:
        if line.startswith("chr"):
            if coordinates:
                labels.append(int(min_p_value < threshhold))
                min_p_value = 1
            start = line.find(':') + 1
            end = line.find('-', start)
            if start == 0 or end == -1:
                coordinates.append(-1)
            else:
                coordinates.append(line[start:end].strip())
        elif line.startswith("No motifs found"):
            continue
        else:
            elements = line.split('\t')
            min_p_value = min(min_p_value, float(elements[5]))
    labels.append(int(min_p_value < threshhold))
    new_file = open(output_filepath, 'w')
    new_file.write('RBPmap\tTrue\n')
    for (line, true_label) in zip(bed_lines, true_labels):
        try:
            index = coordinates.index(line)
            new_file.write(str(labels[index]) + '\t' + true_label +
'\n')
        except ValueError:
            new_file.write('0\t' + true_label + '\n')
    new_file.close()

def make_file_uppercase(input_filepath, output_filepath):
    file = open(input_filepath, 'r')
    lines = file.readlines()
    lines = [line.upper() for line in lines]
    file.close()
    new_file = open(output_filepath, 'w')
    new_file.writelines(lines)
    new_file.close()

def classify_file_uppercase_lowercase(input_filepath,
output_filepath):
    file = open(input_filepath, 'r')
    lines = file.readlines()
    lines = [transform_string(line) for line in lines]
    file.close()

    new_file = open(output_filepath, 'w')
    new_file.writelines(lines)
    new_file.close()

def transform_char(char):
    if char.isupper():
        return 'X'
    elif char.islower():
        return 'x'
    else: return char
def transform_string(s):
    return ''.join(transform_char(char) for char in s)
```